Abstractors: Transformer Modules for Symbolic Message Passing and Relational Reasoning

Awni Altabaa[1] **Taylor Webb**[2] **Jonathan Cohen**[3] **John Lafferty**[4]

May 29, 2023

Abstract: Reasoning in terms of relations, analogies, and abstraction is a hallmark of human intelligence. An active debate is whether this relies on the use of symbolic processing or can be achieved using the same forms of function approximation that have been used for tasks such as image, audio, and, most recently, language processing. We propose an intermediate approach, motivated by principles of cognitive neuroscience, in which abstract symbols can emerge from distributed, neural representations under the influence of an inductive bias for learning that we refer to as a "relational bottleneck." We present a framework that casts this inductive bias in terms of an extension of Transformers, in which specific types of attention mechanisms enforce the relational bottleneck and transform distributed symbols to implement a form of relational reasoning and abstraction. We theoretically analyze the class of relation functions the models can compute and empirically demonstrate superior sample-efficiency on relational tasks compared to standard Transformer architectures.

1. Introduction

Relational learning refers to the inference of rules that operate in terms of relationships between objects, independent of how the objects may be represented. Examples of common relations are "less than" applied to natural numbers, "same color as" applied to visual objects, and "friend of" applied to social relationships. Such relations are important for basic computational functions. For example, consider the task of sorting objects. A standard 52-card deck of playing cards has 13 ranks in each of four suits: clubs, diamonds, hearts, and spades. The cards might be sorted using the relation that takes the suit as the primary attribute, and the rank as the secondary attribute; this would be the typical way of ordering cards in many games. If a sorting algorithm is learned that depends only on the relations between objects, it can in principle be applied in a new domain with little or no modification.

Reasoning in terms of relations and analogies is a hallmark of human intelligence (Holyoak, 2012; Snow et al., 1984). Indeed, the Wisconsin card sorting task (Berg, 1948) has been used for decades as an indicator of decision making function in prefrontal cortex (Monchi et al., 2001). Recognizing the importance of this capability, which is largely separate from function approximation for sensory tasks such as image and audio processing, machine learning research has explored several novel frameworks for relational learning (Barrett et al., 2018; Battaglia et al., 2018; Graves et al., 2014; Mondal et al., 2023; Pritzel et al., 2017; Santoro et al., 2017; Webb et al., 2021; Whittington et al., 2020).

[1]Department of Statistics and Data Science, Yale University; awni.altabaa@yale.edu. [2]Department of Psychology, UCLA; taylor.w.webb@gmail.com. [3]Department of Psychology and Princeton Neuroscience Institute, Princeton University; jdc@princeton.edu. [4]Department of Statistics and Data Science, Wu Tsai Institute, Institute for Foundations of Data Science, Yale University; john.lafferty@yale.edu.

In this paper we propose a framework that casts relational learning in terms of Transformers. The success of Transformers lies in combining the function approximation capabilities of deep learning with the use of attentional mechanisms to support richly context-sensitive processing (Kerg et al., 2020; Vaswani et al., 2017; Wolf et al., 2020). However, it is clear that Transformers are missing core capabilities required for modeling human thought (Mahowald et al., 2023). In particular, they lack mechanisms required to emulate forms of flexibility and efficiency exhibited by the human brain, including an ability to support analogy and abstraction. While large language models show a surprising ability to complete some analogies (Webb et al., 2022), this ability emerges implicitly after processing vast amounts of data. The algorithmic challenge is to provide ways of binding domain-specific information to low dimensional, abstract representations that can be used to compute a given function in any setting for which it is relevant, based on limited data.

Our approach is motivated by a type of inductive bias for relational learning architectures we call the "relational bottleneck," which is motivated by principles of cognitive neuroscience that shed light on the brain subsystems involved when natural intelligence shows an ability to flexibly generalize abstract structure across domains of processing. In particular, the framework of complementary learning systems (Kumaran et al., 2016; McClelland et al., 1995) describes two distinct types of neural mechanisms for learning and memory around which the brain is organized, implementing a tradeoff between slow, incremental forms of learning required to encode stable statistical structure present in the environment (semantic memory), and the ability to rapidly encode and remember novel associations (episodic memory).

Recently, it has been proposed that episodic memory may also serve to implement a relational bottleneck, by allowing components of semantic memory that represent abstract functions to be separated from those responsible for processing domain-specific information, while allowing the two types of components to be coupled through rapid binding and similarity-based retrieval of representations (Webb et al., 2021). This "relational bottleneck" imposes an inductive bias that constrains the flow of information from sensory or motor subsystems to reasoning and decision making subsystems, by restricting this information to relations, as computed through inner products between distributed representations. In this paper we present a framework that recasts this inductive bias in terms of an extension of Transformers, in which specific types of attention mechanisms enforce the relational bottleneck. This creates a potentially powerful combination of deep learning and relational learning that implements a form of symbolic processing, enabling abstraction and generalization from limited data.

2. The Abstractor Framework

At a high level, the primary function of an Abstractor is to compute abstract relational features of its inputs.[1] That is, given a set or sequence of input objects $o_1, \ldots, o_m$, the relational Abstractor learns to model a relation $r(\cdot, \cdot)$ and computes a function on the set of pairwise relations between objects $\{r(o_i, o_j)\}_{ij}$. The relations and the computations on them are learned to carry out a specific prediction task. This learning is often end-to-end, but, crucially, the Abstractor framework naturally supports modular learning.

[1] In this paper, we will tend to use the name 'Abstractor' to refer to both the module, the framework, and models which contain the Abstractor module as a main component.

2.1. Relational symbolic message-passing

At the core of Abstractors is an operation we refer to as *relational symbolic message-passing*. The input to this operation is a relation tensor $R = [r(o_i, o_j)]_{i,j=1}^{m}$, where $r(o_i, o_j) \in \mathbb{R}^{d_r}$ is a vector describing the relation between object o_i and object o_j. We will come back to how an Abstractor models pairwise relations and computes the relation tensor in the next subsection.

The starting point of symbolic message-passing is a set of learnable symbols $s_1, \dots, s_m \in \mathbb{R}^{d_s}$, where the hyperparameter d_s is the dimension of the symbolic vectors. We call these parameters *symbols* because each of them references (or "is bound to") a particular object, but they are independent of the values of these objects. That is, the ith symbol references the ith object, but the value of s_i is independent of the value of o_i. The use of these learned, input-independent symbols is how symbolic message-passing achieves its abstraction.

In relational symbolic message-passing, we perform message-passing on these learned symbolic parameters according to the relation tensor R. In general, this message-passing operation can be described as a set-valued function of the form

$$s_i \leftarrow \text{Update}\Big(s_i,\ \{(R[i,j], s_j)\}_{j \in [m]}\Big), \quad i = 1, \dots, m. \tag{2.1}$$

That is, the value of the ith symbol is updated as a function of the set of tuples $(R[i,j], s_j)$ of the relations with all other objects and the symbols of these objects. The symbols s_j are naturally viewed as values on the nodes of a graph, and the relations $R[i,j]$ are naturally viewed as weights on the edges. A simple but important special case of this is linear message-passing

$$s_i \leftarrow \sum_{j=1}^{m} R[i,j] s_j, \quad i = 1, \dots, m \tag{2.2}$$

In the above, if $d_r > 1$, the operation should be read as

$$R[i,j] s_j = (R[i,j,1] s_j, \dots, R[i,j,d_r] s_j) \in \mathbb{R}^{d_s \times d_r},$$

where d_r is the dimension of the relation. That is, the result is concatenated.

Following message-passing, each updated symbol s_i can be passed through a feedforward neural network $\phi : \mathbb{R}^{d_s \times d_r} \to \mathbb{R}^{d_a}$ to compute a non-linear function of the output. This also controls the dimension of the symbols so that it doesn't grow by a factor of H with each layer (e.g., take $d_a = d_s$). Empirically, a residual connection and layer normalization may be useful.

This message-passing operation can be repeated multiple times to iteratively update the symbolic vectors. The output of relational symbolic message-passing is the set of symbols A at the end of this sequence of message-passing operations. This is summarized in Algorithm 1.

2.2. Multi-head relations and relational cross-attention

Next, we turn our attention to how the Abstractor models pairwise relations and computes the relation tensor $R \in \mathbb{R}^{m \times m \times d_r}$.

