Relational reasoning and generalization using non-symbolic neural networks

Atticus Geiger[a,1], **Alexandra Carstensen**[b], **Michael C. Frank**[b], **and Christopher Potts**[a]

[a]Department of Linguistics, Stanford University; [b]Department of Psychology, Stanford University

This manuscript was compiled on July 27, 2020

Humans have a remarkable capacity to reason about abstract relational structures, an ability that may support some of the most impressive, human-unique cognitive feats. Because equality (or identity) is a simple and ubiquitous relational operator, equality reasoning has been a key case study for the broader question of abstract relational reasoning. This paper revisits the question of whether equality can be learned by neural networks that do not encode explicit symbolic structure. Earlier work arrived at a negative answer to this question, but that result holds only for a particular class of hand-crafted feature representations. In our experiments, we assess out-of-sample generalization of equality using both arbitrary representations and representations that have been pretrained on separate tasks to imbue them with abstract structure. In this setting, even simple neural networks are able to learn basic equality with relatively little training data. In a second case study, we show that sequential equality problems (learning ABA sequences) can be solved with only positive training instances. Finally, we consider a more complex, hierarchical equality problem, but this requires vastly more data. However, using a pretrained equality network as a modular component of this larger task leads to good performance with no task-specific training. Overall, these findings indicate that neural models are able to solve equality-based reasoning tasks, suggesting that essential aspects of symbolic reasoning can emerge from data-driven, non-symbolic learning processes.

relational reasoning | neural networks | generalization

One of the key components of human intelligence is our ability to reason about abstract relations between stimuli. Many of the most unremarkable human activities – scheduling a meeting, following traffic signs, assembling furniture – require a fluency with abstraction and relational reasoning that is unmatched in nonhuman animals. An influential perspective on human uniqueness holds that relational concepts are critical to higher-order cognition (e.g., 1). By far the most common case study of abstract relations has been equality.* Equality is a valuable case study because it is simple and ubiquitous, but also completely abstract in the sense that it can be evaluated regardless of the identity of the stimuli being judged.

Equality reasoning has been studied extensively across a host of systems and tasks, with wildly variant conclusions. In some studies, equality is very challenging to learn: only great apes with either extensive language experience or specialized training succeed in matching tasks in which a *same* pair, AA, must be matched to a novel same pair, BB (2, 3). Preschool children struggle to learn these regularities in a seemingly similar task (4). In contrast, other studies suggest that equality is simple: bees are able to learn abstract identity relationships from only a small set of training trials (5, 6), and human infants can generalize identity patterns (7) and succeed in relational matching tasks (8). We take the central challenge of this literature to be characterizing the conditions that lead to success or failure in learning an abstract relation in a way that can be productively generalized to new stimuli.

The learning task in all of these cases can be described using the predicate *same* (or equivalently, =), which operates over two inputs and returns TRUE if they are identical in some respect. One perspective in the literature is that success in these learning tasks implies the presence of an equivalent symbolic description in the mind of the solver (2, 9). This view does not provide a lever to distinguish which of these tasks are trivial and which are difficult, however. Further, it can fall prey to circularity: because newborns show sensitivity to identity relations (10), then it would follow from this argument that they must have symbolic representations. If this logic applies also to bees, then we presuppose symbolic representations universally and have no account of the gradient difficulty of different tasks for different species.

An explanation of when same–different tasks are trivial and when they are difficult requires a theoretical framework beyond the symbolic/non-symbolic distinction. To make quantitative predictions about task performance, such a framework should ideally be instantiated in a computational model that takes in training data and learns a solution that generalizes when assessed with stimuli analogous to those used in experimental assessments. Symbolic computational models (e.g., 11) can be used to make contact with data about the breadth of generalization, but they require the existence of a symbolic equality

Significance Statement

The relation of equality has served as a key case study for broader questions of abstract relational reasoning. Earlier work concluded that neural networks are incapable of learning general solutions to equality-based reasoning tasks. This claim is seriously in tension with the recent revolutionary success of neural networks in artificial intelligence. We resolve this tension by demonstrating these previous negative results hold only when neural networks are provided hand-crafted feature representations. When neural networks are instead provided the non-featural representations that are now the norm in state-of-the-art artificial intelligence, these models easily learn general solutions to equality-based reasoning tasks, suggesting that essential aspects of symbolic reasoning can emerge from non-symbolic learning processes.

