WINNER-TAKE-ALL BOTTLENECKS ENFORCE DISENTANGLED SYMBOLIC REPRESENTATIONS IN MULTI-TASK LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Winner-take-all (WTA) networks constitute a central circuit motif in cortical networks of the brain. In addition, WTA-like activations are abundant in modern deep learning models in the form of the softmax activation for example in attention layers of transformers. While their role in the extraction of latent factors has been studied for relatively simple generative models, their role in the context of highly non-linearly entangled latent factors has remained elusive. In this article, we show that a WTA bottleneck within a deep neural network can enforce under certain well-defined conditions the extraction of categorical latent factors of the data in a multi-task learning setup. In particular, we prove that under specific conditions, including that the structure of the WTA bottleneck matches the structure of the latent factors and the model solves all tasks perfectly, the emerging representation is a structured permutation of those latent factors. When the structures do not match, we empirically demonstrate on two datasets that a symbolic representation often emerges when the multi-task learning objective is optimized close to its global optimum. This representation is a weaker form of structured permutation in which multiple WTA output neurons may jointly encode a single latent factor category. We additionally evaluate the generalization behavior of the acquired symbolic representation. Our proposed model provides insights into the generalization capabilities of deep neural networks with WTA-like components and may serve as an interface between symbolic and subsymbolic AI systems.

1 INTRODUCTION

Winner-take-all (WTA) networks have been proposed to constitute a central circuit motif in cortical networks (Douglas et al., 1989). In biology, WTA motifs arise from lateral inhibition where excitatory neurons project to inhibitory neuron populations which in turn provide inhibitory feedback to these excitatory cells (Avermann et al., 2012). Theoretical studies have emphasized the role of WTA motifs in the context of learning, showing that local spike timing-dependent plasticity (STDP) in WTA architectures implement an expectation-maximization algorithm in simplified models that is able to extract the hidden causes of sensory data (Nessler et al., 2009; Legenstein et al., 2017; Jonke et al., 2017). Even earlier, WTA architectures were investigated in the context of competitive learning (Rumelhart & Zipser, 1985). Apart from their biological relevance, WTA-like activations are abundant in modern deep learning models in the form of the softmax activation, which can be viewed as a differential version of the hard WTA function. Besides its role as a normalizer when computing the probability distribution over discrete classes in the final classification layer of classical feed-forward and recurrent network architectures, it is also a central component of attention layers in transformers (Vaswani et al., 2017).

While single-layer WTA networks have been shown to enable the extraction of latent factors from sensory input under the assumption of relatively simple generative models (Nessler et al., 2009; Legenstein et al., 2017), their role in the context of observations that arise from highly non-linearly entangled latent factors has remained elusive. In this article, we show that a WTA bottleneck within a deep neural network — that is, a layer consisting of several WTA activations — enforces under certain conditions the extraction of categorical latent factors of the data in a multi-task learning setup. In particular, we consider a deep neural network with a WTA bottleneck in its penultimate

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

layer that observes an arbitrary information-preserving transformation of categorical latent factors $\mathbf{z}$ at its inputs. We prove that, when the network was trained to perfectly solve a large enough number of linear classification tasks on the latent factors $\mathbf{z}$, then the representation $\hat{\mathbf{z}}$ of the WTA bottleneck is a permutation of $\mathbf{z}$ that is further constrained by the structure of the WTA bottleneck (referred to as *structured permutation*).

We then show empirically on two data sets, that similar representations also emerge for architectures that do not fully comply with the assumptions of our theorem. The resulting representation is disentangled (Higgins et al., 2018) and highly symbolic, where a single neuron or a population of neurons encodes the presence of a single abstract feature such as a specific object, color, or position. Such representations are often called factorized representations (Johnston & Fusi, 2023; Bengio et al., 2013). We refer to this specific type of representation where latent features are encoded by populations of neurons as *symbolic representations*. It has been argued that disentangled representations benefit generalization (Bengio et al., 2013; Johnston & Fusi, 2023) (but see (Locatello et al., 2019)). We show that it also holds for the acquired symbolic representations by considering generalization to various test settings. Our approach to disentangled (symbolic) representation learning has the potential to link subsymbolic and symbolic AI and could thus play a key role in neurosymbolic AI.

2 Related work

Learning in WTA architectures First ideas regarding learning in competitive neural ensembles date back to Rumelhart & Zipser (1985; 1986). They considered a single-layer neural network where the neuron with the largest activation (the winner) updates its weights, leading to a clustering of the input space. Later, this idea was investigated in the context of theoretical neuroscience research. In Nessler et al. (2009), it was shown that STDP enables WTA circuits to extract the latent features of its inputs in a well-defined graphical model. This work was generalized to networks with more biologically plausible soft inhibition (Jonke et al., 2017; Legenstein et al., 2017) and to sheets of neurons with lateral inhibition implementing multiple WTA circuits (Bill et al., 2015). WTA architectures have also been studied in the context of hierarchical object recognition (Riesenhuber & Poggio, 1999) and assembly creation for long term memory (Legenstein et al., 2018). See (Maass, 2000) for an analysis of the computational power of WTA activations.

Disentangled representations through multi-task learning A large number of methods for disentangled representation learning (DRL) exist, see (Bengio et al., 2013; Wang et al., 2024) for reviews. Recently, the role of multi task learning for DRL has been studied (Johnston & Fusi, 2023; Vafidis et al., 2025). In particular, Vafidis et al. (2025) have proven that nonlinearly entangled continuous latent factors can be disentangled through multi-task learning given a set of sufficiently diverse tasks that depend linearly on the latent factors. This result is remarkable as it provides a theoretical guarantee for the emergence of disentangled representations. We provide a related result for categorical latent variables with an additional aspect. We find that, when multi-task learning is performed on a WTA bottleneck, then one can not only recover a linear combination of categorical latent variables, but the variables themselves. This result complements work on the identifiability of latent variables in deep generative models (Lee & Gu, 2026; Kivva et al., 2021; 2022).

3 Problem formulation

We now formally define the learning setup considered in this article.

Latent features: We consider a set of m latent categorical random variables $Z_1, \dots, Z_m$, where Z_i can take values in $\{1, \dots, l_i\}$, with $l_i \in \mathbb{N}$, $l_i \geq 2$ as the number of categories. The corresponding one-hot vectors are $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}$, with $\mathbf{z}^{(i)} \in \{0,1\}^{l_i}$. The total latent vector $\mathbf{z}$ is then the concatenation of these vectors $\mathbf{z} = (\mathbf{z}^{(1)}; \dots; \mathbf{z}^{(m)})^\top \in \{0,1\}^l$ with $l = \sum_i l_i$ (see matrix C in Fig. 1b). We denote by $\mathcal{Z}$ the set of all possible total latent vectors defined in that way.

Observables: We observe a nonlinearly transformed version of $\mathbf{z}$, i.e., we a assume an injective function $\Phi : \{0,1\}^l \to \mathbb{R}^d$, see Fig. 1a. For a given $\mathbf{z}$, the observable vector $\mathbf{x}$ is then

$$\mathbf{x} = \Phi(\mathbf{z}). \tag{1}$$

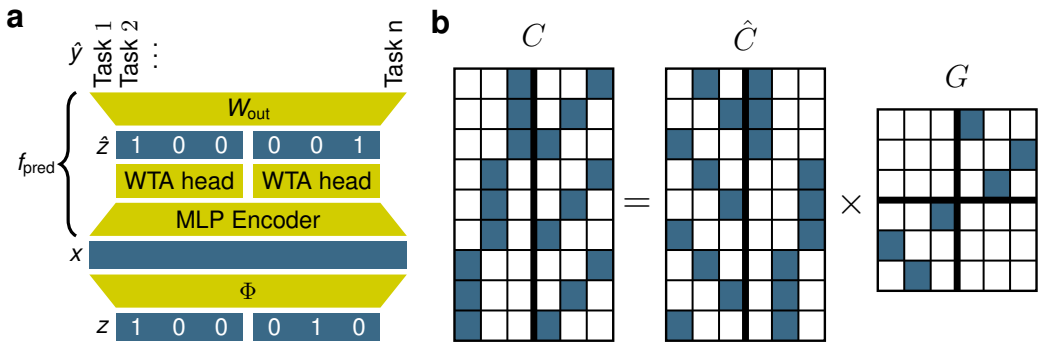

Figure 1: **Theoretical framework. a)** Network setup, showing representations (blue) and mappings (olive). Latent vector $\mathbf{z}$ (bottom) examplified for two latent variables with three categories each. Each latent is coded as a one-hot vector. The latents are mapped through an injective function Φ to an entangled represenation $\mathbf{x}$. The WTA encoder f_{enc} maps $\mathbf{x}$ to a representation $\hat{\mathbf{z}}$, which is constrained by a multi-WTA head. A readout layer (top) is applied to $\hat{\mathbf{z}}$ and trained to solve n tasks. **b)** Structured permutation. C illustrates the "code" matrix for two latent variables with three categories each (blue indicates value 1, white value 0). The rows together encode all possible latent vectors $\mathbf{z}$. The columns of matrix $\hat{C}$, are a structured permutation of the columns of C in the sense that the latent variables are permuted and categories within the latent variables are permuted. C can be obtained from $\hat{C}$ by $C = \hat{C}G$ for structured permutation matrix G.

WTA Encoder: We consider a winner-take-all (WTA) encoder $f_{\text{enc}} : \mathbb{R}^d \to \{0,1\}^l$ which takes as input $\mathbf{x}$ and outputs $\hat{\mathbf{z}} = (\hat{\mathbf{z}}^{(1)}; \ldots; \hat{\mathbf{z}}^{(m)})^\top$, see Fig. 1a. $\hat{\mathbf{z}}$ consists of m binary one-hot vectors which are concatenated together and $\hat{\mathbf{z}}$ has the same structure as the latents $\mathbf{z}$ (we assume identical structure in the proof, although the order of the l_i's can be permuted w.l.o.g. Experiments will be performed with differing structures). The WTA encoder f_{enc} consists of a feed-forward neural network $f_{\text{MLP}} : \mathbb{R}^d \to \mathbb{R}^l$ resulting in an output $\mathbf{a} \in \mathbb{R}^l$ and m WTA heads f_{WTA} applied on top of $\mathbf{a}$. The output $\mathbf{a}$ of f_{MLP} is grouped into m output groups $\mathbf{a}^{(i)} \in \mathbb{R}^{l_i}$. One WTA function $f_{\text{WTA}} : \mathbb{R}^{l_i} \to \{0,1\}^{l_i}$ is applied to each of these groups individually, where f_{WTA} is defined as

$$f_{\text{WTA}}(\mathbf{a}^{(i)})_j = \begin{cases} 1, & \text{if } j = \arg\max_{k \in \{1,\ldots,l_i\}} \mathbf{a}_k^{(i)}, \\ 0, & \text{otherwise,} \end{cases} \qquad j = 1, \ldots, l_i. \tag{2}$$

The function f_{WTA} sets its output corresponding to the maximum value of $\mathbf{a}^{(i)}$ to 1 and all other values to 0 (we assume that there is a unique largest component). The output of the i-th WTA function is denoted by $\hat{\mathbf{z}}^{(i)}$ and the output vectors of all WTAs are concatenated in one vector $\hat{\mathbf{z}} \in \{0,1\}^l$

$$\hat{\mathbf{z}}^{(i)} = f_{\text{WTA}}(\mathbf{a}^{(i)}), \quad \hat{\mathbf{z}} = (\hat{\mathbf{z}}^{(1)}; \ldots; \hat{\mathbf{z}}^{(m)})^\top. \tag{3}$$

We refer to this group of WTA heads as a multi-WTA head.

Readout layer: The representation $\hat{\mathbf{z}}$ serves as input for a fully connected readout layer $f_{\text{readout}} : \{0,1\}^l \to (0,1)^n$ with n sigmoidal outputs, which produce the output $\hat{\mathbf{y}} \in \mathbb{R}^n$ according to

$$\hat{\mathbf{z}}_{\text{out}} = W_{\text{out}} \hat{\mathbf{z}} \tag{4}$$

$$\hat{\mathbf{y}} = \sigma(\hat{\mathbf{z}}_{\text{out}}), \tag{5}$$

with weight matrix $W_{\text{out}} \in \mathbb{R}^{n \times l}$, see Fig. 1a. Note that we do not use a bias vector in this layer. It could be incorporated, but as we will see below, it is not needed. σ denotes the component-wise logistic sigmoid function. This layer will predict the posterior probabilities for n linear tasks, see next paragraph. We refer to the encoder together with the readout layer as the predictor network $f_{\text{pred}} : \mathbb{R}^d \to (0,1)^n$, $f_{\text{pred}} := f_{\text{readout}} \circ f_{\text{enc}}$.