Algorithm 1: Symbolic Message-Passing

```
Input                  : Relation tensor: $R \in \mathbb{R}^{m \times m \times d_r}$
Hyperparameters        : $L,\ d_s, d_a$, hyperparameters of feedforward networks
Learnable parameters   : symbols $S = (s_1, \dots, s_m) \in \mathbb{R}^{d_s \times m}$, feedforward networks $\phi^{(1)}, \dots, \phi^{(L)}$
Output                 : Abstracted sequence: $A = (a_1, \dots, a_m) \in \mathbb{R}^{d_a \times m}$

$(a_1, \dots, a_m) \leftarrow (s_1, \dots, s_m)$
for $l \leftarrow 1$ to $L$ do
    $a_i \leftarrow \sum_{j=1}^{n} R[i,j] a_j, \quad i = 1, \dots, m$
    $a_i \leftarrow \phi^{(l)}(a_i), \quad i = 1, \dots, m$
end
```

The inner product operation is a natural way to capture notions of relations and similarity. In Euclidean space, inner products capture the geometric alignment between vectors. Similarly, for arbitrary objects with vector representations, inner products between these vector representations can capture relations between these objects.

We model pairwise relations as inner products between appropriately encoded (or 'filtered') object representations. In particular, we model the pairwise relation function $r(\cdot, \cdot) \in \mathbb{R}^{d_r}$ in terms d_r learnable 'left encoders' $\phi_1, \dots, \phi_{d_r}$, and d_r 'right encoders' $\psi_1, \dots, \psi_{d_r}$,

$$r(x, y) = \begin{pmatrix} \langle \phi_1(x), \psi_1(y) \rangle \\ \vdots \\ \langle \phi_{d_r}(x), \psi_{d_r}(y) \rangle \end{pmatrix} \in \mathbb{R}^{d_r}. \tag{2.3}$$

In general, ϕ_i, ψ_j can be any learnable maps. These transformations can be thought of as *relational filters*. They extract a particular attribute of the objects such that an inner product of the transformed objects indicates the alignment or similarity along this attribute. Having several different filters allows for modeling rich multi-dimensional relations. This is one notable advantage of this formulation over the CoRelNet model (Kerg et al., 2022), which processes a *1-dimensional* similarity matrix as input to a multi-layer perceptron. In the next section, we analyze the class of functions that the multi-head relation module can model.

In order to promote weight sharing, we focus our attention to inner product relations of the form

$$r(x, y) = \begin{pmatrix} \left\langle W_1^{(1)} \phi(x), W_2^{(1)} \phi(y) \right\rangle \\ \vdots \\ \left\langle W_1^{(d_r)} \phi(x), W_2^{(d_r)} \phi(y) \right\rangle \end{pmatrix} \in \mathbb{R}^{d_r}, \tag{2.4}$$

where ϕ is a common non-linear map, and $W_1^{(i)}, W_2^{(i)}$ are projection matrices for each dimension of the relation. For general functions ϕ, this class of functions is no smaller than the one above (e.g., take ϕ to be the concatenation of $\phi_1, \dots, \phi_{d_r}, \psi_1, \dots, \psi_{d_r}$ and $W_1^{(i)}, W_2^{(i)}$ to be the projection matrices which extract the appropriate components), but does enable greater weight sharing.

We refer to this operation as *Multi-Head Relation* (Algorithm 2). In our implementation, computation of the inner product is done efficiently with Einstein summation. Also, we add a hyperparameter to control whether the relations are modeled as symmetric or asymmetric (as in the description above). If the relations are to be modeled as symmetric, we set $W_1^{(i)} = W_2^{(i)}$. For certain tasks where relations may be naturally symmetric, this may be a useful inductive bias which improves sample-efficiency (e.g., see the discussion in Kerg et al. (2022)).

Algorithm 2: Multi-Head Relation (MHR) module

```
Input                 : sequence of objects: $X = (x_1, \dots, x_m) \in \mathbb{R}^d$
Hyperparameters       : Dimension of relation $d_r$, Projection dimension $d_p$, dimension of embedding $d_e$
Learnable parameters  : projection matrices $W_1^{(i)}, W_2^{(i)} \in \mathbb{R}^{d_p \times d_e},\ i = 1, \dots, d_r$, embedder network
                        $\phi : \mathbb{R}^d \to \mathbb{R}^{d_e}$
Output                : Relation tensor $R \in \mathbb{R}^{m \times m \times d_r}$

for $i, j \leftarrow 1$ to $m$ do
    for $k \leftarrow 1$ to $d_r$ do
        $R[i, j, k] \leftarrow \langle W_1^{(k)} \phi(x_i), W_2^{(k)} \phi(x_j) \rangle$
    end
end
```

2.3. *The Abstractor module: Putting it all together*

The above two sections provide a complete description of relational symbolic message-passing and computing relations via multi-head relation modules. These are the two main components of the Abstractor module.

The initial abstract state is $S = (s_1, \dots, s_m)$ with abstract symbols s_j that are task-dependent but input-independent, trainable using backpropagation. The multi-head relation module learns relations among the input objects and uses those relations to transform the abstract state. Importantly, each 'head' of the multi-head relation module encode learned relations and attributes which can be reused across tasks.

Only relational information, computed through inner products, is used to transform the abstract variables; no information about the individual object representations themselves is directly accessed by the abstract side. This enables greater out-of-distribution generalization ability since it allows for the representations of the objects to change as long as the transformed inner products are approximately preserved (see Section 4 for experiments exploring this). This is a crucial component of the idea of the relational bottleneck.

Algorithm 3 provides an algorithmic description of the archetypical Abstractor.

Note that the relation tensor output by the multi-head relation module is processed with an activation function σ_{rel}. Depending on the task, one good choice for this is the softmax activation function. This normalizes the message-passing operation such that each abstract symbol is updated as a *convex combination* of the other symbols based on the relation tensor. It is important to note that this causes the computed relation between two objects to depend also on the relations with

Algorithm 3: Abstractor

```
Input                   : sequence of objects: $X = (x_1, \dots, x_m) \in \mathbb{R}^d$
Hyperparameters         : # of layers $L$, dim of symbols $d_s$, dim of abstract objects $d_a$, hyperparameters of
                          MHR modules, activation function for relation tensor $\sigma_{\text{rel}}$, hyperparameters of
                          feedforward networks.
Learnable parameters:     symbols $S = (s_1, \dots, s_m) \in \mathbb{R}^{d_s \times m}$, feedforward networks $\phi^{(1)}, \dots, \phi^{(L)}$,
                          parameters of MHR modules.
Output                  : Abstracted sequence: $A = (a_1, \dots, a_m) \in \mathbb{R}^{d_a \times m}$

$(a_1, \dots, a_m) \leftarrow (s_1, \dots, s_m)$
for $l \leftarrow 1$ to $L$ do
    $R \leftarrow \text{MultiHeadRelation}^{(l)}(X)$
    $R \leftarrow \sigma_{\text{rel}}(R)$
    $a_i \leftarrow \sum_{j=1}^{n} R[i,j] a_j, \quad i = 1, \dots, m$
    $a_i \leftarrow \phi^{(l)}(a_i), \quad i = 1, \dots, m$
end
```

other objects (i.e., $R[i, j]$ depends not only on x_i, x_j, but on the full object sequence). Thus, softmax computes the relation between two objects *relative* to the relations with all objects. Depending on the application, this may be a very useful inductive bias or a harmful one. Alternatively, we may apply an activation function σ_{rel} independently for each entry in the relation tensor (e.g., linear, relu, sigmoid, tanh, etc.)

Finally, we remark that this operation is closely related to the multi-head attention operation of Transformers. In fact, computing the relation tensor and performing symbolic message-passing can be achieved via a multi-head attention operation of the form

$$\text{RelationalCrossAttention}\,(E, S) \equiv \text{Attention}\,(Q \leftarrow E, K \leftarrow E, V \leftarrow S)\,, \tag{2.5}$$

where E are the input objects (or some processed-encoding of them), and $S = (s_1, \dots, s_m)$ are the symbolic variables. We refer to this operation as *relational cross-attention*. This is in contrast to standard (encoder-decoder) cross-attention, which takes the form Attention $(Q \leftarrow D, K \leftarrow E, V \leftarrow E)$.

For completeness, Algorithm 4 gives an algorithmic description of the Abstractor module, cast in terms of Transformer-based attention mechanisms. Note that we have added a self-attention operation performed on the abstract symbols. This is merely to show this as an option. It may be useful for some tasks, but, unlike the rest of Abstractor, it not intuitive what this might be computing.