All four authors contributed to writing the paper and experimental design. Atticus Geiger and Chris Potts wrote the code that was used to run the experiments.

The authors declare no conflicts of interest.

*We use the term "equality" here, though different literatures have also used "identity."

[1]To whom correspondence should be addressed. E-mail: atticusg@stanford.edu

predicate and hence again presuppose symbolic abilities in every case of success. Ideally, we would want a model that describes under what conditions *same* is easy and under what conditions it is hard or unlearnable – and how learning proceeds in these hard cases. Here, we aim to lay the foundation for the development of such an account.

We are inspired by an emergent perspective in the animal learning literature that the representations underlying non-human animals' and human infants' successes in equality reasoning tasks are graded (12). This view acknowledges the increasing evidence that other species like pigeons (13), crows (14), and baboons (15) can make true, out-of-sample generalizations of *same* and *different* relations, but it also recognizes that the observed patterns of behavior do not show the hallmarks of all-or-none symbolic representations. Instead, performance is graded. Out-of-sample generalization is possible but the level of performance depends critically on the diversity of the training stimuli (e.g., 16). Success requires hundreds, thousands, or even tens of thousands of training trials. And the outcome of learning is noisy and imperfect.

These learning signatures appear to be a close match to the kind of learning exhibited by neural networks, a flexible framework for arbitrary function learning that has enjoyed a huge resurgence of interest in recent years (17). Contra this proposal, however, (9) argued that a broad class of recurrent neural networks were unable to learn equality relations. These claims were subsequently challenged by the presentation of evidence that neural networks are able to learn (at least aspects of) the tasks that (9) posed (18–22). The subsequent debate (reviewed in 23) revealed a striking lack of consensus on some of the ground rules regarding what sort of generalization would be required to show that the learned function was suitably abstract. In addition, only a narrow range of network architectures and representations was explored.

We revisit this debate here, adopting stringent criteria for generalization and considering a broader range of representations across three models. We model three cases of equality-based reasoning that have featured prominently in discussions of the role of symbols in relational reasoning (Figure 1): (1) learning to discriminate pairs of objects that exemplify the relation *same* or *different*, (2) learning sequences with repeated *same* elements (9), and (3) learning to distinguish hierarchical *same* and *different* relations in a context with pairs of pairs exemplifying these relations (2). We show how using pretraining – an active area of development in recent artificial intelligence research (24–28) – can help achieve far faster learning on the more difficult hierarchical task.

Across these three models, we find strong support for their ability to learn equality relations. These results should serve to revise the conclusions of the earlier debate. Marcus and colleagues (9, 29) showed experimentally that neural networks using feature representations cannot generalize to binary features unseen in training. We agree with this claim (and support it with a direct mathematical argument in Appendix). However, they concluded from this result that neural networks will need to have primitive symbolic operators to solve hard relational reasoning tasks. On this point, we disagree. Our experiments show that networks without these primitives can solve a range of these tasks, as long as they use non-featural representations. Overall, these findings suggest that essential aspects of symbolic reasoning can emerge from entirely data-driven, non-symbolic learning processes.

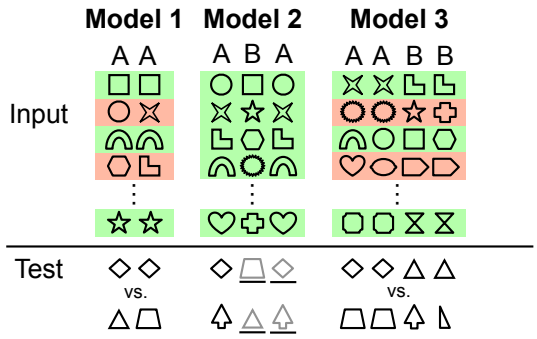


Fig. 1. Relational reasoning tasks. Green and red mark positive and negative training examples, respectively. The sequential task (Model 2) uses only positive instances, and a model succeeds if, prompted with α, it produces a sequence $\beta\ \alpha$ where $\beta \neq \alpha$. For the hierarchical task (Model 3), we show that a model trained on the basic task (Model 1) is effective with no additional training.