Definition of linear tasks: We define n linear binary classification tasks based on the latent categorical variable vector $\mathbf{z}$. The i-th task assumes that $\mathbf{z}$ is originating from one of two classes

indicated by $A_i = 1$ (class 1) and $A_i = 0$ (class 0) with respective prior probabilities $P(A_i = 1)$ and $P(A_i = 0)$. If $\mathbf{z}$ originated from class 1 (resp. 0), it is assumed that each $\mathbf{z}^{(j)}$ was drawn from a categorical distribution with parameters $\mathbf{p}^{(i,j)}$ (resp. $\mathbf{q}^{(i,j)}$). Again, we write $\mathbf{p}^{(i)}$ (resp. $\mathbf{q}^{(i)}$) for the concatenation of these parameter vectors, i.e., $\mathbf{p}^{(i)} = (\mathbf{p}^{(i,1)}; \ldots; \mathbf{p}^{(i,m)})^\top$. Hence, the tuple $(\mathbf{p}^{(i)}, \mathbf{q}^{(i)}, P(A_i = 1))$ defines task i. The task of the readout layer is to predict the posterior probability $P(A_i = 1|\mathbf{z})$ that some $\mathbf{z}$ belongs to class $A_i = 1$. It is convenient to stack the posteriors of all tasks together. We thus define $P(A|\mathbf{z}) \in [0, 1]^n$ as a vector with the i-th component given by $P(A_i = 1|\mathbf{z})$. A standard derivation shows that this posterior can be written as $P(A|\mathbf{z}) = \sigma(W\mathbf{z} + \mathbf{b})$, with $W = [w_{ij}]_{\substack{i=1,\ldots,n \\ j=1,\ldots,l}}$ and $\mathbf{b} = (b_1, \ldots, b_n)^\top$ for $w_{ij} = \ln \frac{p_j^{(i)}}{q_j^{(i)}}$ and $b_i = \ln \frac{P(A_i=1)}{P(A_i=0)}$, see Appendix A.1. Note however that, since $\mathbf{z}$ consists of a constant number of 1-entries, $\mathbf{b}$ can be incorporated into W, so

$$P(A|\mathbf{z}) = \sigma(W\mathbf{z}), \tag{6}$$

for a suitable re-definition of W (see Appendix A.1). This shows that we can write $P(A|\mathbf{z})$ as a linear mapping of the latent vector $\mathbf{z}$ transformed by a sigmoid function.

Structured permutation: We will show that under multi-task learning and well defined conditions, the output of the multi-WTA head evolves a structured permutation of the latent variables. These variables can be exactly reconstructed from the WTA output, but the reconstruction can only be done up to some permutation, as the order of latents and the order of categories within the latents is arbitrary. That means, that in the representation $\hat{\mathbf{z}}$, the latent variables can be permuted and within the latent variables, the categories can be permuted relative to $\mathbf{z}$. Also, a variable Z_i can only be represented by WTA head j if its number of categories matches the number of outputs of the WTA gate. It is therefore a structured permutation that depends on the structure of the latents. We illustrate a structured permutation in Fig. 1b and formally define it in Appendix A.1.

4 Theoretical Results

The predictor network f_{pred} is trained to predict the posterior probabilities $P(A|\mathbf{z})$. Note that due to our assumption that the nonlinear entanglement transformation Φ is injective and $\mathbf{x} = \Phi(\mathbf{z})$, we have $P(A|\mathbf{z}) = P(A|\mathbf{x})$. In other words, all information is available to predict this posteriors from the entangled representation in $\mathbf{x}$. Assume that the predictor network has sufficient capacity and has been perfectly trained, i.e., it is an optimal predictor. Formally, we say that a predictor f_{pred} is optimal (with respect to posterior distributions $P(A|\mathbf{z})$ and entanglement transformation Φ) if

$$f_{\text{pred}}(\Phi(\mathbf{z})) = P(A|\mathbf{z}) \qquad \text{for all } \mathbf{z} \in \mathcal{Z}. \tag{7}$$

Following the linear decodability approach of Vafidis et al. (2025), we first show that in this case, $\mathbf{z}$ is linearly decodable from the readout activations $\hat{\mathbf{z}}_{\text{out}}$ (Equation 4) as well as from the multi-WTA output $\hat{\mathbf{z}}$ if W has full rank (recall that W defines the tasks, see Equation 6). As defined above, $\hat{\mathbf{y}} = f_{\text{pred}}(\Phi(\mathbf{z}))$ and by assumption, f_{pred} is an optimal predictor. We combine Equation 6 and Equation 7

$$\hat{\mathbf{y}} = \sigma(W\mathbf{z}). \tag{8}$$

We see that the latent vector $\mathbf{z}$ is encoded in the posterior prediction $\hat{\mathbf{y}}$. We know from Equation 5 that $\hat{\mathbf{y}} = \sigma(\hat{\mathbf{z}}_{\text{out}})$. Therefore, we combine Equation 8 and Equation 5 and get

$$\sigma(\hat{\mathbf{z}}_{\text{out}}) = \sigma(W\mathbf{z}) \quad \Leftrightarrow \quad \hat{\mathbf{z}}_{\text{out}} = W\mathbf{z}. \tag{9}$$

To solve Equation 9 for $\mathbf{z}$, we can use the pseudo-inverse and get

$$\mathbf{z} = (W^\top W)^{-1} W^\top \hat{\mathbf{z}}_{\text{out}}. \tag{10}$$

Hence, $\mathbf{z}$ is linearly decodable from $\hat{\mathbf{z}}_{\text{out}}$ if $W^\top W$ is invertible, that is, if W has full rank. Since $\hat{\mathbf{z}}_{\text{out}} = W_{\text{out}}\hat{\mathbf{z}}$, this implies

$$\mathbf{z} = (W^\top W)^{-1} W^\top W_{\text{out}} \hat{\mathbf{z}}. \tag{11}$$

Hence, $\mathbf{z}$ can be obtained by a linear mapping from $\hat{\mathbf{z}}$ if W has full rank, which holds for each $\mathbf{z} \in \mathcal{Z}$ and its corresponding encoding $\hat{\mathbf{z}}$. We can arrange all $\mathbf{z} \in \mathcal{Z}$ in a code matrix C where each row is

one latent vector. This can be done using an arbitrary order of the the rows, but for simplicity we can imagine it in lexicographical order as shown for an example in Fig. 1b. Each vector has length $l = \sum_i l_i$ and there are $p := \prod_i l_i$ such vectors. Hence we have $C \in \{0,1\}^{p \times l}$. We define a similar code matrix $\hat{C}$ for the encodings $\hat{\mathbf{z}}$. From Equation 11, we know that we can write

$$C = \hat{C}G \tag{12}$$

where $G = ((W^\top W)^{-1} W^\top W_{\text{out}})^\top$ (see also Fig. 1b).

Our main theoretical result is that the multi-WTA representation $\hat{\mathbf{z}}$ obtained in this way is a structured permutation of the latent variables $\mathbf{z}$, i.e., the columns of $\hat{C}$ are a structured permutation of the columns of C. We say that a matrix C has an m-fold one-hot column-block structure if every row of C consists of m consecutive blocks where each block has exactly one 1 entry and the other entries in the block are 0. Both matrices C and $\hat{C}$ have an m-fold one-hot column-block structure by design. Since both matrices have the same structure and dimensions, and Equation 12 must hold for all rows in C, $\hat{C}$ must also contain all rows from C. The proof of the following Theorem 1 is provided in Appendix A.2 and consists of two steps. In the first step, we show that Equation 12 can be transformed into $C = \hat{C}Q$, where Q is a matrix consisting only of 0's and 1's. In the second step, we can show that Q is a structured permutation matrix. It follows that C is a structured permutation of $\hat{C}$, and vice versa.

Theorem 1. *Let m and $l_1, ..., l_m$ be natural numbers with $l_i \geq 2$. Let $l = \sum_{i=1}^m l_i$ and $p = \prod_{i=1}^m l_i$. Let $C, \hat{C} \in \{0,1\}^{p \times l}$ be two matrices with an m-fold one-hot column-block structure with arbitrary column-block sizes of $l_1, ..., l_m$. Both C and $\hat{C}$ consist of all possible rows of that type. Assume that $C = \hat{C}G$ for some matrix $G \in \mathbb{R}^{l \times l}$. Then, the columns of $\hat{C}$ are a structured permutation of the columns of C with the restriction that only column blocks with the same sizes can be permuted.*

5 Experiments

Our analysis in Section 4 revealed that categorical latent variables can be recovered from the multi-WTA bottleneck of an optimal multi-task predictor. This raises the question whether such a predictor can be trained. After all, WTA nonlinearities are not differentiable and potentially hard to train as a representation has to be found by the optimization algorithm that maps categories of each latent variable to a single WTA gate. A potential solution could be to replace hard WTA gates by softmax units. However, our simulations showed that this approach tends to produce blurry output distributions at the WTA gates (i.e., there is no clear winner), which turned out to be insufficient for the extraction of a structured permutation of latent variables. We therefore adopted an approach with a hard multi-WTA bottleneck. To resolve the gradient problem and to encourage exploration, we used the Straight-Through Gumbel-Softmax Estimator with Gumbel noise (Jang et al., 2017) in combination with a high learning rate and a high temperature coefficient (see Appendix A.3).

5.1 Structured permutations from nonlinearly entangled latent variables

In the following experiment, we tested out theoretical results empirically. In order to obtain data with known ground-truth latents, we generated inputs $\mathbf{x}$ from given latent vectors $\mathbf{z}$ via a nonlinear transformation Φ. We tested our theory in a first setup with 5 latent variables $Z_1, \ldots, Z_5$, each with 5 categories. The latent variables were sampled from independent categorical distributions with uniform probabilities. The transformation Φ was obtained by a Multi-Layer Perceptron (MLP) with 4 hidden layers and Leaky ReLU nonlinearities (see Appendix A.3). The resulting representation vectors $\mathbf{x} \in \mathbb{R}^{70}$ established an entangled representation (see Section 5.3 for a discussion). The multi-WTA encoder was implemented as an MLP with 4 hidden layers and 5 WTA heads. We used 5 outputs for each WTA head to get a WTA representation $\hat{\mathbf{z}}$ with the same structure as the latent variables, consistent with our theory. Since the WTA structure matches the latent structure as assumed in the theory, we refer to this setup as the *matched* setup.

A single fully connected layer was applied to $\hat{\mathbf{z}}$ with n independent logistic sigmoids as output nonlinearities, resulting in the task prediction vector $\hat{\mathbf{y}} \in (0,1)^n$ (n is the number of tasks). We generated a set of $n = 30$ tasks by randomly sampling a tuple $(\mathbf{p}^{(i)}, \mathbf{q}^{(i)}, P(A_i = 1))$ for each task i (see Section 3). The resulting posterior class probabilities were used as targets to minimize

the binary cross entropy of the network output. We trained the network on $100,000$ input samples for $1,000$ epochs and tested the network on $10,000$ samples. We performed this training procedure 20 times with different random seeds. The tasks were solved perfectly in 15 out of the 20 runs (Mean Absolute Error (MAE) less than 10^{-6} on the test set). Consistent with our theory, those models which solved the tasks evolved a structured permutation of $\mathbf{z}$ in $\hat{\mathbf{z}}$ (see Appendix A.6). In other words, each WTA head represented one latent variable with categories represented in permuted order with respect to $\mathbf{z}$. The mapping is illustrated in Appendix A.6, Fig. A1.

We next examined the number of tasks necessary during the multi-task learning to obtain a structured permutation of latents. For this we trained the model on 1, 5, 10, 15, 20, and 25 tasks (20 different random seeds each). Consistent with the theory, with 25 tasks, a structured permutation emerged if the model was able to solve all tasks perfectly (18 out of 20 models). However, this was also the case for 20 tasks (14 out of 20 models solved the tasks perfectly) and for 15 tasks (2 out of 20 models), indicating that fewer tasks suffice in some setups. When less than 15 tasks were used, training did not converge to optimal performance (see Appendix A.6 for detailed results).

5.2 Symbolic representations from nonlinearly entangled latent variables

In practice one does not know the number of categories of latent factors. Therefore, in the following experiments, we investigated whether a symbolic representation, a weaker form of a structured permutation, emerges when the structure of the factors does not match the WTA structure. A structured permutation is no longer achievable if the structures do not match. Therefore, we need the concept of a symbolic representation.

Symbolic representations: We first compare notions of disentangled representations and specifically define our notion of symbolic representations. Vafidis et al. (2025) and Ostojic & Fusi (2024) defined abstract and disentangled representations for continuous latents, which does not necessarily apply directly to categorical variables. Nevertheless, we discuss these notions here in order to bring our notion of symbolic representations into context. According to Vafidis et al. (2025), an abstract representation of the latent variables $z_1, \ldots, z_m$ represents each z_i linearly and approximately mutually orthogonally. Disentangled representations encode each z_i orthogonally, without the necessity of linearity. Note that under this definition, axis-alignment is not required, i.e., it is not required that a given neuron represents a single latent factor. We demand axis-alignment for symbolic representations in the following sense. We say that a representation $(\hat{z}_1, \ldots, \hat{z}_{l'})^\top$ is a symbolic representation of the total latent variable vector $\mathbf{z} = (z_1, \ldots, z_l)^\top$ if for each i there exists a set of neurons such that $z_i = 1$ implies that at least one neuron in the set is active and furthermore for each neuron in the set, activity of the neuron implies $z_i = 1$. In other words, the presence of the latent category can easily be checked by checking whether at least one neuron in the corresponding set is active. This includes the special case when each category is represented by exactly one neuron. Note that a representation that is a structured permutation of the latents is by definition a symbolic representation. A formal definition of symbolic representations as well as a discussion of their relation to structured permutations is given in Appendix A.1.

We tested our model using latent factors with a WTA structure that did not match the factor structure. We used 10 output neurons in each of the 5 WTA heads for latent variables $Z_1, \ldots, Z_5$ with $l_1 = 5$, $l_2 = 8$, $l_3 = 5$, $l_4 = 3$, and $l_5 = 9$ categories respectively. In this setup we had more categories and therefore we used a larger entangled representational space $\mathbf{x} \in \mathbb{R}^{100}$. We trained the model on $n = 50$ tasks 20 times with different random seeds. 19 out of 20 models solved the tasks perfectly (see Appendix A.7). The resulting representation $\hat{\mathbf{z}}$ of one model, which solved the tasks perfectly, is shown in Fig. 2a. Again, each WTA head can be mapped to exactly one latent variable. For a given WTA head, several outputs are mapped to one category such that they encode a category in a shared manner. Each category can be decoded by a logical OR over these outputs, hence the network has developed a symbolic representation of the latent variables. We evaluated the representations of each model (see Appendix A.3) and found that 17 out of 19 models that solved the tasks perfectly evolved a fully symbolic representation of the latents. However, even when the latent variables could not fully be recovered, the number of symbolically represented categories was high (94.33% on average and 99.3% on average on the perfect runs, see Appendix A.7).