2.4. Relational learning using Transformers

The Abstractor module fits naturally into Transformer models. Hence, we position the Abstractor framework as an extension of Transformers. In particular, combining an Abstractor with Transformer Encoder and Decoder modules forms a powerful sequence model with enhanced abilities for relational reasoning and abstraction.

In our extended framework, processing occurs in encoder/decoder modules that handle particular types of information, separated from modules for "abstract inference." The encoders/decoders

Algorithm 4: Abstractor (cast in terms of Transformer operations)

```
Input                  : sequence of objects: $X = (x_1, \dots, x_m) \in \mathbb{R}^d$
Hyperparameters        : # of layers $L$, dim of symbols $d_s$, dim of abstract objects $d_a$, hyperparameters of
                         attention modules, hyperparameters of feedforward networks.
Learnable parameters   : symbols $S = (s_1, \dots, s_m) \in \mathbb{R}^{d_s \times m}$, feedforward networks $\phi^{(1)}, \dots, \phi^{(L)}$,
                         parameters of attention modules.
Output                 : Abstracted sequence: $A = (a_1, \dots, a_m) \in \mathbb{R}^{d_a \times m}$

$A \leftarrow S$
for $l \leftarrow 1$ to $L$ do
    $A \leftarrow$ SelfAttention$^{(l)}(A)$;
    $A \leftarrow$ RelationalCrossAttention$^{(l)}(X, S)$;
    $A \leftarrow \phi^{(l)}(A)$;
end
```

and the Abstractor modules communicate through cross-attention mechanisms that couple abstract states with specific information in encoder/decoder modules. The abstract layers are composable to include a hierarchy of abstract modules in which higher order relations are learned from lower level relations, analogous to how convolutional layers are composed in deep neural networks.

The architecture has three types of states: encoder states E, decoder states D, and abstract states A. The encoder states are vectors that represent domain-specific information (e.g., sensory or motor), which are often successfully modeled by standard deep learning frameworks, including standard Transformers. The abstract states A are vectors that are learned and processed using symbolic message-passing based on the relations between the encoder states. In particular, the encoder states are separated from the abstract states by a "relational bottleneck" that only allows information about relations (that is, inner-products) between encoder states to influence the learning and transformation of abstract states.

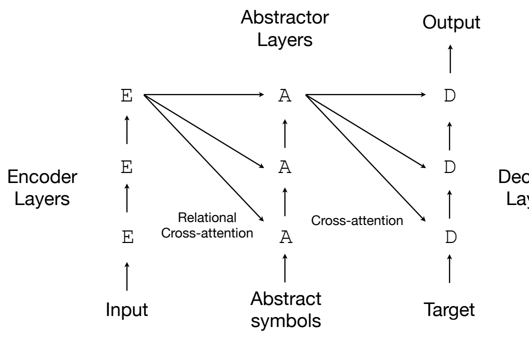


Fig 1: Algorithmic framework integrating Transformers and relational learning, implementing a form of the "relational bottleneck."

This ability to integrate with Transformers and process domain-specific information modeled by an Encoder gives the Abstractor framework greater flexibility compared to existing relational models like ESBN.

2.5. Configuring Abstractors for different tasks

Abstractors can be used to approach a variety of relational learning tasks. In the case of classification or regression, the default architecture would be

$$\texttt{Encoder} \rightarrow \texttt{Abstractor}$$

and the discriminant or regression function is computed as $f(A)$, where A is the final abstract states. For relational sequence-to-sequence tasks, the default architecture is

$$\texttt{Encoder} \to \texttt{Abstractor} \to \texttt{Decoder}$$

In a "fully relational" task, the decoder only attends to the Abstractor, and therefore only uses relational information from the input. Fully relational tasks are those which can be solved using only relational information, without any information about the individual objects. An example of a fully relational task is sorting objects; we give experimental details for this example in Section 4.

In a "partially-relational" task, the relational information is crucial, but information about individual objects is also important. Here, we propose an architecture in which the decoder attends to both the Abstractor and encoder modules. This can be done by either concatenating the encoder and Abstractor states (i.e.: attend to concat(E, A)) or to iteratively cross-attend to the Encoder and Abstractor. We call this a "sensory-connected" Abstractor.

This provides an extension of general sequence-to-sequence models with Transformers. In this paper, to highlight the capabilities of the framework, we focus our experiments on fully relational Abstractors. However, partially relational Abstractors are likely to be necessary for more realistic tasks. We hypothesize that a sensory-connected Abstractor model would yield benefits on language tasks.

We note that learning higher order relations is made possible by composing abstractors, as in the architecture

$$\texttt{Encoder} \to \texttt{Abstractor} \to \texttt{Abstractor} \to \texttt{Decoder}.$$

Since a one-layer Abstractor is able to compute a large class of functions on relations, chaining together Abstractors allows the computation of relations on relations (higher-order relations). We formalize these comments in Section 3.

3. Function classes

In this section, we analyze the class of functions which can be modeled by the Abstractor module. We do this by analyzing the class of functions of its two main components: the multi-head relation module and symbolic message-passing.

We start by presenting a universal approximation result for "inner product neural networks", as used in the multi-head relation module. This will be useful when characterizing the class of functions computable by Abstractors, but is also of independent interest more generally for relational machine learning.

3.1. Function class of inner product relations

This section analyzes the class of functions which can be modeled by "inner product relations" and the multi-head relation module. Consider vectors living in a space $\mathcal{X}$. We would like to learn a relation function $r : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^{d_r}$ which maps pairs of objects in $\mathcal{X}$ to a d_r-dimensional vector describing the relation between these objects. We specialize Equation (2.3) to the symmetric form, $\phi_i = \psi_i$:

$$R(x, y) = \begin{pmatrix} \langle \phi_1(x), \phi_1(y) \rangle \\ \vdots \\ \langle \phi_{d_r}(x), \phi_{d_r}(y) \rangle \end{pmatrix}, \tag{3.1}$$

where $\phi_1, \ldots, \phi_{d_r}$ are learnable transformations corresponding to each dimension of the relation.

In a deep learning model, a natural choice is for $\phi_1, \ldots, \phi_{d_r}$ to be d_r different neural networks (e.g., MLPs, CNNs, etc. depending on the object space $\mathcal{X}$). Hence, the parameters of R are $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_{d_r})$, where θ_i are the parameters of ϕ_i.

The following result characterizes the class of relation functions computable by (3.1) when $\phi_1, \ldots, \phi_{d_r}$ are feedforward networks. We make use of Mercer's theorem and universal approximation properties of feedforward networks to obtain a universal approximation result for (symmetric) inner product relational neural networks.

Theorem 3.1 (Function class of inner product relational neural networks)**.**

Consider an inner product relational neural network modeling a d_r-dimensional relation via inner products of neural networks,

$$\langle x, y \rangle_{MLP} := \begin{bmatrix} \langle MLP_{\theta_1}(x), MLP_{\theta_1}(y) \rangle \\ \vdots \\ \langle MLP_{\theta_{d_r}}(x), MLP_{\theta_{d_r}}(y) \rangle \end{bmatrix}.$$

Suppose the data lies in a compact Hausdorff space $\mathcal{X}$ (e.g., a metric space) with a finite countably additive measure. In particular, $\mathcal{X}$ can be any compact subset of $\mathbb{R}^d$.

Then, $\langle \cdot, \cdot \rangle_{MLP}$ is a Mercer kernel along each of the d_r dimensions.

Furthermore, for any vector-valued relation function $r : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^{d_r}$ which is a Mercer kernel in each dimension, there exists an inner product relational neural network which approximates r arbitrarily closely in the supremum norm (i.e.: uniformly over $(x, y) \in \mathcal{X} \times \mathcal{X}$). More precisely, for all $\epsilon > 0$, there exists d_r neural networks with parameters $\theta_1, \ldots, \theta_{d_r}$ such that $\sup_{x,y \in \mathcal{X}} \| r(x, y) - \langle x, y \rangle_{MLP} \|_\infty < \epsilon$.

Proof.