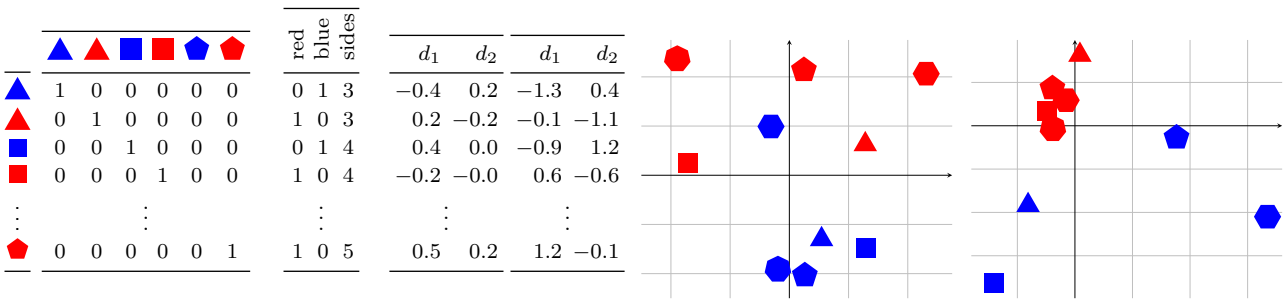
Designing theoretical models of equality learning

We begin by discussing two critical design considerations for our models: (1) the standards for generalization by which models should be evaluated and (2) the type of representations they should use. To summarize this discussion: we select generalization tasks with fully disjoint training and test vocabularies to provide the most stringent test of generalization. Further, we adopt both randomly initialized non-featural representations and pretrained non-featural representations for our subsequent models, and we show analytically that other kinds of representations are more limited in their capacity to make successful out-of-sample generalizations.

Generalization. The standard approach to training and evaluating neural networks is to choose a dataset, divide it randomly into training and assessment sets, train the system on the training set, and then use its performance on the assessment set as a proxy for its capacity to generalize to new data.

The standard approach is fine for many purposes, but it raises concerns in a context in which we are trying to determine whether a network has truly acquired a global solution to a target function. In particular, where there is any kind of overlap between the training and assessment vocabularies (primitive elements), we can't rule out that the network might be primarily taking advantage of idiosyncrasies in the underlying dataset to effectively cheat – to memorize aspects of the training set and learn a local approximation of the target function that happens to provide traction during assessment.

To address this issue, we follow (9) in proposing that networks must be evaluated on assessment sets that are completely disjoint in every respect from the train set, all the way down to the entities involved. For example, below, we train on pairs (a, a) and (a, b), where a and b are representations from a train vocabulary V_T. At test time, we create a new assessment vocabulary V_A, derive equality and inequality pairs (α, α) and (α, β) from that vocabulary, and assess the trained network on these new examples. In adopting these methods, we get a clear picture of the system's capacity to generalize, and we can



	1	0	0	0	0	0
	0	1	0	0	0	0
	0	0	1	0	0	0
	0	0	0	1	0	0
⋮			⋮			
	0	0	0	0	0	1

red	blue	sides
0	1	3
1	0	3
0	1	4
1	0	4
⋮		
1	0	5

d_1	d_2
−0.4	0.2
0.2	−0.2
0.4	0.0
−0.2	−0.0
⋮	
0.5	0.2

d_1	d_2
−1.3	0.4
−0.1	−1.1
−0.9	1.2
0.6	−0.6
⋮	
1.2	−0.1

(a) **(b)** Localist. **(c)** Distributed. **(d)** Random. **(e)** Pretrained. **(f)** The matrix in Figure 2d. **(g)** The matrix in Figure 2e.

Fig. 2. Each matrix is a method for representing the shapes in Figure 2a, where each row is a vector representation of one shape. Localist and distributed are feature representations where each vector dimension encodes the value of a single property. Random and pretrained are non-featural representations where the values of properties are encoded implicitly in two dimensions. Random representations and localist representations encode only identity, whereas distributed representations and pretrained representations encode color and number of sides.

safely say that its performance during assessment is a window into whether a global solution to identity has been learned. This is a very challenging setting for any machine learning model. For the sequential same–different task, we will see that it even requires us to depart from usual model formulations.