We next investigated whether symbolic representations also emerge when the number of WTA heads exceeds the number of latent factors in the input distribution. We used two additional WTA heads

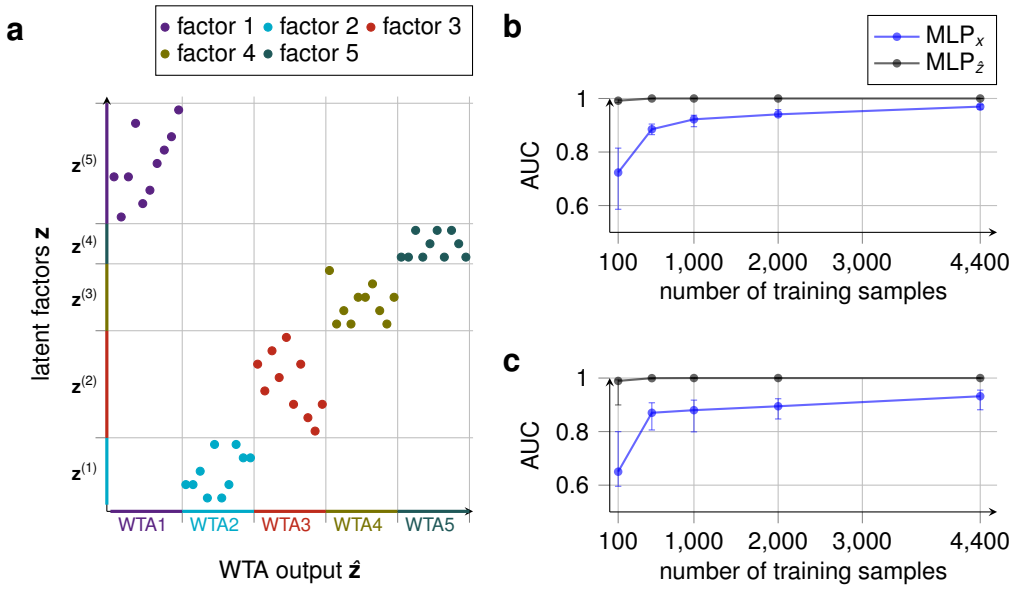

Figure 2: **Emergence of symbolic representations and generalization behavior. a)** Mapping between the WTA output neurons (x-axis, 10 per WTA head) of a trained multi-task model and the categories of the latent factors (y-axis, number of categories varies among the latents). A dot indicates that the corresponding WTA output was active at least once when the corresponding latent category was present. Hence, this WTA output neuron partly encodes the category or even encodes the category entirely if no other point appears in the same category. Note that each latent factor is represented by exactly one WTA head. **b)** Test performance (median AUC with first and third quartiles over 20 seeds) of an MLP trained on the symbolic representation $\hat{\mathbf{z}}$ (black) versus performance of an MLP trained directly on the entangled representation $\mathbf{x}$ (blue) for a new task as a function of the number of training examples (random train-test split). **c)** Same as panel b, but for generalization to an unseen latent value pair (pair-of-categories train-test split).

(overall 7 WTA heads) for the same latent structure as in the previous experiment. A symbolic representation emerged in 6 out of 20 runs. On average, 87.17% of all categories could be recovered. We wondered whether the percentage of symbolic representations could be improved in oversized multi-WTA bottlenecks by encouraging the readout layer to use as few WTA output neurons as possible, thus promoting sparsity of the representation (Lachapelle et al., 2023). To this end, we used ℓ_1 regularization on the readout layer weights to force the model to use as few WTA output neurons as possible (see Appendix A.3). Indeed, when we used 7 WTA heads in combination with ℓ_1 regularization, then a symbolic representation emerged in 14 out of 20 runs (compared to 6 out of 20 without regularization). On average, 87.83% of all categories could be recovered (see Appendix A.7).

We also trained models in the unmatched setup using data generated from latent factors with additional confounding factors. Despite the additional noise introduced by the confounding factors, the models achieved results comparable to those of the models without confounding factors (see Appendix A.11).

In summary, our results show that multi-task learning with WTA bottlenecks can extract symbolic representations also in the case when the multi-WTA bottleneck is clearly larger than the latent factor structure. However, as the bottleneck size increases, the network representation becomes less constrained and training can find loopholes to represent latents in a partly entangled manner.

5.3 Generalization behavior of symbolic representations

Johnston & Fusi (2023) showed that an abstract representation of continuous latent variables generalizes better to new linear classification tasks than an entangled representation. We first tested whether

this result also holds for categorical latent variables. To obtain an abstract representation, we used our usual multi-task learning setup (Fig. 1a) but without the WTA heads (see Appendix A.4). Hence, the representation layer was a linear layer, similar to Johnston & Fusi (2023). Using this setup, we found that the evolved representation indeed has better generalization properties than the entangled representation, see Appendix A.8.

In the following, we aim to show that this improved generalization also transfers to symbolic representations. Our experimental protocol is based on Johnston & Fusi (2023) and has been adapted to categorical latents. To this end, we compared the generalization capability of classifiers trained on the evolved symbolic representation $\hat{\mathbf{z}}$ to that of classifiers trained on the entangled representation $\mathbf{x}$ in the unmatched setup described above. We examined three different types of generalization.

First, we randomly split the set $\mathcal{Z}$ of all possible 5400 latent vectors into three disjoint sets, a training set $\mathcal{Z}_{\text{train}}$, a validation set $\mathcal{Z}_{\text{val}}$, and a test set $\mathcal{Z}_{\text{test}}$ (we refer to this split as the *random train-test split*). Consequently, the models were tested on category combinations that were not part of the training procedure. We defined a new linear classification task on the latent variables and trained a multi layer perceptron $\text{MLP}_{\mathbf{x}}$ on the entangled representation $\mathbf{x}$ and another multi layer perceptron $\text{MLP}_{\hat{\mathbf{z}}}$ on the multi-WTA representation $\hat{\mathbf{z}}$. To ensure a fair comparison, we used for both models the validation set for hyperparameter optimization (HPO) and tested them on the test set. We used the Area Under the Receiver Operating Characteristic curve (AUC)(Fawcett, 2006) to evaluate the generalization ability of both models in dependence of the training set size, see Fig. 2b. The AUC values are reported as median [Q1-Q3], where Q1 is the first and Q3 is the third quartile. Even for a training set size of 100 samples (less than 2% of all possible latent vectors), $\text{MLP}_{\hat{\mathbf{z}}}$ exhibited nearly perfect generalization with an AUC of 0.992 $[0.985 - 0.995]$ on the test set (train: AUC of 1 $[0.999 - 1]$). The test performance of $\text{MLP}_{\mathbf{x}}$ was clearly worse even for 4400 training samples (AUC of 0.970 $[0.960 - 0.977]$; train: AUC of 0.987 $[0.973 - 0.991]$).

We next tested generalization to samples that include a combination of feature-values of the first two latent variables that has never been observed during training (see Appendix A.4, we refer to this split as the *pair-of-categories train-test split*). The gap increased for this stronger type of generalization between both models, see Fig. 2c. Again, for a training set size of 100 samples, $\text{MLP}_{\hat{\mathbf{z}}}$ exhibited good generalization (test: AUC of 0.989 $[0.900 - 0.993]$; train: AUC of 1 $[0.995 - 1]$). The test performance of $\text{MLP}_{\mathbf{x}}$ at 4400 training samples dropped strongly (AUC of 0.932 $[0.882 - 0.955]$; train: AUC of 0.932 $[0.900 - 0.968]$).

It was argued that disentangled representations possess the ability to automatically generalize over irrelevant factors. For example, if a learner has learned to distinguish red and green berries for one berry shape, then it can generalize this distinction to other berry shapes, as color is disentangled from shape (Johnston & Fusi, 2023; Vafidis et al., 2025). In fact, this test has been proposed to identify entangled representations (Johnston & Fusi, 2023). To this end, we trained MLPs on a task which did not depend on the first latent variable. Here, the training set contained only samples where the first latent variable has the constant value 1. The test set consisted of all possible samples which were not used in the training procedure (see Appendix A.4, we refer to this split as the *constant-category train-test split*). $\text{MLP}_{\mathbf{x}}$ achieved an AUC of 0.904 $[0.832 - 0.934]$ on the training set and an AUC of 0.814 $[0.721 - 0.855]$ on the test set. Hence, it did not generalize well, indicating an entangled representation. In contrast, $\text{MLP}_{\hat{\mathbf{z}}}$ exhibited excellent generalization with an AUC of 0.999 $[0.998 - 1]$ on the training set and an AUC of 0.959 $[0.928 - 0.978]$ on the test set. All results are summarized in Appendix A.9.

5.4 Symbolic representations from visual input

So far, we defined the entanglement transformation Φ explicitly through a neural network. We next asked whether our results also apply to a categorical variant of the standard dsprites data set (Matthey et al., 2017) that has been used frequently to study disentanglement (Burgess et al., 2018). The dsprites data set consists of images of simple shapes (rectangle, circle, heart). We used non-rotated shapes at 64 different locations and colorized them with 10 different colors, resulting in 4 latent factors with $l_1 = 3$ (shape), $l_2 = 8$ (position x), $l_3 = 8$ (position y), $l_4 = 10$ (shape color). Hence, the transformation Φ is implicit in the generation of the input images.

We trained a model with 4 WTA heads, each comprising 10 output neurons, following the procedure described in Section 5.1. The model was trained on $n = 50$ tasks for $1{,}000$ epochs, repeated across

20 random seeds. In 7 out of 20 runs, the model solved the tasks perfectly and in 6 of these 7 runs a symbolic representation emerged. Over all 20 runs, a symbolic representation emerged in 9 cases, and 89.3% of all categories could be recovered. We also performed the experiment with 6 WTA heads to investigate if a symbolic representation emerges when the WTA structure is much larger than the latent factor structure. In 15 out of 20 runs a symbolic representation emerged and 98.4% of all categories could be recovered. When using ℓ_1 regularization, on the readout layer weights, a symbolic representation emerged in 11 out of 20 runs and 95.9% of the categories could be recovered. Detailed results can be found in Appendix A.10.

In summary, symbolic representations emerged for both, generic and visual entanglement setups also the WTA structure is larger than the latent factor structure. Our results indicate that the emergence of symbolic representation could benefit from a larger WTA structure, possibly combined with ℓ_1 regularization could be helpful, although our results are not fully conclusive in this respect.

We also repeated the generalization experiment described in Section 5.3 for the dsprites data set. The results were very similar to those in Section 5.3 (see Fig. A3b) and can be found in the Appendix A.10.

6 Discussion

We have proven that under certain conditions, neural networks with multi-WTA bottlenecks evolve representations that recover categorical latent factors of their inputs up to a structured permutation.

This result is surprising as it shows that categorical latent variables can be recovered exactly from arbitrarily entangled representations with multi-task learning. The disentanglement does however not come for free. It necessitates a set of sufficiently different tasks that indirectly provide information about the latents. The number of tasks needed is however moderate. According to our theory, the full rank of the task matrix W is sufficient, which implies that the number of tasks should be at least l, that is, the cumulative number of categories of the latent variables. Our simulation results indicate that structured permutations emerge even for smaller numbers of tasks (Section 5.1). Our theoretical result is based on the assumption that these tasks are predictions of posterior probabilities over classes (leading to a quasi-linear model). This raises the question whether other types of tasks that arise naturally in biological or practically relevant situations would induce the same or at least similar types of representations. Natural candidates are self-supervised tasks such as autoencoding (Bank et al., 2023; Kingma & Welling, 2013) or prediction (Nagai, 2019; Oord et al., 2018). Since our multi-WTA bottleneck naturally implements a sparse code, autoencoding relates our framework to sparse autoencoders (Makhzani & Frey, 2013; 2015).

To derive our theoretical result, we assumed that the multi-WTA bottleneck has the same structure as the latent variables (i.e., the number of outputs of the WTAs match the number of number of categories of the latents). Our simulation results revealed that symbolic representations (the weaker form of a structured permutation) also emerge with a reasonable probability if the number of WTA outputs exceed the category number and if multi-task performance is close to the global optimum. This however can only be true up to some extent, as for large WTAs, the network can find easy ways to encode the values of latents in an entangled manner. It would be interesting to investigate whether there are conditions for the emergence of symbolic representations in this more general case.

From the neuroscience perspective, our model fits well to the known importance of WTA motifs in cortical circuits (Douglas et al., 1989; Jonke et al., 2017) and the sparse coding theory (Olshausen & Field, 2004). It is also in line with symbol-like object representations found in higher cortical areas (Quiroga et al., 2005). Some authors have argued that linear mixtures of latent variables have advantages over (symbolic) axis-aligned representations in terms of a higher flexibility to learn new functions (Rigotti et al., 2013; Johnston & Fusi, 2023; Vafidis et al., 2025). We argue that both types of representations have advantages (see also below), which means that likely a larger family of representations is extracted and utilized in cortex.

Both, in terms of machine learning and neuroscience, symbolic representations have the advantage that they can function as an interface to symbolic reasoning modules, giving rise to high-level cognitive capabilities such as abstraction, variable binding (Feldman, 2013; Müller et al., 2020), and mathematical skills. Therefore, symbolic feature extractors could play a key role in the rapidly growing field of neurosymbolic artificial intelligence.