Denote the given relation function r by its d_r components:

$$r(x, y) = (r_1(x, y), \ldots, r_{d_r}(x, y)). \tag{3.2}$$

By assumption, r_i is a Mercer kernel for each $i = 1, \ldots, d_r$. Consider the component r_i. By Mercer's theorem (Mercer, 1909; Micchelli et al., 2006; Sun, 2005), there exists $(\psi_i)_{i \in \mathbb{N}}$, $\lambda_i \geq 0$ such that $r_i(x, y) = \sum_{i=1}^{\infty} \lambda_i \psi_i(x) \psi_i(y)$, where ψ_i and λ_i are eigenfunctions and eigenvalues of the integral operator

$$T_r : L_2(\mathcal{X}) \to L_2(\mathcal{X})$$
$$T_r(f) = \int_{\mathcal{X}} r(\cdot, x) f(x) dx.$$

Furthermore, the convergence of the series is uniform:

$$\lim_{n\to\infty} \sup_{x,y\in\mathcal{X}} |r_i(x,y) - \sum_{j=1}^{n} \lambda_j \psi_j(x)\psi_j(y)| = 0 \tag{3.3}$$

Let $\widetilde{n}_i$ be such that

$$\sup_{x,y\in\mathcal{X}} \left| r_i(x,y) - \sum_{j=1}^{n} \lambda_j \psi_j(x)\psi_j(y) \right| < \frac{\epsilon}{2} \tag{3.4}$$

Now, for $j = 1, \ldots, \widetilde{n}_i$, let the ith neural network with parameters θ_i be a function from $\mathcal{X}$ to $\widetilde{n}_i$-dimensional space. Let $(\sqrt{\lambda_1}\psi_1, \ldots, \sqrt{\lambda_{\widetilde{n}_i}}\psi_{\widetilde{n}_i})$ be the function to be approximated by the ith neural network. By the universal approximation property of neural networks, for any ϵ_1, there exists a neural network with parameters $\widehat{\theta}_i$ such that

$$\sup_{x\in\mathcal{X}} \left| (\text{MLP}(x))_j - \sqrt{\lambda_j}\psi_j(x) \right| < \epsilon_1 \tag{3.5}$$

We refer to (Barron, 1993; Cybenko, 1989; Hornik et al., 1989) for results guaranteeing the existence of neural networks which can approximate any continuous function over a bounded domain.

For ease of notation, we denote $\text{MLP}_{\widehat{\theta}_i}$ simply by MLP, omitting the dependence on fixed i. Furthermore, $\text{MLP}(x)_j$ is the jth component of the output of $\text{MLP}(x)$. Now note that the approximation error for r_i is bounded by

$$\begin{aligned}
&\sup_{x,y\in\mathcal{X}} |r_i(x,y) - \langle \text{MLP}(x), \text{MLP}(y)\rangle| \\
&\quad = \sup_{x,y\in\mathcal{X}} \left| r_i(x,y) - \sum_{j=1}^{\widetilde{n}_i} \text{MLP}(x)_j \text{MLP}(y)_j \right| \\
&\quad \le \sup_{x,y\in\mathcal{X}} \left(\left| r_i(x,y) - \sum_{j=1}^{\widetilde{n}_i} \lambda_j \psi_j(x)\psi_j(y) \right| + \left| \sum_{j=1}^{\widetilde{n}_i} \lambda_i \psi_j(x)\psi_i(y) - \text{MLP}(x)_j \text{MLP}(y)_j \right| \right)
\end{aligned} \tag{3.6}$$

The first term is less than $\frac{\epsilon}{2}$ by (3.4). The second term can be bounded uniformly on x, y by

$$\begin{aligned}
&\left| \left(\sum_{j=1}^{\widetilde{n}_i} \lambda_i \psi_j(x)\psi_i(y) \right) - \langle \text{MLP}(x), \text{MLP}(y)\rangle \right| \\
&\le \sum_{j=1}^{\widetilde{n}_i} |\lambda_i \psi_j(x)\psi_i(y) - \text{MLP}(x)_j \text{MLP}(y)_j| \\
&\le \sum_{j=1}^{\widetilde{n}_i} \left(|\text{MLP}(x)_j| \left| \sqrt{\lambda_j}\psi_j(y) - \text{MLP}(y)_j \right| + |\text{MLP}(y)_j| \left| \sqrt{\lambda_j}\psi_j(x) - \text{MLP}(x)_j \right| \right)
\end{aligned}$$

Let ϵ_1 in (3.5) be small enough such that the above is smaller than $\frac{\epsilon}{2}$. Then, by (3.6), we have that

$$\sup_{x,y \in \mathcal{X}} |r_i(x,y) - \langle \text{MLP}(x), \text{MLP}(y) \rangle| \leq \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon$$

We repeat this for each component of the relation function r_i, $i = 1, \ldots, d_r$, obtaining d_r neural networks each with parameters $\widehat{\theta}_i$. Thus, $\sup_{x,y \in \mathcal{X}} \|r(x,y) - \langle x, y \rangle_{\text{MLP}}\|_\infty < \epsilon$. □

Remark 3.1. The result also holds for universal approximators other than feedforward neural networks, with a nearly identical proof.

Theorem 3.1 shows that inner products of neural networks (of the form in the multi-head relation module) can approximate arbitrary continuous, symmetric, positive semi-definite relation functions.

This theorem characterizes the class of functions that the *symmetric* multi-head relation module can model. The class of functions in Equation (2.3) is strictly larger. As mentioned in the previous section, one way to model multi-dimensional asymmetric relations is through a single 'embedder' network ϕ, and d_r pairs of projection matrices $W_1^{(i)}, W_2^{(i)}, i = 1, \ldots, d_r$ (Equation (2.4)). This has the same function class as Equation (2.3) but enables greater weight sharing. The function class of non-symmetric inner product relations can also be characterized.

3.2. Class of relational functions computable by symbolic message-passing

In this section, we analyze the class of functions computable by symbolic message-passing as described in Algorithm 1. We view this operation as processing relations, mapping a relation tensor to a sequence of objects.

From equation (2.2), the symbolic message-passing operation is clearly bijective as a function on the input relation tensor R, for an appropriate choice of the symbol parameters $S = (s_1, \ldots, s_m)$. For example, choosing $S = I_{m \times m}$ (i.e.: the ith symbolic vector is the indicator m-vector with a 1 in the ith position, $s_i = e_i$) reproduces the relation tensor after one message-passing operation:

$$s_i' \leftarrow \sum_{j=1}^{n} R[i,j] e_j = \begin{pmatrix} R[i,1] \\ R[i,2] \\ \vdots \\ R[i,n] \end{pmatrix}.$$

More generally, one linear step of symbolic message-passing yields updated symbolic vectors such that s_i' is a linear function of the vector containing all objects' relations with object i:

$$s_i' \leftarrow S \begin{pmatrix} R[i,1] \\ \vdots \\ R[i,n] \end{pmatrix}.$$

Following the linear step in symbolic message-passing, each updated symbolic state is transformed via a neural network. Hence, the ith abstracted value after symbolic message-passing is given by

$$a_i = \phi\left(S R_i\right),$$

where ϕ is a neural network, and R_i is the vector of object i's relations with every other object, $R_i = \begin{pmatrix} R[i,1] & \cdots & R[i,n] \end{pmatrix}^\top$. Hence, a_i summarizes all the information about object i's relations to all other objects. We formalize this discussion in the following lemma, which follows from universal approximation properties of feed-forward networks.

Lemma 3.1 (Function class of symbolic message-passing)**.**

A one-step symbolic message-passing operation (in Algorithm 1) can compute arbitrary functions of a each object's relations with other objects in the input sequence. That is, there exists a choice of symbols $s_1, \dots, s_m$ and parameters of the feed-forward network such that a_i computes an arbitrary function of object i's relations, $R_i = \begin{pmatrix} R[i,1] & R[i,2] & \cdots & R[i,n] \end{pmatrix}^\top$.

Thus, the abstracted sequence after a single step of symbolic message-passing has the form

$$A^{(1)} = (a_1^{(1)}, \dots, a_m^{(1)}) = (\phi(R_1), \phi(R_2), \dots, \phi(R_m)), \tag{3.7}$$

where ϕ is an arbitrary learnable function shared by all abstracted objects, and r_i is the vector of object i's relations with every other object.

That is, $a_i^{(1)}$ summarizes object i's relations with other objects. With further symbolic message-passing operations, the ith abstracted vector can be made to represent information about other relations, not necessarily involving the ith object. For example, at the second layer, the abstracted vectors take the form

$$a_i^{(2)} = \phi^{(2)}\left(\sum_{j=1}^{n} R[i,j] a_j^{(1)}\right) = \phi^{(2)}\left(\sum_{j=1}^{n} R[i,j] \phi^{(1)}(R_j)\right). \tag{3.8}$$

3.3. *Composing Abstractors to compute relations on relations*

As described in Section 2, the Abstractor framework supports composing Abstractors in the form

$$\texttt{Encoder} \to \texttt{Abstractor} \to \cdots \to \texttt{Abstractor} \to \texttt{Output}.$$

Here, we analyze the function class generated by a composition of several Abstractors. Thus, each Abstractor models and processes the relations between the abstract symbols at the previous Abstractor. To make the analysis tractable, we assume that each Abstractor has a single-layer.