Representations. Essentially all modern machine learning models represent objects using vectors of real numbers. However, there are important differences in how these vectors are used to encode the properties of objects. In this section, we characterize two broad approaches to such property encoding – *featural representations* and *non-featural representations* – and argue that the differences between them have not been given sufficient attention in the debate about the ability of neural networks to perform relational reasoning.

We ground our discussion in a hypothetical universe of blocks which vary by shape and color. Figure 2a is a partial view of them, and Figure 2b–Figure 2e present four different ways of encoding the properties of these objects in vectors.

Featural Representations. The defining characteristic of *featural* vector representations is that each dimension encodes the value of a single property. The properties can be binary, integer-valued, or real-valued.

We use the term *localist* for the special case of featural representations in which only identity properties are represented and there is an identity property corresponding to each object. In Figure 2b, each column represents the property of being an object, and every object is represented as a vector that has a single unit with value 1. There is no shared structure across objects; all are equally (un)related to each other as far as the model is concerned.

A featural representation that is not localist is often called a *distributed representation*. Here, column dimensions encode specific properties of objects. Coming back to our universe of blocks, we can represent the properties of being red and being blue with two different binary features, and the property of having a certain number of sides as a single analog feature, as in Figure 2c. Unlike with localist representations, objects in this space can have complex relationships to each other, as encoded in the shared structure given by the columns.

Featural representations – both localist and distributed – have the appealing property that they are easy for researchers to interpret because of the tight correspondence between column dimensions and properties. However, this transparency actually inhibits neural networks from discovering general solutions; such models work far better with representations that have property values implicitly encoded in the abstract structure of the vector space. We demonstrate this result analytically in Appendix for the case of binary features. The core insight is that networks cannot learn anything about column dimensions that are not represented in their training data; whatever weights are associated with those dimensions are unchanged by the learning process, so predictions about those dimensions remain random at test time. (30–32) overcome this analytic limation of featural representations by modifying standard neural architectures or network weight priors. We simply choose not to use featural representations, and instead opt for non-featural representations which do not have this analytic limation and are the norm in state-of-the-art artificial intelligence models.

Non-Featural Representations. A *non-featural representation* is a vector that encodes property values implicitly across many dimensions. Perhaps the simplest non-featural representations are *completely random* vectors, as in Figure 2d. Random representations can be seen as the non-featural counterpart to localist representations. In both of these representation schemes, all the objects are equally (un)related to each other, since column-wise patterns are unlikely in random representations and, to the extent that they are present, they exist completely by chance. However, in random representations, all the column dimensions can contribute meaningfully to identifying objects, whereas a localist representation has only one vector unit that determines the identity of any given object.

Random representations are a starting point that encodes object identity. To encode more diverse information, we can *pretrain* random representations via a learning process. This will imbue them with rich structure that implicitly encodes property values across many dimensions. Figure 2e provides a simple example. This matrix is the results of pretraining the representations in Figure 2d on the task of predicting whether the object is blue, whether the object is red, and the number of sides the object has. (Appendix provides technical details on our pretraining approach.) Superficially, the two matrices look equally random. However, this is only because our pretraining process leads to property values being implicitly represented by structures in the vector representation space. The random representations in Figure 2f have no such structure, while the pretrained representations in Figure 2g do – e.g., there is a

line that separates blue and red objects.

Pretraining need not be restricted to input representations. All the parameters of a model can be pretrained. In this setting, a small amount of task-specific training (often called "fine-tuning") might suffice. This offers the possibility that networks might be used as modular components to solve more complex tasks. We realize this possibility with our third experiment, where a model pretrained on a simple equality is used as a modular component to compute hierarchical equality.

Model 1: Same–different relations

First, we investigate whether a supervised single layer feed-forward neural network can learn the equality relation in the strict setting we describe above where train and test vocabulary are disjoint. The input is a pair of vectors (a, b) which correspond to the two stimulus objects. These vectors are non-featural representations that do not have features encoding properties of the objects or their identity. During training, this model is presented with positive and negative labeled examples. During testing, this model is tasked with categorizing inputs unseen during training. It is straightforward to show that a network like this is capable of learning equality as we have defined it. Appendix provides an analytic solution to the equality relation using this neural model.