AI USE STATEMENT

In this work, we used generative AI tools to assist in writing proofs, formulating mathematical claims, and translating text. We also used generative AI tools to provide feedback on research methodology or experiments and for coding assistance, particularly for implementing methods, cleaning and reformatting data sets, and generating synthetic datasets. We have not used generative AI tools to provide critical ingredients for proving mathematical claims; propose or refine hypotheses; support qualitative and thematic data analysis; or interpret results. The item concerning help in developing theoretical models or conceptual frameworks is not applicable to this work. Additionally, we used generative AI tools to create or modify scientific figures or images; suggest experimental parameters; create or edit software code; and summarize or analyze existing literature. We have reviewed all AI-assisted work. AI-generated code was reviewed, and AI-generated feedback on the proof was assessed for validity. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To support reproducibility of our theoretical results, we provide the problem formulation in the main text (see Section 3), the necessary definitions in the Appendix A.1, and the proof of the theorem in Appendix A.2.

To support reproducibility of our experiments, we describe the architectures of the models used in our experiments in Section 3, Section 5, Appendix A.3, Appendix A.4 and Appendix A.5. The training and evaluation procedures are described in Section 5, Appendix A.3, Appendix A.4 and Appendix A.5. We describe the generation of the synthetic dataset in Section 5 and Appendix A.3. We also use the publicly available dsprites dataset and describe in Appendix A.5 how it is preprocessed for our experiments. Upon acceptance, we will make the code used to reproduce the experiments publicly available on GitHub.

ACKNOWLEDGMENTS

REFERENCES

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

Michael Avermann, Christian Tomm, Celine Mateo, Wulfram Gerstner, and Carl CH Petersen. Microcircuits of excitatory and inhibitory neurons in layer 2/3 of mouse barrel cortex. *Journal of neurophysiology*, 107(11):3116–3134, 2012.

Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pp. 353–374, 2023.

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

Johannes Bill, Lars Buesing, Stefan Habenschuss, Bernhard Nessler, Wolfgang Maass, and Robert Legenstein. Distributed bayesian computation and self-organized learning in sheets of spiking neurons with local lateral inhibition. *PloS one*, 10(8):e0134356, 2015.

Christopher P Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-VAE. *arXiv preprint arXiv:1804.03599*, 2018.

Pytorch Contributors. Pytorch documentation linear layer. https://docs.pytorch.org/docs/2.11/generated/torch.nn.Linear.html, 2026. Accessed: 2026-05-05.

Rodney J Douglas, Kevan AC Martin, and David Whitteridge. A canonical microcircuit for neocortex. *Neural computation*, 1(4):480–488, 1989.

William Falcon and The PyTorch Lightning team. Pytorch lightning, March 2025. URL https://doi.org/10.5281/zenodo.15053754.

Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2005.10.010. ROC Analysis in Pattern Recognition.

Jerome Feldman. The neural binding problem (s). *Cognitive neurodynamics*, 7(1):1–11, 2013.

Irina Higgins, David Amos, David Pfau, Sebastien Racaniere, Loic Matthey, Danilo Rezende, and Alexander Lerchner. Towards a definition of disentangled representations. *arXiv preprint arXiv:1812.02230*, 2018.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkE3y85ee.

W Jeffrey Johnston and Stefano Fusi. Abstract representations emerge naturally in neural networks trained to perform multiple tasks. *Nature Communications*, 14(1):1040, 2023.

Zeno Jonke, Robert Legenstein, Stefan Habenschuss, and Wolfgang Maass. Feedback inhibition shapes emergent computational properties of cortical microcircuit motifs. *Journal of Neuroscience*, 37(35):8511–8523, 2017.

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

Bohdan Kivva, Goutham Rajendran, Pradeep Ravikumar, and Bryon Aragam. Learning latent causal graphs via mixture oracles. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 18087–18101. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/966aad8981dcc75b5b8ab04427a833b2-Paper.pdf.

Bohdan Kivva, Goutham Rajendran, Pradeep Ravikumar, and Bryon Aragam. Identifiability of deep generative models without auxiliary information. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 15687–15701. Curran Associates, Inc., 2022. doi: 10.52202/068431-1141. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/649f080d8891ab4d4b262cb9cd52e69a-Paper-Conference.pdf.

Sebastien Lachapelle, Tristan Deleu, Divyat Mahajan, Ioannis Mitliagkas, Yoshua Bengio, Simon Lacoste-Julien, and Quentin Bertrand. Synergies between disentanglement and sparsity: Generalization and identifiability in multi-task learning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 18171–18206. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/lachapelle23a.html.

Seunghyun Lee and Yuqi Gu. Deep discrete encoders: Identifiable deep generative models for rich data with discrete latent layers. *Journal of the American Statistical Association*, 121(553):194–208, 2026. doi: 10.1080/01621459.2025.2587922.

Robert Legenstein, Zeno Jonke, Stefan Habenschuss, and Wolfgang Maass. A probabilistic model for learning in cortical microcircuit motifs with data-based divisive inhibition. *arXiv preprint arXiv:1707.05182*, 2017.

Robert Legenstein, Wolfgang Maass, Christos H Papadimitriou, and Santosh S Vempala. Long term memory and the densest k-subgraph problem. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, pp. 57–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018.

Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *international conference on machine learning*, pp. 4114–4124. PMLR, 2019.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Wolfgang Maass. On the computational power of winner-take-all. *Neural computation*, 12(11): 2519–2535, 2000.

Alireza Makhzani and Brendan Frey. K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013.

Alireza Makhzani and Brendan J Frey. Winner-take-all autoencoders. *Advances in neural information processing systems*, 28, 2015.

Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

Michael G Müller, Christos H Papadimitriou, Wolfgang Maass, and Robert Legenstein. A model for structured information representation in neural networks of the brain. *eneuro*, 7(3), 2020.

Yukie Nagai. Predictive learning: its key role in early cognitive development. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 374(1771), 2019.

Bernhard Nessler, Michael Pfeiffer, and Wolfgang Maass. Stdp enables spiking neurons to detect hidden causes of their inputs. *Advances in neural information processing systems*, 22, 2009.

Bruno A Olshausen and David J Field. Sparse coding of sensory inputs. *Current opinion in neurobiology*, 14(4):481–487, 2004.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Srdjan Ostojic and Stefano Fusi. Computational role of structure in neural activity and connectivity. *Trends in Cognitive Sciences*, 28(7):677–690, 2024.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: an imperative style, high-performance deep learning library*. Curran Associates Inc., Red Hook, NY, USA, 2019.

R Quian Quiroga, Leila Reddy, Gabriel Kreiman, Christof Koch, and Itzhak Fried. Invariant visual representation by single neurons in the human brain. *Nature*, 435(7045):1102–1107, 2005.

Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature neuroscience*, 2(11):1019–1025, 1999.

Mattia Rigotti, Omri Barak, Melissa R Warden, Xiao-Jing Wang, Nathaniel D Daw, Earl K Miller, and Stefano Fusi. The importance of mixed selectivity in complex cognitive tasks. *Nature*, 497 (7451):585–590, 2013.

David E Rumelhart and David Zipser. Feature discovery by competitive learning. *Cognitive science*, 9(1):75–112, 1985.

David E Rumelhart and David Zipser. Feature discovery by competitive learning. In *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*, pp. 151–193. 1986.

Pantelis Vafidis, Aman Bhargava, and Antonio Rangel. Disentangling representations through multitask learning. In *The Thirteenth International Conference on Learning Representations*, 2025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Xin Wang, Hong Chen, Si'ao Tang, Zihao Wu, and Wenwu Zhu. Disentangled representation learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):9677–9696, 2024.

Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL `https://github.com/facebookresearch/hydra`.

A Appendix

A.1 Definitions and derivations for Section 3

Definition of linear tasks: We define n linear binary classification tasks based on the latent categorical variable vector $\mathbf{z}$. The i-th task assumes that $\mathbf{z}$ is originating from one of two classes indicated by $A_i = 1$ (class 1) and $A_i = 0$ (class 0) with respective prior probabilities $P(A_i = 1)$ and $P(A_i = 0)$. If $\mathbf{z}$ originated from class 1 (resp. 0), it is assumed that each $\mathbf{z}^j$ was drawn from a categorical distribution with parameters $\mathbf{p}^{(i,j)}$ (resp. $\mathbf{q}^{(i,j)}$). Again, we write $\mathbf{p}^{(i)}$ (resp. $\mathbf{q}^i$) for the concatenation of these parameter vectors, i.e., $\mathbf{p}^{(i)} = (\mathbf{p}^{(i,1)}; ...; \mathbf{p}^{(i,m)})^\top$.

A standard and well-known derivation shows that the posterior $P(A_i = 1|\mathbf{z})$, that $\mathbf{z}$ belongs to class $A_i = 1$, can be written as

$$P(A_i = 1|\mathbf{z}) = \sigma\left(\ln \frac{P(A_i = 1, \mathbf{z})}{P(A_i = 0, \mathbf{z})}\right) \tag{13}$$

$$= \sigma\left(\ln \frac{P(\mathbf{z}|A_i = 1)}{P(\mathbf{z}|A_i = 0)} + \ln \frac{P(A_i = 1)}{P(A_i = 0)}\right). \tag{14}$$

We can write $P(\mathbf{z}^{(j)}|A_i = 1) = \prod_{k=1}^{l_j} (p_k^{(i,j)})^{z_k^{(j)}}$, with k denoting the category index of a latent categorical variable j. We assume that the latent categorical variables are independent. Therefore, we can multiply over all categoricals and write $P(\mathbf{z}|A_i = 1) = \prod_{j=1}^{m} \prod_{k=1}^{l_j} (p_k^{(i,j)})^{z_k^{(j)}}$. The same applies to $P(\mathbf{z}|A_i = 0)$. We use this in Equation 14 to obtain

$$P(A_i = 1|\mathbf{z}) = \sigma\left(\sum_{j=1}^{m}\sum_{k=1}^{l_j} z_k^{(j)} \ln \frac{p_k^{(i,j)}}{q_k^{(i,j)}} + \ln \frac{P(A_i = 1)}{P(A_i = 0)}\right). \tag{15}$$

We can merge the indices j and k into a single index j, which runs over all l categories

$$P(A_i = 1|\mathbf{z}) = \sigma\left(\sum_{j=1}^{l} z_j \ln \frac{p_j^{(i)}}{q_j^{(i)}} + \ln \frac{P(A_i = 1)}{P(A_i = 0)}\right). \tag{16}$$

We define $w_{ij} = \ln \frac{p_j^{(i)}}{q_j^{(i)}}$ and $b_i = \ln \frac{P(A_i=1)}{P(A_i=0)}$ and get

$$P(A_i = 1|\mathbf{z}) = \sigma\left(\sum_{j=1}^{l} w_{ij} z_j + b_i\right) = \sigma\left(\mathbf{w}_i^\top \mathbf{z} + b_i\right). \tag{17}$$

It is convenient to stack the posteriors of all tasks together. We thus define $P(A|\mathbf{z}) \in [0, 1]^n$ as a vector with the i-th component given by $P(A_i = 1|\mathbf{z})$. Therefore, we can write

$$P(A|\mathbf{z}) = \sigma\left(W\mathbf{z} + \mathbf{b}\right), \tag{18}$$

with $W = [w_{ij}]_{\substack{i=1,...,n \\ j=1,...,l}}$ and $\mathbf{b} = (b_1, \ldots, b_n)^\top$. Note however that, since each $\mathbf{z} \in \mathcal{Z}$ has exactly m 1-entries, $\mathbf{b}$ can be incorporated into W. To incorporate $\mathbf{b}$ into W, one can add $\frac{b_i}{m}$ to each each entry in the i-th row of W for each i. Hence, we can write

$$P(A|\mathbf{z}) = \sigma\left(W\mathbf{z}\right), \tag{19}$$

for this re-definition of W.

Definition of structured permutation: We define the latent structure with the help of the vector $\boldsymbol{\ell}$ of the number of latent categories for each latent factor

$$\boldsymbol{\ell} = (l_1, \ldots, l_m). \tag{20}$$

We furthermore define the function $t_{\boldsymbol{\ell}} : \{0, \ldots, m\} \to \mathbb{N}$ as

$$t_{\boldsymbol{\ell}}(k) = \sum_{i=1}^{k} l_i. \tag{21}$$

Note that $t_{\boldsymbol{\ell}}(k-1)+1, \ldots, t_{\boldsymbol{\ell}}(k)$ are the indices of latent variable Z_k in $\mathbf{z}$. We now formally define a structured permutation on the structure $\boldsymbol{\ell}$ in terms of valid permutation matrices R on that structure such that $\hat{\mathbf{z}}^\top = \mathbf{z}^\top R$ is a structured permutation of $\mathbf{z}$. Let π_d denote the set of permutations of the vector $(1, \ldots, d)^\top$. We define the set of valid permutations $\pi_{\boldsymbol{\ell}}$ of latents in structure $\boldsymbol{\ell}$ as

$$\pi_{\boldsymbol{\ell}} = \{\mathbf{s} | \mathbf{s} \in \pi_m \text{ and } l_i = l_{s_i} \forall i \in \{1, \ldots, m\}\}. \tag{22}$$

R is a structured permutation matrix for the structure $\boldsymbol{\ell}$ if

- R is a permutation matrix, and
- there exists an $\mathbf{s} \in \pi_{\boldsymbol{\ell}}$ such that the submatrix $[r_{ij}]_{\substack{i=t_{\boldsymbol{\ell}}(s_i-1)+1,\ldots,t_{\boldsymbol{\ell}}(s_i) \\ j=t_{\boldsymbol{\ell}}(i-1)+1,\ldots,t_{\boldsymbol{\ell}}(i)}}$ of R is a permutation matrix for all $i \in \{1, \ldots, m\}$.

Definition of symbolic representations: Consider a representation $\hat{\mathbf{z}} = (\hat{z}_1, \ldots, \hat{z}_{l'})^\top \in \{0,1\}^{l'}$ and a total latent vector $\mathbf{z} = (z_1, \ldots, z_l)^\top$.