We saw in the previous section that a one-layer Abstractor is able to compute arbitrary functions of each object's relations in the sequence. Observe that the output sequence of abstracted objects is a sequence of 'relational vectors'. That is, objects which summarize relational information. Hence, chaining together a sequence of Abstractors allows the computation of relations on relations.

Lemma 3.2 (Function class for compositions of Abstractors)**.** *A chain of k single-layer Abstractors*

$$X \to \texttt{Abstractor} \to \cdots \to \texttt{Abstractor} \to A,$$

each given by Algorithm 3, is able to compute arbitrary kth order relational functions.

The precise meaning of a "kth order relational function" will be made clear in the proof below.

Proof sketch. In Section 3.2 we characterized the output of a 1-layer Abstractor as

$$a_i^{(1)} = \phi^{(1)}\left(S^{(1)} \begin{bmatrix} R^{(1)}[i,1] \\ \vdots \\ R^{(1)}[i,n] \end{bmatrix} \right) \equiv \phi^{(1)}\left(S^{(1)} R_i^{(1)} \right),$$

where the relation tensor $R^{(1)}$ is computed by a multi-head relation module applied to the input object sequence $X = (x_1, \ldots, c_m)$.

Note that we will now use the superscript to denote the order in the Abstractor composition chain rather than the layer depth within a single Abstractor (all Abstractors have a depth of one).

Let the second Abstractor's symbols be denoted by $S^{(2)} = (s_1^{(2)}, \ldots, s_m^{(2)})$. Then,

$$a_i^{(2)} = \phi^{(2)}\left(S^{(2)} \begin{bmatrix} R^{(2)}[i,1] \\ \vdots \\ R^{(2)}[i,n] \end{bmatrix} \right),$$

where $R^{(2)}$ is the relation tensor computed by the second Abstractor's multi-head relation module:

$$R^{(2)}[i,j,k] = \sigma_{\text{rel}}^{(2)}\left(\left\langle W_1^{(k)} \phi_r^{(2)}(a_i^{(1)}), W_2^{(k)} \phi_r^{(2)}(a_j^{(1)}) \right\rangle \right), \quad i,j \in [m], k \in \left[d_r^{(2)}\right],$$

where $i, j \in [m]$ indexes the relation pair, k indexes the dimension of the relation, W_1, W_2 are projection matrices, ϕ_r is the relational filter/embedder, and σ_{rel} is the relation activation function.

$R^{(2)}$ is a relation tensor which computes the relations between the abstract object output *by the previous Abstractor*, $A^{(1)} = (a_1^{(1)}, \ldots, a_m^{(1)})$. The space over which $R^{(2)}$ computes relations is itself a space of relation vectors. Recall that $a_i = \phi^{(1)}(S^{(1)} R_i^{(1)})$ summarizes object i's relations. Hence, $R^{(2)}[i,j]$ models the relation between object i's relations and object j's relations. Hence, the abstract objects at the second Abstractor can be written as

$$a_i^{(2)} = \widetilde{\phi}^{(2)}\left(\begin{bmatrix} \left\langle \widetilde{\phi}_r^{(2)}(R_i^{(1)}), \widetilde{\psi}_r^{(2)}(R_1^{(1)}) \right\rangle \\ \vdots \\ \left\langle \widetilde{\phi}_r^{(2)}(R_i^{(1)}), \widetilde{\psi}_r^{(2)}(R_m^{(1)}) \right\rangle \end{bmatrix} \right),$$

where $\widetilde{\phi}_r^{(2)}, \widetilde{\psi}_r^{(2)}$ are "relational filters" which encapsulate the projection matrices W_1, W_2 and the maps $\phi_r^{(1)}, \phi_r^{(2)}$, while $\widetilde{\phi}^{(2)}$ encapsulates the symbols $S^{(1)}, S^{(2)}$ and the maps $\sigma_{\text{rel}}^{(2)}, \phi^{(1)}, \phi^{(2)}$. By taking these maps to be the identity, the class of relation functions on the first-order relations $(R_1, \ldots, R_m)$ is readily characterized by Theorem 3.1 in terms of the parameters of the second Abstractor's multi-head relation module.

More generally, at layer l, we have

$$R^{(l)}[i,j,k] = \sigma_{\text{rel}}^{(l)}\left(\left\langle W_1^{(k)}\phi_r^{(l)}(a_i^{(l-1)}), W_2^{(k)}\phi_r^{(l)}(a_j^{(l-1)})\right\rangle\right), \quad i,j \in [m], k \in \left[d_r^{(l)}\right],$$
$$a_i^{(l)} = \phi^{(l)}\left(S^{(l)}R_i^{(l)}\right).$$

Thus, $R^{(l)}$ computes lth order-relations, and $a_i^{(l)}$ is a function processing the lth-order relations involving object i. □

3.4. Robustness and error correction

In this section, we consider the robustness of the symbolic message-passing operation to noise. Suppose a relation tensor R (say computed by a multi-head relation module) is given and transformed by symbolic message-passing via Algorithm 1. Suppose that the output of the message-passing operation is corrupted by noise. Can the relations be recovered reliably?

In linear symbolic message-passing, the symbols are transformed by $A = SR$ so that each abstract variable a_j is in the convex hull of the set of symbols. As long as S has rank m, relations are uniquely determined from the abstract symbols. Here we point out how the transformed symbols can be robust to noise if the symbols are sufficiently redundant.

Specifically, suppose that the symbols S are transformed to A and corrupted with additive noise:

$$A = SR + \Xi \tag{3.9}$$

where a fraction ϵ of the entries of Ξ are drawn from an adversarial noise distribution, and the other entries are zero; dropout noise is also possible. This can be studied as an instance of compressed sensing and "model repair" (Candès and Randall, 2018; Gao and Lafferty, 2020). In particular, the relations can be recovered using the robust regression estimator

$$\widehat{r}_j = \operatorname*{argmin}_{u \in \mathbb{R}^m} \|a_j - Su\|_1 \tag{3.10}$$

where $A = (a_1, a_2, \dots, a_m)$ with columns $a_j \in \mathbb{R}^d$. The main lemma in Gao and Lafferty (2020) states that the following two conditions suffice:

Condition A: There exists some σ^2, such that for any fixed $c_1, ..., c_d$ satisfying $\max_i |c_i| \le 1$,

$$\left\| \frac{1}{d}\sum_{i=1}^{d} c_i s_{i\cdot} \right\|^2 \le \frac{\sigma^2 m}{d}, \tag{3.11}$$

with high probability, where $s_{i\cdot} \in \mathbb{R}^m$ is the ith row of S.

Condition B: There exist $\underline{\kappa}$ and $\overline{\kappa}$, such that

$$\inf_{\|\Delta\|=1} \frac{1}{d}\sum_{i=1}^{d} |s_{i\cdot}^T \Delta| \quad \ge \quad \underline{\kappa}, \tag{3.12}$$

$$\sup_{\|\Delta\|=1} \frac{1}{d}\sum_{i=1}^{d} |s_{i\cdot}^T \Delta|^2 \quad \le \quad \overline{\kappa}^2, \tag{3.13}$$

with high probability.

Theorem 3.2. *Assume the symbol matrix S satisfies Condition A and Condition B. Then if*

$$\frac{\overline{\kappa}\sqrt{\frac{m}{d}\log\left(\frac{ed}{k}\right)} + \epsilon\sigma\sqrt{\frac{m}{d}}}{\underline{\kappa}(1-\epsilon)} \tag{3.14}$$

is sufficiently small, the linear program (3.10) *recovers R, so that $\widehat{r}_j = r_j$ with high probability.*

The condition is essentially that

$$\frac{1}{1-\epsilon}\sqrt{\frac{m}{d}} \tag{3.15}$$

is small, meaning that the dimension d of the symbols needs to be sufficiently large relative to the dimension k of the relation.