We say that a category z_i is represented in a symbolic manner if there exists a subset $\mathcal{I}^{(i)} \subseteq \{1, \ldots, l'\}$ such that

1. $z_i = 0 \Rightarrow \hat{z}_j = 0 \quad \forall j \in \mathcal{I}^{(i)}$, and
2. $z_i = 1 \Rightarrow \exists j \in \mathcal{I}^{(i)}$ such that $\hat{z}_j = 1$.

We refer to the number of categories for which these conditions hold as *number of symbolic-encoded categories*. We say that a representation $\hat{\mathbf{z}}$ is a symbolic representation of $\mathbf{z}$, if all categories z_i of $\mathbf{z}$ are represented in a symbolic manner in $\hat{\mathbf{z}}$.

To obtain a structured permutation from a symbolic representation, all WTA output neurons that jointly encode a category must be merged into a single WTA output neuron. In addition, redundant and irrelevant WTA heads and WTA output neurons must be removed. If this is done perfectly, the WTA structure is identical to the structure of the latent factors. One practical way to extract a structured permutation from a symbolic representation could be the analysis of the readout layer. The weights from the WTA output neurons to the readout, which encode the same category, should be highly correlated across tasks.

A.2 Proof of Theorem 1

In this section, we prove Theorem 1 from Section 4. In the main text, we wrote $\hat{C}$ for the code matrix derived from the multi-WTA bottleneck. To avoid cluttered notation, we write D for this matrix here. Recall from the main text: We say that a matrix C has an m-fold one-hot column-block structure if every row of C consists of m consecutive blocks where each block has exactly one 1 entry and the other entries in the block are 0.

We first prove the following intermediate lemma, where $\mathbf{1}_p$ denotes the all-ones column vector of size p.

Lemma 1. *Let m and $l_1, ..., l_m$ be natural numbers with $l_i \geq 2$. Let $l = \sum_{i=1}^{m} l_i$ and $p = \prod_{i=1}^{m} l_i$. Let $C, D \in \{0,1\}^{p \times l}$ be two matrices with an m-fold one-hot column-block structure. Both C and D consist of all possible rows of that type. Assume that $C = DG$ for some matrix $G \in \mathbb{R}^{l \times l}$. Then, C can be expressed as $C = DQ + \mathbf{1}_p \mathbf{b}^\top$ for a matrix $Q \in \{0,1\}^{l \times l}$ and a vector $\mathbf{b} \in \mathbb{Z}^l$. Furthermore, tile each column of Q into m blocks of sizes $l_1, ..., l_m$. Then, in every column of Q, no block consists entirely of 1's.*

Proof. Denote by $\mathbf{d}^{\text{row},k}$ ($\mathbf{d}^{\text{col},k}$) the k-th row (column) of a matrix D. We have

$$c_i^{\text{row},k} = \mathbf{d}^{\text{row},k}\mathbf{g}^{\text{col},i}. \tag{23}$$

Consistent with the m-fold one-hot column block structure of D, we can divide D into m column blocks of sizes $l_1, ..., l_m$ and index these blocks by $1, \ldots, m$. We see from Equation 23 that each column block corresponds to a block of rows in G in the sense that the vector of the β-th block in a row of D is multiplied with the vector of the β-th block in a column of G. We thus divide the rows of G in such blocks and index them the same way by $1, \ldots, m$.

Consider an arbitrary column block β of D and an arbitrary column i of G. Since, by definition, D contains all one-hot-block combinations, there exist l_β rows in D which are identical outside the column block β and necessarily different within the column block. In the following, we consider only these l_β rows of D and call the set of these row indices K. Denote by $\alpha_k^\beta \in \{1, ..., l_\beta\}$ the index within column block β such that $d^{\text{row},k,\beta}_{\alpha_k^\beta} = 1$, where $\mathbf{d}^{\text{row},k,\beta}$ is a vector containing block β from row k of D. For all $\alpha_k^\beta \in \{1, ..., l_\beta\}$ it holds that

$$c_i^{\text{row},k} = const^{i,\beta} + g^{\text{col},i,\beta}_{\alpha_k^\beta} \in \{0, 1\}, \tag{24}$$

where $k \in K$ and $const^{i,\beta}$ is a constant term for a specific i and β. $\mathbf{g}^{\text{col},i,\beta}$ is a vector containing block β from column i of G. Equation 24 implies that $g^{\text{col},i,\beta}_{\alpha_k^\beta} \in \{-const^{i,\beta}, 1 - const^{i,\beta}\}$. Therefore, $g^{\text{col},i,\beta}_{\alpha_k^\beta}$ can take only two different values, a base value $g^{\text{col},i,\beta}_{\text{base}} = -const^{i,\beta}$ or the value $g^{\text{col},i,\beta}_{\text{base}} + 1$.

Hence, we have

$$c_i^{\text{row},k} = \mathbf{d}^{\text{row},k}\mathbf{g}^{\text{col},i} = \sum_{\beta=1}^{m} g^{\text{col},i,\beta}_{\text{base}} + \mathbf{d}^{\text{row},k}\mathbf{q}^{\text{col},i} = g^{\text{col},i}_{\text{base}} + \mathbf{d}^{\text{row},k}\mathbf{q}^{\text{col},i}, \tag{25}$$

with $\mathbf{q}^{\text{col},i} \in \{0, 1\}^l$ and $g^{\text{col},i}_{\text{base}} = \sum_{\beta=1}^m g^{\text{col},i,\beta}_{\text{base}}$. We compile $g^{\text{col},i}_{\text{base}}$ in a vector $\mathbf{b}$ such that $b_i = \sum_{\beta=1}^m g^{\text{col},i,\beta}_{\text{base}}$ and write this in matrix form:

$$C = DQ + \mathbf{1}_p\mathbf{b}^\top. \tag{26}$$

Now assume that $\mathbf{q}^{\text{col},i}$ in Equation 25 has a block that consists of only 1's. This adds a contribution of 1 independent of $\mathbf{d}$. Therefore, we replace all blocks of column i consisting only of 1's of $\mathbf{q}^{\text{col},i}$ with blocks consisting only of 0's and add for each of these blocks a 1 to b_i. We do this for all columns i.

Since C, D, Q are integer matrices, it follows that $\mathbf{b} \in \mathbb{Z}^l$.

□

Using this result, we prove now Theorem 1. For clarity, we restate the Theorem 1 below, using D in place of $\hat{C}$ (see Theorem 2).

Theorem 2. *Let m and $l_1, ..., l_m$ be natural numbers with $l_i \geq 2$. Let $l = \sum_{i=1}^m l_i$ and $p = \prod_{i=1}^m l_i$. Let $C, D \in \{0, 1\}^{p\times l}$ be two matrices with an m-fold one-hot column-block structure with arbitrary column-block sizes of $l_1, ..., l_m$. Both C and D consist of all possible rows of that type. Assume that $C = DG$ for some matrix $G \in \mathbb{R}^{l\times l}$. Then, the columns of D are a structured permutation of the columns of C with the restriction that only column blocks with the same sizes can be permuted.*

Proof. We know from Lemma 1 that we can write C as $C = DQ+\mathbf{1}_p\mathbf{b}^\top$ for a matrix $Q \in \{0, 1\}^{l\times l}$ and a vector $\mathbf{b} \in \mathbb{Z}^l$. Furthermore, we know that there is no block in the column vectors of Q that consists of only 1's. We first show that we can write $C = DQ$, i.e., we show that $\mathbf{b} = 0$.

We determine the set of values that $(D\mathbf{q}^{\text{col},j})_i$ can take. Fix a column j of Q. Partition $\mathbf{q}^{\text{col},j}$ into m blocks of sizes $l_1, ..., l_m$. For block β, let a_β be the number of ones of $\mathbf{q}^{\text{col},j}$ in that block ($0 \leq a_\beta \leq l_\beta$).

For any row $\mathbf{d}$ of D, the scalar $(\mathbf{dq}^{\text{col},j})$ equals the number of blocks for which $\mathbf{d}$ picks a position where $\mathbf{q}^{\text{col},j}$ has a 1. As $\mathbf{d}$ ranges over all rows of D (all choices of one position per block), the attainable values of $\mathbf{dq}^{\text{col},j}$ form exactly the full integer interval

$$\{\mathbf{dq}^{\text{col},j} : \mathbf{d} \text{ is a row of } D\} = \{s_{\min}, s_{\min}+1, \ldots, s_{\max}\}, \tag{27}$$

where

- $s_{\min} = |\{\beta : a_\beta = l_\beta\}|$ (number of blocks that are all-ones),
- $s_{\max} = |\{\beta : a_\beta \geq 1\}|$ (number of blocks that have at least one 1).

In particular, we have $s_{\min} = 0$ since by construction $\mathbf{q}^{\text{col},j}$ has no block with all-ones (see Lemma 1). Furthermore, 1 is attainable iff there is at least one block with $a_\beta \geq 1$.

Because C has all possible rows of the one-hot-per-block type and $l_i \geq 2$, each column of C contains both 0's and 1's across the p rows (its column sum is $s_\beta = p/l_\beta$).

Therefore, for each j, $\{c_{ij} | i = 1, \ldots, p\} = \{0, 1\}$. But $C_{ij} = \mathbf{d}^{\text{row},i}\mathbf{q}^{\text{col},j} + b_j$. Hence $\{\mathbf{d}^{\text{row},i}\mathbf{q}^{\text{col},j} + b_j | i = 1, \ldots, p\} = \{0, 1\}$ for a constant b_j. From above, we know that the attainable set of $\mathbf{d}^{\text{row},i}\mathbf{q}^{\text{col},j}$ is an interval of consecutive integers $\{s_{\min}, \ldots, s_{\max}\}$ with $s_{\min} = 0$. Therefore, we have to find a $s_{\max} \in \mathbb{N}$ and a $b_j \in \mathbb{Z}$ such that $\{0, \ldots, s_{\max}\} + b_j = \{0, 1\}$ holds. The only way it can be exactly two points that become $\{0, 1\}$ after adding b_j is hence $s_{\max} = 1$, $b_j = 0$.

Since this holds for every column j, we conclude that $\mathbf{b} = \mathbf{0}$. Therefore, we can write $C = DQ$ and the following must hold:

$$\{\mathbf{d}^{\text{row},i}\mathbf{q}^{\text{col},j} | i = 1, \ldots, p\} = \{0, 1\}. \tag{28}$$

This equation implies that there can be only 1's in one block of each $\mathbf{q}^{\text{col},j}$. Consider an arbitrary column j of Q and assume there is a 1 in two different blocks of $\mathbf{q}^{\text{col},j}$. We know that D contains all one-hot block combinations, therefore there exists a $\mathbf{d}$ such that $\mathbf{dq}^{\text{col},j} = 2$. This leads to a contradiction with Equation 28. Therefore, each column of C depends only on one column block of D. Therefore, the nonzero entries of $\mathbf{q}^{\text{col},j}$ are contained in exactly one block. Since a block consisting entirely of 1's cannot exist, the other blocks must consist only of 0's, and the block with 1's cannot consist entirely of 1's. If a column in C depends on multiple columns in D, then the column in C contains a 1 as long as any of the columns in D contains a 1 in the same row.

We show that all columns of one arbitrary column block β of C depend on the same column block of D. Consider one arbitrary column block β of C. Assume we have two columns j_1 and j_2 in the column block β and these columns depend on different column blocks of D. We know that D contains all one-hot block combinations, therefore there exists a $\mathbf{d}$ such that $\mathbf{dq}^{\text{col},j_1} = \mathbf{dq}^{\text{col},j_2} = 1$. This contradicts the one-hot column block structure of C. Therefore, all columns of one column block of C depends on only one column block of D. Consider one arbitrary column block β of C, we denote the column block of D, which β depends on, by $\pi(\beta)$.

We have m column blocks in C and m column blocks in D and every column block of C depends on exactly one column block of D. We show that there is a bijective mapping between the column blocks of C and the column blocks of D. Assume we have two column blocks β_1 and β_2 with $\beta_1 \neq \beta_2$ in C such that $\pi(\beta_1) = \pi(\beta_2)$. Therefore, there exists a column block β_3 in D on which no column block in C depends. By definition, we know that $l_{\beta_3} \geq 2$ and D consist of one-hot block combinations, therefore there exist two rows i_1 and i_2 in D where only the column block β_3 changes. It follows that $\mathbf{d}^{\text{row},i_1}Q = \mathbf{d}^{\text{row},i_2}Q$. This contradicts the definition of C that C consist of all p one-hot block combinations and C consists of p rows. It follows that each row must be unique. Therefore, it is not possible that two different column blocks in C depend on the same column block in D. It is also not possible that a column block of C doesn't depend on any column block of D. Assume there exist a column block β of C which depend on no column block of D. For an arbitrary column $\mathbf{d}$ it follows that $\mathbf{dq}^{\text{col},j,\beta} = 0 \quad \forall j \in \{1, ..., l_\beta\}$. This contradicts the one-hot column block structure of C. Therefore, each column block of C depends on exactly one column block of D and two different column blocks of C cannot depend on the same column block of D. Since C and D have the same number of column blocks, there must be a bijective mapping between the column blocks of C and the column blocks of D.

We show that there is a bijective mapping between the columns of a column block β of C and the columns of $\pi(\beta)$ if $l_\beta = l_{\pi(\beta)}$. Consider an arbitrary column j_1 from an arbitrary column block β_1 and $\mathbf{c}^{\text{col},j_1}$ depends on multiple columns of one column block $\pi(\beta_1)$. We know that every column in C consists of both 0's and 1's. Therefore, every column in C must depend at least on one column in D. It follows that there must exist two columns j_2 and j_3 with $j_2 \neq j_3$ in β_1 which depend on the same column in $\pi(\beta_1)$. We know that D contains all one-hot block combinations, therefore there exists a $\mathbf{d}$ such that $\mathbf{dq}^{\text{col},j_2} = \mathbf{dq}^{\text{col},j_3} = 1$. This contradicts the one-hot column block structure of C. Therefore, each column of column block β_1 depend on exactly on column of $\pi(\beta_1)$ and there exists no two columns in β_1 which depend on the same column of $\pi(\beta_1)$. It follows that there is a bijective mapping between the columns of a column block β_1 and the columns of $\pi(\beta_1)$.