3.5. Sparse, high-dimensional relations

The above setting ensures enough redundancy to recover the relations, constraining the number of symbols k to be small relative to the symbol dimension d. This is not appropriate in the situation where the relations are over a large number m of elements, for example, the contents of the entire episodic memory. In this setting we assume that the relation tensor $R \in \mathbb{R}^{m\times m}$ is sparse; that is, each column $r_j \in \Delta_m$ has at most k nonzero entries: $\|r_j\|_0 \leq k$. To recover the relation we now use the robust lasso estimator, which is a related linear program

$$\widehat{r}_j = \operatorname*{argmin}_{u\in\mathbb{R}^m} \|a_j - Su\|_1 + \lambda\|u\|_1. \tag{3.16}$$

Here we have an analogous theorem, stating that if

$$\frac{\overline{\kappa}/\underline{\kappa}}{1-\epsilon}\sqrt{\frac{k}{d}\log(2m)} \leq c, \tag{3.17}$$

for some sufficiently small constant $c > 0$, the robust lasso estimator (3.16) satisfies

$$\|\widehat{r}_j - r_j\| \leq C\frac{\overline{\kappa}/\underline{\kappa}^2}{1-\epsilon}\sqrt{\frac{\sigma^2 k}{d}\log(2m)} \tag{3.18}$$

for some constant C. This implies that we can accurately recover the relation tensor in the high dimensional setting, even when many of the entries of the transformed abstract symbols are corrupted.

The above discussion shows how the relation tensor can be recovered from the transformed symbols, even under adversarial noise, assuming there is sufficient redundancy in the symbols. This implies that it is possible to predict as well from the transformed symbols as from the relations, without explicitly recovering the relations. Using ideas from (Hand and Voroninski, 2017; Song et al., 2019), it may be possible to extend this theory to nonlinear mappings $y = \varphi(Au) + \eta$ where $\varphi(\cdot)$ is an activation function.

4. Experiments

4.1. Warm up: Ability to learn asymmetric and multi-dimensional relations

One recent work on relational machine learning is Kerg et al. (2022) where, based on prior work of Webb et al. (2021), the authors argue for a particular type of inductive bias in relational models and propose CoRelNet. The architecture is: given a sequence of objects $(x_1, \dots, x_m)$, embed them using an MLP ϕ, then compute the similarity matrix $R = \text{Softmax}(A), A = \left[\langle \phi(x_i), \phi(x_j) \rangle\right]_{ij}$. The final output is an MLP applied to the flattened similarity matrix. They demonstrate that this model can solve a series of simple tasks with high sample-efficiency compared to models like ESBN and standard Transformers. However, CoRelNet has some notable limitations. One is that, as described, it is only able to model symmetric relations[2]—R is symmetric by definition. Another limitation is that it can only model single-dimensional relations—for each pair of objects (i, j), their modeled relation is a single-dimensional scalar R_{ij}. The Abstractor is able to model a significantly larger class of relations. In particular, it is able to model asymmetric and multi-dimensional relations through the MultiHeadRelation operation. This is demonstrated by the following simple experiment. Note that this is merely intended as a warm up; we don't wish to imply that this task is not solvable by standard models which lack a relational bottleneck.

We generate $N = 32$ "random objects" represented by iid Gaussian vectors, $o_i \overset{iid}{\sim} \mathcal{N}(0, I_d) \in \mathbb{R}^d$, and associate an order relation to them $o_1 \prec o_2 \prec \cdots \prec o_N$. We train several different relational models to learn this order relation. Note that $\prec$ is *not symmetric*. Of the $N^2 = 1024$ possible pairs (o_i, o_j), 15% are held out as a validation set (for early stopping) and 35% as a test set. We evaluate learning curves by training on the remaining 50% and computing accuracy on the test set (10 trials for each training set size). Note that under this set up, we are evaluating the models on pairs they have never seen. Thus, the models will need to generalize based on the transitivity of the $\prec$ relation. We observe that a simple Abstractor model is able to learn the relation while CoRelNet cannot (Figure 2a).

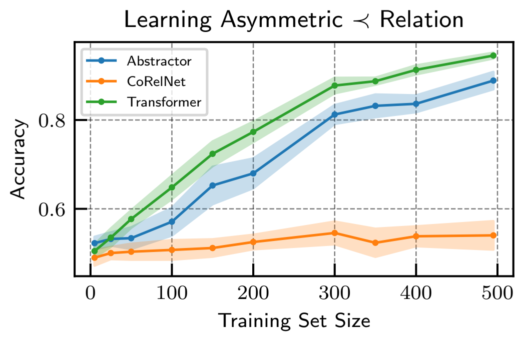
For completeness, we also compare to a standard Transformer model, which performs slightly better than the Abstractor on this simple task. We include this to highlight that an Abstractor will not necessarily be superior to a Transformer uniformly on all tasks, but rather that it can realize significant gains on tasks with complex relational structure, as demonstrated in the following experiments.

4.2. Superior sample-efficiency on relational tasks compared to plain Transformers

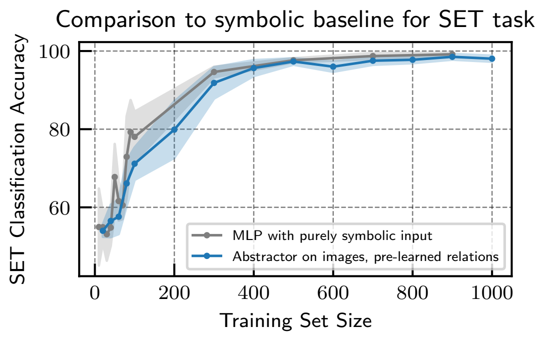
The next experiment extends the idea of learning an order relation $\prec$ on random objects: now, the task is to fully sort sequences of randomly permuted random objects.

[1]We ran the experiments described here on RTX 2080ti, RTX 3090, and A100 GPUs, available to us through our institution's internal cluster. The models here are relatively small; powerful GPUs are not required to train a single model. We found the use of GPUs useful for evaluating learning curves over several trials.

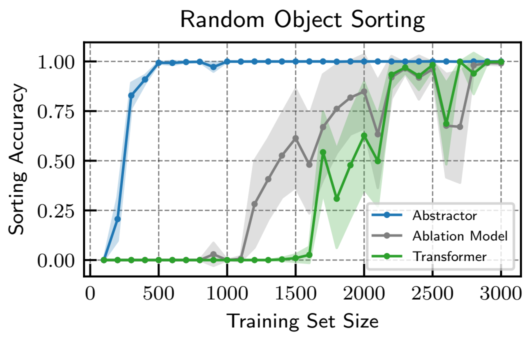
[2]This is not fundamental to the CoRelNet model, which can learn asymmetric relations via the natural modification $A = \left[\langle W_1 \phi(x_i), W_2 \phi(x_j) \rangle\right]_{ij}$, where W_1, W_2 are trainable matrices.



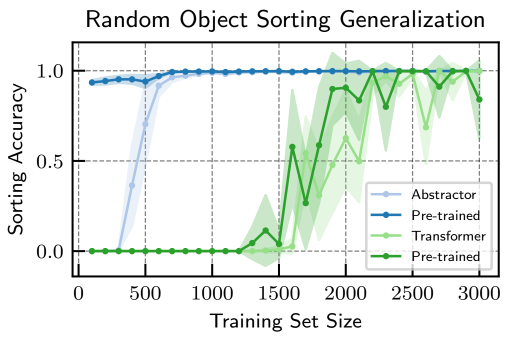
(a) The Abstractor learns the transitive ≺ relation and generalizes, whereas CoRelNet's learning curve is flat at the baseline accuracy of 0.5.



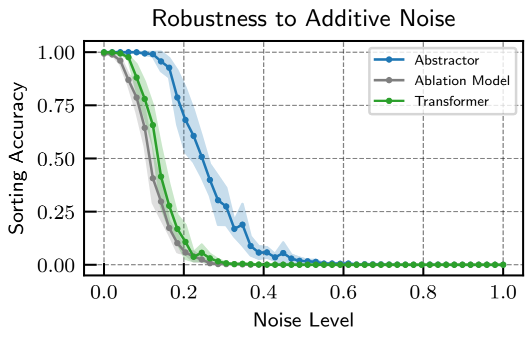
(b) Comparison of Abstractor trained on images of cards and MLP with relations hand-encoded symbolically as bit vectors.



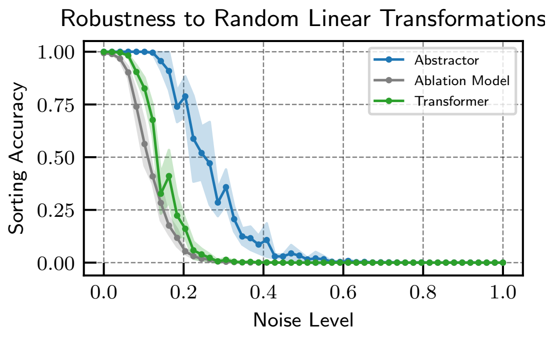
(c) Learning curves on sorting sequences of random objects. The Abstractor is dramatically more sample-efficient.