We show now that column blocks of D are mapped only to such column blocks of C which have the same number of columns. Consider an arbitrary column block β_1. We know that each column of C consists of both 0's and 1's. Therefore, each column of C depends on at least one column in D. Assume we have $l_{\beta_1} > l_{\pi(\beta_1)}$, then there exist two columns j_1 and j_2 with $j_1 \neq j_2$ in β_1 which depend on the same column in $\pi(\beta_1)$. This leads again to a contradiction. Assume we have $l_{\beta_1} < l_{\pi(\beta_1)}$. We know that we have a bijective mapping between the column blocks of C and the column blocks of D and therefore there exists an block β_2 in C which depends on a column block $\pi(\beta_2)$ such that $l_{\beta_2} > l_{\pi(\beta_2)}$. This leads again to a contradiction. Therefore, a column block in C depends on exactly one column block in D with the same number of columns.

Hence, the columns of C are a structured permutation of the columns of D (permutation of blocks and within-block permutations) with the restriction that only column blocks with the same sizes can be permuted, which further implies that also the columns of D are a structured permutation of the columns of C with the restriction that only column blocks with the same sizes can be permuted.

□

A.3 Details to Section 5.1 and Section 5.2

Architecture of Φ We used an MLP with 4 layers and a Leaky ReLU as activation function for Φ. The dimensions of the layers were $(25 \to 70 \to 70 \to 70)$ for the matched setup and $(30 \to 100 \to 100 \to 100)$ for the unmatched setup. The linear layers in the MLPs were randomly initialized from a uniform distribution, which is implemented by default in PyTorch linear layers (Contributors, 2026), and the weights were frozen afterwards.

Architecture of multi-WTA encoder We used an MLP with 4 layers and Leaky ReLU as activation function except for the last layer. Directly before the last layer, we used layer normalization. We used 5 WTA heads on top of the last layer as described in Section 3. Each WTA head consisted of a Gumbel-softmax function with discretized one-hot vectors as output. The one-hot vectors of all WTA heads were stacked together to obtain $\hat{\mathbf{z}}$. The layer dimensions of the MLP were $(70 \to 70 \to 70 \to 25)$ for the matched setup and $(100 \to 100 \to 100 \to 50)$ for the unmatched setup.

For our experiments, we used the following implementation of the function f_{WTA}, which is based on Jang et al. (Jang et al., 2017). We used the Gumbel softmax

$$f_{\text{gs}}(\mathbf{a}^{(i)})_k = \frac{\exp(\frac{a_k^{(i)} + g_k}{\tau})}{\sum_{j=1}^{l_i} \exp(\frac{a_j^{(i)} + g_j}{\tau})} \tag{29}$$

where $g_1, ..., g_{l_i}$ is sampled from a Gumbel(0,1) distribution (see (Jang et al., 2017)) and τ is the temperature. In our expeiments f_{WTA} was then computed as

$$f_{\text{WTA}}(\mathbf{a}^{(i)}) = \text{one_hot}(\underset{k \in \{1,...,l_i\}}{\arg\max} [f_{\text{gs}}(\mathbf{a}^{(i)})_k]). \tag{30}$$

The Straight-Through Gumbel-Softmax Estimator (Jang et al., 2017) used during training in the forward pass $f_{\text{WTA}}(\mathbf{a}^{(i)})$ and in the backward pass only $f_{\text{gs}}(\mathbf{a}^{(i)})$ instead of $f_{\text{WTA}}(\mathbf{a}^{(i)})$ such that $\frac{\partial f_{\text{WTA}}(\mathbf{a}^{(i)})}{\partial \mathbf{a}^{(i)}} = \frac{\partial f_{\text{gs}}(\mathbf{a}^{(i)})}{\partial \mathbf{a}^{(i)}}$. During evaluation f_{WTA} was computed without Gumbel noise.

Table A1: Hyperparameter used for multi-task training. We used AdamW as optimizer (Loshchilov & Hutter, 2019).

	Setup		
Hyperparameter	unmatched	unmatched confounding	matched
optimizer	AdamW	AdamW	AdamW
optimizer β_1, β_2	0.9, 0.999	0.9, 0.999	0.9, 0.999
learning rate	0.020	0.005	0.008
learning rate scheduler	cos. annealing	cos. annealing	cos. annealing
learning rate scheduler $\eta_{\min}$	10^{-6}	10^{-6}	10^{-6}
learning rate scheduler $T_{\max}$	1000	2000	1000
weight decay	0.0	0.0	0.0
number of epochs	1000	2000	1000
Gumbel softmax τ	20	20	6
Gumbel softmax τ exp. decay	0.999	0.999	0.999
number of tasks	50	50	25

Architecture of readout layer We used a linear layer without a bias to project $\hat{\mathbf{z}}$ to the n tasks. Finally, a sigmoid activation function is applied on top to predict the n tasks.

Task generation We generated a task according to the following procedure. We sampled for the i-th task a categorical distribution parametrization $\mathbf{p}^{(i,j)}$ for each latent variable j using a Dirichlet distributions with $\alpha = 1$ for all categories. We stacked all $\mathbf{p}^{(i,j)}$ together to obtain $\mathbf{p}^{(i)}$. We repeated this process a second time to obtain $\mathbf{q}^{(i)}$. We sampled the prior probability $P(A_i = 1)$ from a uniform distribution on the interval $(0, 1)$. Subsequently we computed $w_{ij} = \ln \frac{p_j^{(i)}}{q_j^{(i)}}$ and $b_i = \ln \frac{P(A_i=1)}{P(A_i=0)}$ and calculated the posterior for each sample according to Equation 17 to get the target value for each sample.

Input vectors were produced based on vectors $\mathbf{z}$ that were sampled from categorical distributions with uniform probabilities over categories.

Hyperparameters for multi-task learning We used the hyperparameters shown in Table A1 to train the models in the multi-task learning setup. To enable the models to find the optimal solution, we used high learning rates and a large temperature τ within the Gumbel-softmax. The high learning rate and the large temperature enable the model to explore different mappings between $\mathbf{z}$ and $\hat{\mathbf{z}}$. We also used an exponential decay for τ and cosine annealing for the learning rate to reduce exploration towards the end of training and converge to a final model. If we used ℓ_1 regularization, we used the mean value of all absolute values of the readout layer's weights, multiplied it by 0.003, and added it to our loss.

For multi-task learning, networks were trained on $100,000$ input samples for $1,000$ epochs and tested on $10,000$ samples. We performed this training procedure 20 times with different random seeds which lead to different network initializations and different shuffles of the training samples.

Evaluation of symbolic representation We evaluated whether a representation is symbolic by checking the conditions given in *Definition of symbolic representations* in Appendix A.1.

Task performance metrics Task performance was evaluated using the AUC. The target values for a task are probability values ranging from 0 to 1. For the AUC, we dichotomized the probability value. Values greater than or equal to 0.5 were set to 1, and values less than 0.5 were set to 0. The dichotomized probability values and the model output were used to calculate the AUC.

To evaluate how closely the target probabilities align with the model outputs during multi-task training, we used the mean absolute error (MAE). The tasks were considered perfectly solved if the MAE on the test set was less than 10^{-6}.

A.4 Details to Section 5.3.

Readout MLPs The following multilayer perceptrons were trained on different representations to solve a new linear classification task. These multilayer perceptrons were used to evaluate which representations promote generalization to new linear classification tasks. $\mathrm{MLP}_{\mathbf{x}}$ was trained on the entangled representation $\boldsymbol{x}$. To reproduce the generalization results of Johnston et al. (Johnston & Fusi, 2023) on categorical latent variables, we used $\mathrm{MLP}_{\mathbf{a}}$. $\mathrm{MLP}_{\mathbf{a}}$ was trained on the f_{MLP} output $\boldsymbol{a}$ of a trained multi-task model without multi-WTA bottleneck f_{WTA}. Another readout $\mathrm{MLP}_{\hat{\mathbf{z}}}$ was trained on the multi-WTA output $\hat{\mathbf{z}}$ of a trained multi-task model.

Architecture of readout MLP We used for $\mathrm{MLP}_{\mathbf{x}}$, $\mathrm{MLP}_{\hat{\mathbf{z}}}$ and $\mathrm{MLP}_{\mathbf{a}}$ an MLP with Leaky ReLU activation function after each layer except for the last layer which consisted of a single neuron with a logistic sigmoid nonlinearity. The number and dimensions of hidden layers were not fixed but determined by hyperparameter optimization, see paragraph *Training procedure* below. This ensured a fair comparison as the hyperparameter optimization chooses the hyperparameter with best validation binary cross entropy.

Train-test splits for generalization evaluations We split the input samples $\mathcal{Z}$ into training set $\mathcal{Z}_{\text{train}}$, validation set $\mathcal{Z}_{\text{val}}$, and test set $\mathcal{Z}_{\text{test}}$ in three different ways, in order to test different types of generalization in our models. For each split type, we had $\mathcal{Z}_{\text{train}} \cup \mathcal{Z}_{\text{val}} \cup \mathcal{Z}_{\text{test}} = \mathcal{Z}$ and $\mathcal{Z}_{\text{train}} \cap \mathcal{Z}_{\text{val}} \cap \mathcal{Z}_{\text{test}} = \{\}$.

For the first split type (random train-test split), we randomly chose 500 samples for the validation and 500 for the test set. The other 4400 samples were used for the training set. Hence, the test set consisted of category combinations of the latent variables which were not part of the training process.

In the second split type (pair-of-categories train-test split), we made sure that the models were tested on a test set which consisted of samples with a particular category combination of the first two latents which has never been seen during training. To this end, we selected the first category of the first latent variable and the first category of the second latent variable and used all 135 samples, which contain this combination, for the test set. We used the second category of the first latent variable and the second category of the second latent variable and used all 135 samples which contain this combination for the validation set. The other 5130 samples which were not part of test and validation set were used for the training set.

For the third split type (constant-category train-test split), we selected the first category of the first latent variable and used all 1080 samples which contain this category for the training set. We selected the second category of the first latent variable and used all 1080 samples which contain this category for the validation set. The other 3240 samples which were not part of training and validation set were used for the test set.

Training procedure We used a Bayesian hyperparameter optimization approach (Tree-structured Parzen Estimator) (Akiba et al., 2019) to find appropriate hyperparameter for $\mathrm{MLP}_{\mathbf{x}}$, $\mathrm{MLP}_{\mathbf{a}}$ and $\mathrm{MLP}_{\hat{\mathbf{z}}}$. For each experiment we used 20 trials to find an appropriate number of layers, dimension size of hidden layers, learning rate, and so on. The optimized hyperparameters and the search space is depicted in Table A2. If the hyperparameter dropout value was greater than 0, we used a dropout layer after the Leaky ReLU activation in each hidden layer. If the hyperparameter layernorm was True, we used a layer normalization on the input. In each trial, an MLP was trained for a maximum of 500 epochs, where training terminated early if the validation loss did not improve within 10 epochs. The hyperparameter from the trial that performed best on the validation set were used for the final run. The resulting MLP was then evaluated on the test set.

We performed this training procedure 20 times with different random seeds which lead to different network initializations and differently shuffles of the training samples. In addition, for each seed, an independently sampled task was generated on which the generalization was evaluated.

Table A2: Hyperparameters and the corresponding search space used for hyperparameter optimization.

Hyperparameter	Range
hidden dim.	$\{16, 32, 64, 128, 256\}$
number of hidden layers	$\{1, 2, 3, 4, 5\}$
dropout	$\{0, 0.1, 0.2\}$
layernorm	$\{\text{True}, \text{False}\}$
learning rate	$[10^{-4}, 10^{-1}]$
weight decay	$[10^{-6}, 10^{-2}]$
batch size	$\{16, 32, 64, 128, 256\}$

A.5 Details to Section 5.4

Dsprites data set The dsprites data set introduced in Matthey et al. (2017) is built on 5 latent factors: shape, scale, orientation, x-, and y-position. Scale and orientation are not suitable as categorical features, so they were set to a fixed value (largest scale and no rotation). The position factors, with 32 values in each direction, were subsampled evenly to 8 possible position in each dimension.To increase the variety of possible feature combinations, we introduced color to the dataset, increasing the dimensions from 64×64 to $64 \times 64 \times 3$ and assigning a set of visually distinct colors, generated by heuristically selecting points from a coarse RGB lattice, emphasizing variation across color channels. This results in a total of 1920 samples.

To test generalization, the pair-of-categories train-test split described above in Section A.4 was used as follows:

- All samples with the first shape and the first 3 colors were used as test set.
- All samples with the second shape and the subsequent 3 colors were used as validation set.
- All remaining samples were used as training set.

Hence, the validation and test set each consisted of $3 \cdot 8 \cdot 8 = 192$ samples, 10% of the total data set.

The constant-category train-test split was the same as above in Section A.4. The training, validation and test set consisted of $8 \cdot 8 \cdot 10 = 640$ samples each.

Architecture The network architecture was similar to Section 5.2 and 5.3, described in A.3 and A.4. It only differed slightly in the encoder architecture, which used 4 layers $(256 \to 256 \to 256 \to 40)$, followed by 4 WTA heads of size 10 each.

Training procedure We used the same training procedure as in Section 5.2 and 5.3, described in A.3 and A.4.

Hyperparameters for multi-task learning We used the same hyperparameters as for the unmatched setup (see Table A1), with the exception that we used a learning rate of 0.005.