(d) Learning curves with and without pre-training on a similar sorting task. The Abstractor benefits significantly from pre-training.



(e) The Abstractor is more robust to corruption by additive noise.



(f) The Abstractor is more robust to corruption by a random linear transformation.

Fig 2: Experiments. Shaded regions indicate twice the standard error of the mean.

We generate random objects in the following way. First, we generate two sets of random attributes $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$, $a_i \overset{iid}{\sim} \mathcal{N}(0, I) \in \mathbb{R}^4$ and $\mathcal{B} = \{b_1, \dots, b_{12}\}$, $b_i \overset{iid}{\sim} \mathcal{N}(0, I) \in \mathbb{R}^8$. To each set of attributes, we associate the strict ordering relation $a_1 \prec a_2 \prec a_3 \prec a_4$ and $b_1 \prec b_2 \prec \cdots \prec b_{12}$, respectively. Our random objects are formed by the Cartesian product of these two attributes $\mathcal{O} = \mathcal{A} \times \mathcal{B}$, yielding $N = 4 \times 12 = 48$ objects (i.e., each object in $\mathcal{O}$ is a vector in $\mathbb{R}^{4+8}$ formed by a concatenation of one attribute value in $\mathcal{A}$ and one in $\mathcal{B}$). Then, we associate with $\mathcal{O}$ the strict ordering relation corresponding to the order relation of $\mathcal{A}$ as the primary key and the order relation of $\mathcal{B}$ as the secondary key. i.e., $(a_i, b_j) \prec (a_k, b_l)$ if $a_i \prec a_k$ or if $a_i = a_k$ and $b_j \prec b_l$.

Given a set of objects in $\mathcal{O}$, the task is to sort it according to $\prec$. More precisely, the input sequences are randomly permuted sequences of 10 objects in $\mathcal{O}$ and the target sequences are the indices of the object sequences in sorted order (i.e., the 'argsort'). The training data are sampled uniformly from the set of length-10 sequences in $\mathcal{O}$. We also generate a non-overlapping validation dataset (used during training for early stopping) and a testing dataset (used during evaluation).

We evaluate learning curves on an Abstractor, a Transformer, and an "Ablation" model (10 trials for each training set size). The Abstractor uses the architecture `Encoder` $\to$ `Abstractor` $\to$ `Decoder`. The Encoder-to-Abstractor interface uses relational cross-attention and the Abstractor-to-Decoder interface uses standard cross-attention. The Ablation Model aims to test the effects of the relational cross-attention in the Abstractor model—it is architecturally identical to the Abstractor model with the crucial exception that the Encoder-to-Abstractor interface instead uses standard cross-attention. The hyperparameters of the models are chosen so that the parameter counts are similar. We find that the Abstractor is dramatically more sample-efficient than the Transformer and the Ablation model (Figure 2c).

4.3. Ability to generalize to similar tasks

Continuing with the object-sorting task and the dataset generated as described above, we test the Abstractor's ability to generalize from similar relational tasks through pre-training. The main task uses the same dataset described above. The pre-training task involves the same object set $\mathcal{O}$ but the order relation is changed. The ordering in attribute $\mathcal{A}$ is randomly permuted, while the ordering in attribute $\mathcal{B}$ is kept the same. A strict ordering relation $\prec$ on $\mathcal{O}$ is obtained in the same way—using the order in $\mathcal{A}$ as the primary key and the order in $\mathcal{B}$ as the secondary key.

The Abstractor model here uses the architecture `Abstractor` $\to$ `Decoder` (i.e., no Transformer encoder), and the Transformer is the same as the previous section. We pre-train both models on the pre-training task and then, using those learned weights for initialization, evaluate learning curves on the original task. Since the Transformer requires more training samples to learn the object-sorting task, we use a pre-training set size of 3,000, chosen based on the results of the previous subsection so that it is large enough for the Transformer to learn the pre-training task. This experiment assesses the models' ability to generalize relations learned on one task to a new task. Figure 2d shows the learning curves for each model with and without pre-training. We observe that when the Abstractor is pre-trained, its learning curve on the object-sorting task is significantly accelerated. The Transformer does not benefit from pre-training.

4.4. Robustness and Out-of-Distribution generalization

In this experiment, we evaluate robustness to a particular type of noisy corruption. We train each model on the same object-sorting task described above. We use a fixed training set size of 3,000 for the same reason —it is large enough that all models are able to learn the task. On the hold out test set, we corrupt the object representations by applying a random linear transformation. In particular, we randomly sample a random matrix the entries of which are iid zero-mean Gaussian with variance σ^2, $\Phi \in \mathbb{R}^{d\times d}, \Phi_{ij} \sim \mathcal{N}(0, \sigma^2)$. Each object in $\mathcal{O}$ is then corrupted by this random linear transformation, $\widetilde{o}_i = \Phi o_i, \quad$ for each $i \in [48]$. We also test robustness to additive noise via $\widetilde{o}_i = o_i + \varepsilon_i, \varepsilon_i \sim \mathcal{N}(0, \sigma^2 I_d)$.

The models are evaluated on the hold-out test set with objects replaced by their corrupted version. We evaluate the sorting accuracy of each model while varying the noise level σ (5 trials at each noise level). The results are shown in figures 2e and 2f. We emphasize that the models are trained only on the original objects in $\mathcal{O}$, and are not trained on objects corrupted by any kind of noise.

This experiment can be interpreted in two lights: the first is robustness to noise. The second is a form of out-of -distribution generalization. Note that the objects seen by the models post-corruption lie in a different space than those seen during training. Hence the models need to learn relations that are in some sense independent of the value representation. As a theoretical justification for this behavior, Zhou et al. (2009) shows that $\langle \Phi x, \Phi y\rangle \approx \langle x, y\rangle$ in high dimensions, for a random matrix Φ with iid Gaussian entries. This indicates that models whose primary computations are performed via inner products, like Abstractors, may be more robust to this kind of corruption.

4.5. Modularity and comparison to purely symbolic representations

SET is a relatively straightforward but challenging cognitive task that engages reasoning faculties in a deliberative, attentionally directed manner, requiring several levels of abstraction over sensory embeddings. Players are presented with 12 cards, each of which contains figures that vary along four dimensions (color, number, pattern, and shape) and they must find subsets of three cards which obey a deceptively simple rule: along each dimension, all cards in a SET must either have the same or unique values. For example, in the figure to the right, the cards with two solid blue/purple diamonds, two striped blue squiggles, and two open blue oblongs form a SET: same color, same number, different patterns, different shapes.

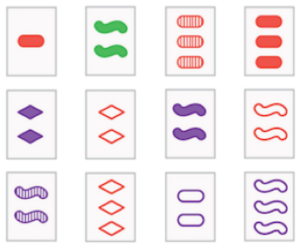


Fig 3: The SET game

To simulate the task of deciding if a triple forms a SET, we first train a convolutional neural network to process the color images of the cards (a full deck includes 81 cards). The CNN is trained to predict the attribute of each card, as a multi-label classification, and then an embedding of dimension $d = 32$ of each card is obtained. This embedding layer uses an MLP to map the convolutional feature maps into a distributed representation. Next, we train Abstractors separately for each of the four attributes to learn same/different relations, where the task is to decide if an input pair of cards is the same or different for that attribute. We then use the query and key mappings W_Q and W_K learned for these relations to initialize the relations in a multi-head Abstractor. The Abstractor is then trained on a dataset of triples of cards, half of which form a SET.

This is compared to a baseline symbolic model where, instead of images, the input is a vector with 12 bits, explicitly encoding the relations. That is, for each of the four attributes, a binary symbol is computed for each pair of three input cards—1 if the pair is the same in that attribute, and 0 otherwise. A two-layer MLP is then trained to decide if the triple forms a SET. The MLP using the symbolic representation can be considered as a lower bound on the performance achievable by the Abstractor. This comparison shows that the Abstractor is able to solve a task using relations learned in other tasks (modularity), with a sample efficiency that is not far from that obtained with purely symbolic, noise-free encodings of the relevant relations.

This subtask suggests how Abstractors might be viewed as an intermediate between strong "nativist" approaches that assume a pre-existing foundation of symbolic primitives and purely "empiricist" approaches that assume similar capabilities can emerge simply from processing large amounts of data (Lake et al., 2015).