A.6 Supplementary results for Section 5.1

Fig. A1 illustrates the structured permutation learned by one of the models in the matched setup (30 tasks).

Fig. A2 illustrates the structured permutation learned by one of the models in the matched setup (15 tasks).

Table A3 shows the mean MAE on the test set after multi-task learning. It also shows how many categories were successfully recovered.

Table A4 shows mean, minimum and standard deviation of the MAE on the test set after multi-task learning for different numbers of tasks. Additionally it shows how many runs solved the tasks perfectly and the percentage of these runs where a structured permutation emerged.

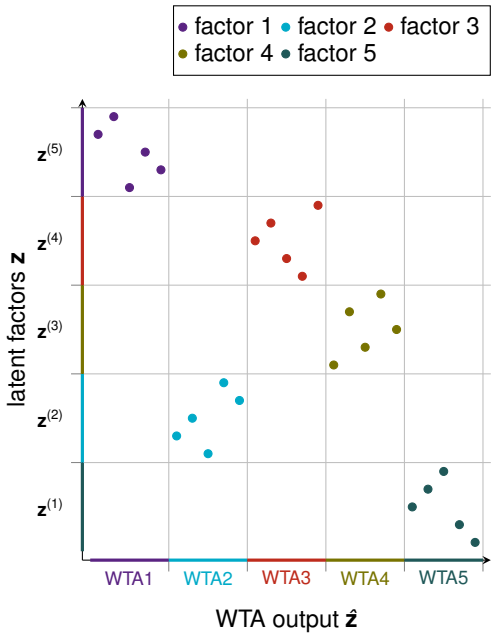

Figure A1: **Emergence of symbolic representations.** Mapping between the WTA output neurons (x-axis, 5 per WTA head), of a trained multi-task model in the matched setup with 30 tasks, and categories of the latent factors (y-axis). A dot indicates that the corresponding WTA output was active at least once when the corresponding latent category was present. Hence, this WTA output neuron encodes exactly one category, because a structured permutation emerged at the WTA output.

Table A3: Matched Setup on 30 tasks and the MAE on the test set. A symbolic representation is present when all 25 categories are encoded.

Run	MAE	Symbolic-encoded categories (max. 25)
1	2.62×10^{-8}	25
2	3.23×10^{-7}	25
3	9.59×10^{-8}	25
4	8.82×10^{-7}	25
5	1.20×10^{-7}	25
6	5.90×10^{-8}	25
7	4.69×10^{-7}	25
8	8.56×10^{-9}	25
9	3.58×10^{-8}	25
10	2.89×10^{-7}	25
11	2.45×10^{-8}	25
12	1.03×10^{-6}	25
13	7.91×10^{-8}	25
14	0.0854	1
15	1.26×10^{-8}	25
16	1.54×10^{-7}	25
17	0.0002	23
18	0.0712	6
19	2.44×10^{-7}	25
20	0.0587	11

Table A4: The mean, standard deviation (SD), and minimum of the Mean Absolute Error (MAE) after multi-task training on test dataset across 20 runs with different seeds in the matched setup. The column "# solved perfectly" shows the number of models that solved the n tasks perfectly after training. The last column depicts the percentage of models which solved the n tasks perfectly also have developed a valid structured permutation. NA stands for not available, because no model solved the n tasks perfectly.

n tasks	Mean MAE	SD MAE	Min MAE	# solved perfectly (max. 20)	Valid symbolic rep. in %
1	0.0118	0.0004	0.0108	0	NA
5	0.0633	0.0008	0.0618	0	NA
10	0.0916	0.0030	0.0836	0	NA
15	0.0320	0.0377	1.33×10^{-7}	2	100
20	0.0157	0.0327	8.59×10^{-9}	14	100
25	0.0032	0.0144	3.85×10^{-8}	18	100
30	0.0108	0.0267	8.56×10^{-9}	15	100

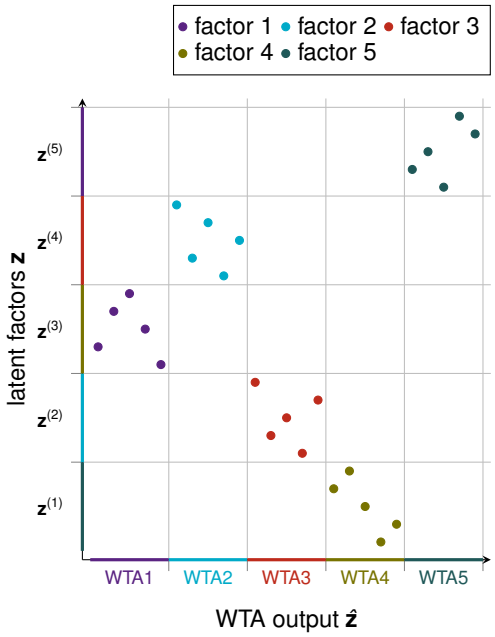

Figure A2: **Emergence of symbolic representations.** Mapping between the WTA output neurons (x-axis, 5 per WTA head), of a trained multi-task model in the matched setup with 15 tasks, and categories of the latent factors (y-axis). A dot indicates that the corresponding WTA output was active at least once when the corresponding latent category was present. Hence, this WTA output neuron encodes exactly one category, because a structured permutation emerged at the WTA output.

A.7 Supplementary results for Section 5.2

Table A5 shows the mean MAE on the test set after multi-task learning of a model (unmatched setup) with 5 WTA heads, each with 10 neurons. It also shows how many categories were successfully recovered. Table A6 shows the same metrics, with the exception that 7 WTA heads were used. The same applies to Table A7 with the additional exception that a ℓ_1 regularization on the weights of the readout layer were used.

Table A5: Unmatched Setup with 5 WTA heads on 50 tasks and the MAE on the test set is reported. A symbolic representation is present when 30 categories are encoded.

Run	MAE	Symbolic-encoded categories (max. 30)
1	7.99×10^{-9}	30
2	7.96×10^{-9}	30
3	8.04×10^{-9}	30
4	8.00×10^{-9}	30
5	8.60×10^{-9}	30
6	8.75×10^{-9}	30
7	0.3110	0
8	8.86×10^{-9}	30
9	2.04×10^{-8}	28
10	8.69×10^{-9}	30
11	8.25×10^{-9}	30
12	7.95×10^{-9}	30
13	1.86×10^{-8}	30
14	8.68×10^{-9}	30
15	8.80×10^{-9}	30
16	8.66×10^{-9}	30
17	8.68×10^{-9}	30
18	8.66×10^{-9}	30
19	8.54×10^{-9}	28
20	8.82×10^{-9}	30

Table A6: Unmatched Setup with 7 WTA heads on 50 tasks and the MAE on the test set is reported. A symbolic representation is present when 30 categories are encoded.

Run	MAE	Symbolic-encoded categories (max. 30)
1	0.3110	0
2	9.56×10^{-9}	28
3	9.20×10^{-9}	30
4	9.15×10^{-9}	30
5	0.3110	0
6	9.25×10^{-9}	29
7	5.21×10^{-8}	27
8	8.95×10^{-9}	29
9	1.36×10^{-8}	29
10	9.54×10^{-9}	29
11	8.80×10^{-9}	30
12	9.02×10^{-9}	29
13	9.03×10^{-9}	29
14	8.84×10^{-9}	30
15	9.32×10^{-9}	29
16	9.07×10^{-9}	29
17	9.20×10^{-9}	28
18	1.84×10^{-8}	28
19	9.24×10^{-9}	30
20	9.55×10^{-9}	30

Table A7: Unmatched Setup with 7 WTA heads and ℓ_1 regularization on 50 tasks and the MAE on the test set is reported. A symbolic representation is present when 30 categories are encoded.

Run	MAE	Symbolic-encoded categories (max. 30)
1	0.0014	28
2	0.0010	30
3	0.3110	0
4	0.0009	30
5	0.0010	30
6	0.0009	30
7	0.0008	30
8	0.0010	30
9	0.0009	30
10	0.0008	30
11	0.0009	29
12	0.0009	30
13	0.0010	22
14	0.0011	30
15	0.0009	28
16	0.0009	30
17	0.3110	0
18	0.0009	30
19	0.0009	30
20	0.0009	30

A.8 Supplementary results for Section 5.3: Generalization of abstract representations

Table A8 shows the detailed AUC metrics for training $\mathrm{MLP}_{\mathbf{x}}$ and $\mathrm{MLP}_{\mathbf{a}}$ on random train-test splits. The same metrics are shown in Table A9 for the pair-of-categories train-test split. The detailed results for training on the constant-category train-test split are shown in Table A10.

Table A8: Train and test performance (AUC) of $\mathrm{MLP}_{\mathbf{x}}$ and $\mathrm{MLP}_{\mathbf{a}}$ for the random train-test split with varying number of training samples. The median and the first and third quartile of the AUC across 20 runs with different seeds are reported.

	Train AUC					
# train samples	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{a}}$	Q1 $\hat{\mathbf{a}}$	Q3 $\hat{\mathbf{a}}$
100	0.875	0.768	0.906	1.000	1.000	1.000
500	0.940	0.908	0.951	1.000	1.000	1.000
1000	0.961	0.944	0.971	1.000	1.000	1.000
2000	0.962	0.949	0.977	1.000	1.000	1.000
4400	0.977	0.970	0.985	1.000	1.000	1.000
	Test AUC					
# train samples	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{a}}$	Q1 $\hat{\mathbf{a}}$	Q3 $\hat{\mathbf{a}}$
100	0.743	0.625	0.805	0.992	0.988	0.994
500	0.896	0.860	0.910	1.000	0.999	1.000
1000	0.917	0.900	0.935	1.000	1.000	1.000
2000	0.940	0.921	0.955	1.000	1.000	1.000
4400	0.962	0.949	0.972	1.000	1.000	1.000

Table A9: Train and test performance (AUC) of $\mathrm{MLP}_{\mathbf{x}}$ and $\mathrm{MLP}_{\mathbf{a}}$ for the pair-of-categories train-test split with varying number of training samples. The median and the first and third quartile of the AUC across 20 runs with different seeds are reported.

	Train AUC					
# train samples	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{a}}$	Q1 $\hat{\mathbf{a}}$	Q3 $\hat{\mathbf{a}}$
100	0.731	0.643	0.823	1.000	0.999	1.000
500	0.834	0.643	0.879	1.000	1.000	1.000
1000	0.889	0.806	0.913	1.000	1.000	1.000
2000	0.906	0.852	0.934	1.000	1.000	1.000
4400	0.938	0.916	0.967	1.000	1.000	1.000
	Test AUC					
# train samples	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
100	0.697	0.644	0.799	0.990	0.974	0.997
500	0.832	0.651	0.894	1.000	0.999	1.000
1000	0.875	0.834	0.907	1.000	0.999	1.000
2000	0.910	0.880	0.948	1.000	1.000	1.000
4400	0.927	0.890	0.963	1.000	1.000	1.000

Table A10: Train and test performance (AUC) of $\mathrm{MLP}_{\mathbf{x}}$ and $\mathrm{MLP}_{\mathbf{a}}$ for the constant-category train-test split. The median and the first and third quartile of the AUC across 20 runs with different seeds are reported.

	AUC					
Set	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
Train	0.881	0.824	0.920	1.000	0.999	1.000
Test	0.784	0.740	0.848	0.965	0.896	0.985

A.9 Supplementary results for Section 5.3: Generalization of symbolic representations

Table A11 shows the detailed AUC metrics for training $\mathrm{MLP}_{\mathbf{x}}$ and $\mathrm{MLP}_{\hat{\mathbf{z}}}$ on random train-test splits. The same metrics are shown in Table A12 for the pair-of-categories train-test split. The detailed results for training on the constant-category train-test split are shown in Table A13.

Table A11: Train and test performance (AUC) of $\mathrm{MLP}_{\mathbf{x}}$ and $\mathrm{MLP}_{\hat{\mathbf{z}}}$ for the random train-test split with varying number of training samples. The median and the first and third quartile of the AUC across 20 runs with different seeds are reported.

	Train AUC					
# train samples	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
100	0.824	0.656	0.906	1.000	0.999	1.000
500	0.935	0.923	0.961	1.000	0.999	1.000
1000	0.953	0.938	0.974	1.000	1.000	1.000
2000	0.974	0.948	0.980	1.000	1.000	1.000
4400	0.987	0.973	0.991	1.000	1.000	1.000
	Test AUC					
# train samples	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
100	0.724	0.586	0.815	0.992	0.985	0.995
500	0.885	0.865	0.904	1.000	1.000	1.000
1000	0.922	0.895	0.937	1.000	1.000	1.000
2000	0.941	0.934	0.958	1.000	1.000	1.000
4400	0.970	0.960	0.977	1.000	1.000	1.000

Table A12: Train and test performance (AUC) of $\mathrm{MLP}_{\mathbf{x}}$ and $\mathrm{MLP}_{\hat{\mathbf{z}}}$ for the pair-of-categories train-test split with varying number of training samples. The median and the first and third quartile of the AUC across 20 runs with different seeds are reported.

	Train AUC					
# train samples	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
100	0.694	0.543	0.800	1.000	0.995	1.000
500	0.855	0.765	0.874	1.000	0.999	1.000
1000	0.883	0.811	0.908	1.000	1.000	1.000
2000	0.900	0.872	0.942	1.000	1.000	1.000
4400	0.932	0.900	0.968	1.000	1.000	1.000
	Test AUC					
# train samples	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
100	0.650	0.596	0.800	0.989	0.900	0.993
500	0.870	0.806	0.908	1.000	0.997	1.000
1000	0.880	0.799	0.918	1.000	1.000	1.000
2000	0.895	0.847	0.923	1.000	1.000	1.000
4400	0.932	0.882	0.955	1.000	1.000	1.000

Table A13: Train and test performance (AUC) of $\mathrm{MLP}_{\mathbf{x}}$ and $\mathrm{MLP}_{\hat{\mathbf{z}}}$ for the constant-category train-test split. The median and the first and third quartile of the AUC across 20 runs with different seeds are reported.