5. Discussion

In this work we have proposed a framework which extends Transformer models to naturally support types of relational learning, through cross-attention mechanisms that enforce a relational bottleneck, so that only information about relations between encoder states are used in transformations of the abstract states. Building on insights gained from the implementation of a relational bottleneck in other forms (Kerg et al., 2022; Webb et al., 2021), this exploits the powerful attentional capabilities of the Transformer architecture to identify relevant relationships. Experiments with sorting and other relational tasks indicate that this framework has the potential to combine the benefits of function approximation over sensory states, as exploited in many deep learning models, with abstraction and relational reasoning abilities supported by symbolic processing. Limitations of the current approach suggest interesting future directions to better understand the potential of this framework, and how it may relate to the algorithms of human cognition as implemented in the brain, opening up possibilities for improved alignment of natural and artificial intelligences.

Two directions to highlight are the use of external memories and attentional control. From a statistical perspective, the use of an external memory with relational cross attention effectively allows nonparametric and semiparametric models. To make this explicit, note that the classical kernel regression estimator for the Gaussian kernel can be seen as relational cross-attention with relations and weights given by the kernel and values $y_{1:n}$. Viewed in terms of episodic memory, this has bindings $\{x_i \| y_i\}$ with values y_i that might be seen as being stored on the abstract side if they are rewards or labels that are associated with a particular episode x_i. Under a learned relation, the model associates the reward with particular features or attributes of the inputs, as the kernel changes to compute relations as inner products $\langle W_Q x, W_K x_i \rangle$, leading to a semiparametric model. The number of keys grows unboundedly with the number of episodes experienced, while the query remains of constant size. Realizing this approach will allow a regulation of the bias-variance tradeoff through the way that attention is implemented, while limiting the number of parameters that need to be learned.

A second direction, also motivated from cognitive neuroscience principles, is to replace parallel execution of attention heads with serial evaluation as directed by a controller. In standard Transformers, attention operations are spread across multiple attention heads, each of which is restricted in scope, and that are evaluated in parallel across GPUs by embedding them in matrix multiplication. This is a powerful, but energy inefficient approach, which also reduces the pressure for learned representations to be abstract and attentional operations to be generalizable, in ways that are exhibited by the flexibility of human cognition. It will be important to add a "cognitive control" mechanism that resides on the abstract side and is responsible for selecting attention heads to be evaluated in each step in a serial fashion (Cohen, 2017). This could be seen as analogous to evaluative, gating and updating functions in anterior cingulate cortex, prefrontal cortex and basal ganglia (Braver and Cohen, 2000; Frank et al., 2001; O'Reilly and Frank, 2006; Shenhav et al., 2013) and implemented using neural network mechanisms such as LSTMs (Hochreiter and Schmidhuber, 1997) or variations on the Transformer architecture (Vaishnav and Serre, 2023).

Acknowledgments

Research supported in part by NSF grant CCF-1839308 and a Vannevar Bush Faculty Fellowship to JDC.

References

Barrett, D. G., Hill, F., Santoro, A., Morcos, A. S., and Lillicrap, T. (2018). Measuring abstract reasoning in neural networks. In *Proceedings ICML*.

Barron, A. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks. arXiv:1806.01261.

Berg, E. A. (1948). A simple objective technique for measuring flexibility in thinking. *J. Gen. Psychol.*, 39:15–22.

Braver, T. S. and Cohen, J. D. (2000). On the Control of Control: The Role of Dopamine in Regulating Prefrontal Function and Working Memory. In *Control of Cognitive Processes: Attention and Performance XVIII*. The MIT Press.

Candès, E. J. and Randall, P. A. (2018). Highly robust error correction by convex programming. *IEEE Transactions on Information Theory*, 54(7):2829–2840.

Cohen, J. D. (2017). Cognitive control: Core constructs and current considerations. *The Wiley handbook of cognitive control*, pages 1–28.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314.

Frank, M. J., Loughry, B., and O'Reilly, R. C. (2001). Interactions between frontal cortex and basal ganglia in working memory: a computational model. *Cognitive, Affective, & Behavioral Neuroscience*, 1(2):137–160.

Gao, C. and Lafferty, J. (2020). Model repair: Robust recovery of overparameterized statistical models. arXiv:2005.09912.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural Turing machines. arXiv:1410.5401.

Hand, P. and Voroninski, V. (2017). Global guarantees for enforcing deep generative priors by empirical risk. *arXiv*, abs/1705.07576.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Holyoak, K. J. (2012). Analogy and relational reasoning. In Holyoak, K. J. and Morrison, R. G., editors, *The Oxford Handbook of Thinking and Reasoning*. New York: Oxford University Press.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

Kerg, G., Kanuparthi, B., Goyal, A., Goyette, K., Bengio, Y., and Lajoie, G. (2020). Untangling tradeoffs between recurrence and self-attention in artificial neural networks. *Advances in Neural Information Processing Systems*, 33:19443–19454.

Kerg, G., Mittal, S., Rolnick, D., Bengio, Y., Richards, B., and Lajoie, G. (2022). On neural architecture inductive biases for relational tasks. *arXiv preprint arXiv:2206.05056*.

Kumaran, D., Hassabis, D., and McClelland, J. L. (2016). What Learning Systems do Intelligent Agents Need? Complementary Learning Systems Theory Updated. *Trends in Cognitive Sciences*, 20(7):512–534.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

Mahowald, K., Ivanova, A. A., Blank, I. A., Kanwisher, N., Tenenbaum, J. B., and Fedorenko, E. (2023). Dissociating language and thought in large language models: A cognitive perspective. *arXiv preprint arXiv:2301.06627*.

McClelland, J. L., McNaughton, B. L., and O'Reilly, R. C. (1995). Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102(3):419–457.

Mercer, J. (1909). Functions of Positive and Negative Type, and their Connection with the Theory of Integral Equations. *Philosophical Transactions of the Royal Society of London. Series A,* *Containing Papers of a Mathematical or Physical Character*, 209:415–446. Publisher: The Royal Society.

Micchelli, C. A., Xu, Y., and Zhang, H. (2006). Universal kernels. *J. Mach. Learn. Res.*, 7:2651–2667.

Monchi, O., Petrides, M., Petre, V., Worsley, K., and Dagher, A. (2001). Wisconsin card sorting revisited: Distinct neural circuits participating in different stages of the task identified by event-related functional magnetic resonance imaging. *Jour. Neuroscience*, 21(19):7733–7741.

Mondal, S. S., Webb, T. W., and Cohen, J. (2023). Learning to reason over visual objects. In *International Conference on Learning Representations*.

O'Reilly, R. C. and Frank, M. J. (2006). Making working memory work: A computational model of learning in the prefrontal cortex and basal ganglia. *Neural computation*, 18(2):283–328.

Pritzel, A., Uria, B., Srinivasan, S., Badia, A. P., Vinyals, O., Hassabis, D., Wierstra, D., and Blundell, C. (2017). Neural episodic control. In *International conference on machine learning*, pages 2827–2836. PMLR.

Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. (2017). A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4967–4976.

Shenhav, A., Botvinick, M. M., and Cohen, J. D. (2013). The expected value of control: An integrative theory of anterior cingulate cortex function. *Neuron*, 79(2):217–240.

Snow, R. E., Kyllonen, P. C., and Marshalek, B. (1984). The topography of ability and learning correlations. In Sternberg, R. J., editor, *Advances in the psychology of human intelligence*, volume 2, pages 47–103.

Song, G., Fan, Z., and Lafferty, J. (2019). Surfing: Iterative optimization over incrementally trained deep networks. In *Proceedings of Neural Information Processing Systems (NeurIPS)*. arXiv:1907.08653.

Sun, H. (2005). Mercer theorem for RKHS on noncompact sets. *J. Complexity*, 21(3):337–349.

Vaishnav, M. and Serre, T. (2023). GAMR: A guided attention model for (visual) reasoning. In *The Eleventh International Conference on Learning Representations (ICLR)*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Webb, T., Holyoak, K. J., and Lu, H. (2022). Emergent analogical reasoning in large language models. arXiv:2212.09196.

Webb, T., Sinha, I., and Cohen, J. D. (2021). Emergent symbols through binding in external memory. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Whittington, J. C., Muller, T. H., Mark, S., Chen, G., Barry, C., Burgess, N., and Behrens, T. E. (2020). The Tolman-Eichenbaum machine: Unifying space and relational memory through generalization in the hippocampal formation. *Cell*, 183(5):1249–1263.e23.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

Zhou, S., Lafferty, J., and Wasserman, L. (2009). Compressed and privacy-sensitive sparse regression. *IEEE Trans. Information Theory*, 55(2):846–866.