	AUC					
Set	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
Train	0.904	0.832	0.934	0.999	0.998	1.000
Test	0.814	0.721	0.855	0.959	0.928	0.978

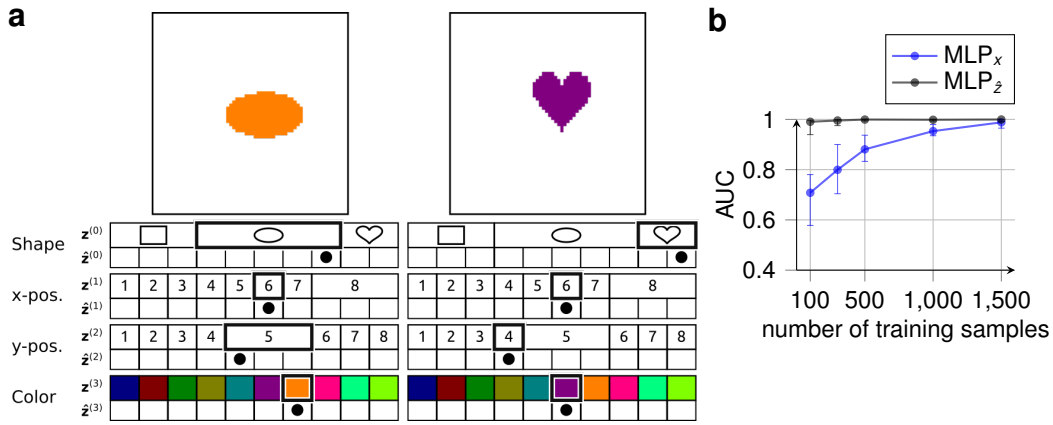

Figure A3: **Symbolic representation and generalization behavior on visual input. a)** Symbolic representation for two example inputs. Top: Input image $\mathbf{x}$. Bottom: For each latent factor $\mathbf{z}$, the input category that generated the sample and the corresponding WTA output $\hat{\mathbf{z}}$ after applying the mapping from WTA outputs to latent categories described in Appendix A.3. In each row, the active input category is highlighted, and the active output neuron is marked by a black dot. **b)** Generalization to samples with an unseen latent value pair. Test performance of an MLP trained on the symbolic representation $\hat{\mathbf{z}}$ (black) versus performance of an MLP trained directly on the entangled representation $\mathbf{x}$ (blue) for a new task as a function of the number of training examples.

A.10 Supplementary results for Section 5.4

In this section we show the detailed results for experiments with the dsprites data set. Table A14 shows the detailed metrics after training the encoder 20 times with 4 WTA heads on 50 tasks. Table A15 shows the detailed metrics with 6 WTA heads and Table A16 with 6 WTA heads and ℓ_1 regularization. Table A17 shows the generalization performance of $\text{MLP}_{\mathbf{x}}$ and $\text{MLP}_{\hat{\mathbf{z}}}$ for the random train-test split, Table A18 for the pair-of-categories train-test split and Table A19 for the constant-category train-test split.

In Fig. A3a two example inputs from the dsprites dataset are shown. In addition, a symbolic representation is shown. The first three neurons of the first WTA head jointly encode the rectangle shape, the next five neurons encode the oval shape, and the remaining two neurons encode the heart. Fig. A3b shows the generalization performance of the multi-task models with 4 WTA heads on the pair-of-categories train-test split as a function of the training set size.

Table A14: Results of the dsprites unmatched setup with 4 WTA heads on 50 tasks with 20 different seeds and the MAE on the test set is reported. A symbolic representation is present when 29 categories are encoded.

Run	MAE	Symbolic-encoded categories (max. 29)
1	9.06×10^{-7}	29
2	0.0027	27
3	0.0176	25
4	4.86×10^{-8}	29
5	1.65×10^{-6}	29
6	1.17×10^{-7}	29
7	2.04×10^{-6}	29
8	0.0533	14
9	2.96×10^{-6}	22
10	0.0183	25
11	8.63×10^{-9}	26
12	0.0165	25
13	6.49×10^{-6}	29
14	0.0290	14
15	0.0099	26
16	0.0021	26
17	6.75×10^{-8}	29
18	0.0098	27
19	3.07×10^{-7}	29
20	3.71×10^{-7}	29

Table A15: Results of the dsprites unmatched setup with 6 WTA heads on 50 tasks with 20 different seeds and the MAE on the test set is reported. A symbolic representation is present when 29 categories are encoded.

Run	MAE	Symbolic-encoded categories (max. 29)
1	1.91×10^{-6}	29
2	8.43×10^{-9}	29
3	8.64×10^{-8}	29
4	2.21×10^{-6}	27
5	4.96×10^{-8}	29
6	3.43×10^{-8}	29
7	1.70×10^{-7}	29
8	1.11×10^{-7}	28
9	1.74×10^{-6}	29
10	3.05×10^{-6}	27
11	7.05×10^{-8}	29
12	1.84×10^{-6}	29
13	3.50×10^{-6}	27
14	1.54×10^{-6}	27
15	2.48×10^{-8}	29
16	5.23×10^{-7}	29
17	1.35×10^{-7}	29
18	1.25×10^{-8}	29
19	5.05×10^{-8}	29
20	2.12×10^{-7}	29

Table A16: Results of the dsprites unmatched setup with 6 WTA heads and ℓ_1 regularization on 50 tasks with 20 different seeds and the MAE on the test set is reported. A symbolic representation is present when 29 categories are encoded.

Run	MAE	Symbolic-encoded categories (max. 29)
1	0.0012	25
2	0.0010	29
3	0.0010	29
4	0.0010	29
5	0.0012	27
6	0.0010	29
7	0.0010	29
8	0.0010	28
9	0.0017	25
10	0.0011	26
11	0.0010	29
12	0.0010	29
13	0.0011	29
14	0.0011	29
15	0.0011	29
16	0.0013	25
17	0.0013	28
18	0.0010	28
19	0.0016	25
20	0.0010	29

Table A17: Train and test performance (AUC) of $\mathrm{MLP}_{\mathbf{x}}$ and $\mathrm{MLP}_{\hat{\mathbf{z}}}$ on the random train-test split dsprites dataset with varying number of training samples. The median and the first and third quartile of the AUC across 20 runs with different seeds are reported.

	Train AUC					
# train samples	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
100	0.947	0.890	0.986	1.000	0.999	1.000
300	0.990	0.979	0.997	1.000	1.000	1.000
500	0.997	0.996	0.999	1.000	1.000	1.000
1000	0.999	0.998	0.999	1.000	1.000	1.000
1500	0.999	0.997	0.999	1.000	1.000	1.000
	Test AUC					
# train samples	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
100	0.648	0.571	0.708	0.995	0.983	0.998
300	0.818	0.753	0.868	1.000	0.999	1.000
500	0.908	0.856	0.926	1.000	0.999	1.000
1000	0.969	0.942	0.979	1.000	1.000	1.000
1500	0.985	0.976	0.992	1.000	1.000	1.000

Table A18: Train and test performance (AUC) of $\mathrm{MLP}_{\mathbf{x}}$ and $\mathrm{MLP}_{\hat{\mathbf{z}}}$ on the pair-of-categories dsprites dataset with varying number of training samples. The median and the first and third quartile of the AUC across 20 runs with different seeds are reported.

	Train AUC					
# train samples	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
100	0.954	0.855	0.989	1.000	0.982	1.000
300	0.995	0.964	0.999	0.999	0.999	1.000
500	0.998	0.988	0.999	1.000	1.000	1.000
1000	0.998	0.996	1.000	1.000	0.999	1.000
1500	0.999	0.997	0.999	1.000	0.999	1.000
	Test AUC					
# train samples	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
100	0.708	0.578	0.780	0.991	0.939	0.995
300	0.799	0.704	0.900	0.995	0.975	0.999
500	0.881	0.833	0.937	0.999	0.994	1.000
1000	0.953	0.935	0.980	0.998	0.993	1.000
1500	0.988	0.965	0.992	0.999	0.992	1.000

Table A19: Train and test performance (AUC) of $\mathrm{MLP}_{\mathbf{x}}$ and $\mathrm{MLP}_{\hat{\mathbf{z}}}$ for the constant-category train-test split on the dsprites data set. The median and the first and third quartile of the AUC across 20 runs with different seeds are reported.

	AUC					
Set	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
Train	0.990	0.966	0.996	1.000	0.997	1.000
Test	0.899	0.850	0.941	0.989	0.972	0.997

A.11 Experiments with confounding factors

In the previous setups, all latent factors were relevant and we assumed an injective map Φ from the latents to entangled representation $\mathbf{x}$. In reality, this mapping may be confounded by other factors that are neither observable nor relevant for the learner (and therefore considered noise). We emulated this situation by introducing additional confounding latent variables in $\mathbf{z}$ that did not influence the posterior of the tasks used to extract the WTA representation. From the viewpoint of the network, these confounding variables thus introduced noise in $\mathbf{x}$. To test the resilience of our model to such noise, we considered the unmatched setup as described above, with the addition of 5 confounding latent variables with same structure as the original latent variables. We used a larger entangled representational space with $\mathbf{x} \in \mathbb{R}^{500}$, to avoid that the additional noise reduces the information of $\mathbf{z}$ in $\mathbf{x}$. We used for the encoder the following structure: $500 \to 100 \to 100 \to 50$. We also used a lower learning rate with 0.005 and trained the models in the multi-task training for 2000 epochs. We evaluated the models for symbolic representation using the training dataset. We trained again 20 models with different random seeds, and all models developed a symbolic representation of the non-confounding latents (see Table A20).

The generalization performance of networks trained on these representations was comparable to the setup without confounding factors. See Table A21 for generalization results on the random train-test split, Table A22 for generalization results on the pair-of-category train-test split and Table A23 for generalization results on the constant-category train-test split.

Table A20: Unmatched Setup with confounding factors on 50 tasks and the MAE on the test set is reported. A symbolic representation is present when 30 categories are encoded.

Run	MAE	Symbolic-encoded categories (max. 30)
1	0.0003	30
2	0.0003	30
3	0.0003	30
4	0.0002	30
5	0.0003	30
6	0.0004	30
7	0.0003	30
8	0.0003	30
9	0.0004	30
10	0.0003	30
11	0.0004	30
12	0.0003	30
13	0.0004	30
14	0.0003	30
15	0.0004	30
16	0.0003	30
17	0.0004	30
18	0.0004	30
19	0.0005	30
20	0.0004	30

Table A21: Train and test performance (AUC) of $\mathrm{MLP}_{\mathbf{x}}$ and $\mathrm{MLP}_{\hat{\mathbf{z}}}$ on the random train-test split with varying number of training samples and confounding factors. The median and the first and third quartile of the AUC across 20 runs with different seeds are reported.

	Train AUC					
# train samples	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
100	0.847	0.549	0.989	1.000	1.000	1.000
500	0.971	0.965	0.988	1.000	1.000	1.000
1000	0.988	0.971	0.996	1.000	1.000	1.000
2000	0.992	0.982	0.996	1.000	1.000	1.000
4400	0.997	0.994	0.998	1.000	1.000	1.000
	Test AUC					
# train samples	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
100	0.760	0.589	0.822	0.991	0.984	0.995
500	0.910	0.900	0.922	1.000	1.000	1.000
1000	0.936	0.922	0.940	1.000	1.000	1.000
2000	0.945	0.941	0.956	1.000	1.000	1.000
4400	0.967	0.961	0.977	1.000	1.000	1.000

Table A22: Train and test performance (AUC) of $\mathrm{MLP}_{\mathbf{x}}$ and $\mathrm{MLP}_{\hat{\mathbf{z}}}$ for the pair-of-categories train-test split with varying number of training samples and confounding factors. The median and the first and third quartile of the AUC across 20 runs with different seeds are reported.

	Train AUC					
# train samples	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
100	0.914	0.740	0.984	1.000	0.997	1.000
500	0.938	0.769	0.966	1.000	0.999	1.000
1000	0.965	0.939	0.977	1.000	0.999	1.000
2000	0.967	0.953	0.987	1.000	0.999	1.000
4400	0.974	0.955	0.987	1.000	0.999	1.000
	Test AUC					
# train samples	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
100	0.745	0.614	0.794	0.983	0.937	0.989
500	0.868	0.564	0.917	1.000	0.997	1.000
1000	0.915	0.885	0.949	1.000	0.999	1.000
2000	0.929	0.889	0.958	1.000	0.999	1.000
4400	0.934	0.908	0.960	1.000	1.000	1.000

Table A23: Train and test performance (AUC) of $\mathrm{MLP}_{\mathbf{x}}$ and $\mathrm{MLP}_{\hat{\mathbf{z}}}$ on the constant-category train-test split with confounding factors. The median and the first and third quartile of the AUC across 20 runs with different seeds are reported.

	AUC					
Set	Median $\mathbf{x}$	Q1 $\mathbf{x}$	Q3 $\mathbf{x}$	Median $\hat{\mathbf{z}}$	Q1 $\hat{\mathbf{z}}$	Q3 $\hat{\mathbf{z}}$
Train	0.958	0.825	0.980	1.000	0.998	1.000
Test	0.861	0.758	0.879	0.966	0.920	0.985

A.12 Assets and code availability

The main Python packages used in our experiments were PyTorch (Paszke et al., 2019), PyTorch Lightning (Falcon & team, 2025), Optuna (Akiba et al., 2019), and Hydra (Yadan, 2019).

The code for reproduction of our experimental results is available at [GitHub link will be inserted in case of acceptance].