This result shows that equality in our sense is learnable in principle, but it doesn't resolve the question of whether networks can find this kind of solution given finite training data. To address this, we train the network on a stream of pairs of random vectors. Half of these are identity pairs (a, a), labeled with 1, and half are non-identity pairs (a, b), labeled with 0. Trained networks are assessed on the same kind of balanced dataset, with vectors that were never seen in training so that, as discussed earlier, we get a clear picture of whether they have found a generalizable solution.

Results. Figure 3a shows our results. The representations used in these experiments are random representations that were pretrained using a linear classifier for 0, 3, 5, or 10 different binary feature discrimination tasks. For example, following Figure 2, a three-task model might be trained to encode the binary properties of being blue, having four sides, and being red. For all representations, this neural model reaches above-chance performance almost immediately, but requires upwards of 1,000 examples to achieve near perfect accuracy. Interestingly, we see a clear speed-up, with more pretraining tasks resulting in the largest gains; by grounding our representations in "property domains" (as represented by the different task dimensions), we imbue them with implicit structure that makes learning easier.

Discussion. Our assessment pairs have nothing in common with the training pairs except insofar as both involve vectors of real numbers of the same dimensionality. During training, the network is told (via labels) which pairs are equality pairs and which are not, but the pairs themselves contain no information about equality per se. It thus seems fair to us to say that these networks have learned equality, or at least how to simulate that relation with near perfect accuracy. Further, the use of representations that are structured by pretraining results in faster learning.

Model 2: Sequential same–different (ABA task)

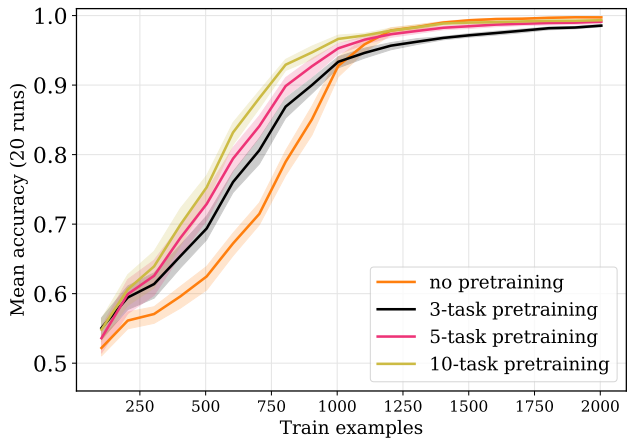
Our first model is simple and successfully learns equality. However, this model is supervised with both positive and negative evidence. In the initial debate around these issues, supervision with negative evidence was dismissed as an unreasonably strong learning regime (e.g., 33). While this argument likely holds true for language learning (in which supervision is generally agreed not to be binary or direct; 34, 35), it is not necessarily true for learning more generally. Nevertheless, learning of sequential rules without negative feedback is possible for infants (9, 36). In experiments of this type, infants are presented with a set of positive examples. Our next model explores whether neural network models can learn this task in a challenging regime with no negative supervision.

To explore learning with only positive instances, we use a neural LSTM language model, a recursive network with the ability to selectively forget and remember information(37). Language models are sequential: at each timestep, they predict an output given their predictions about the preceding timesteps. As typically formulated, the prediction function is just a classifier: at each timestep, it predicts a probability distribution over the entire vocabulary of options, and the item with the highest probability is chosen as a symbolic output. This output becomes the input at the next timestep, and the process continues.

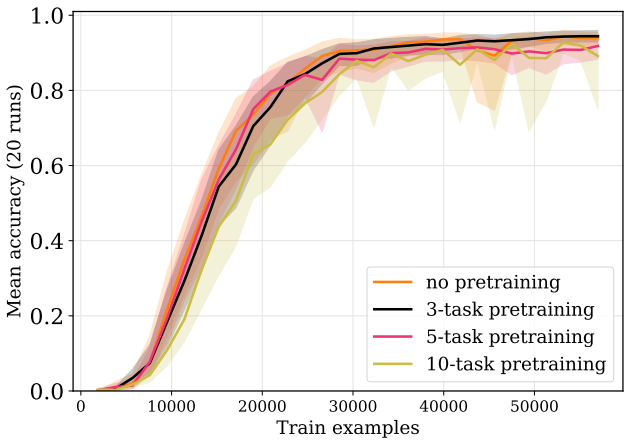
This formulation will not work in situations in which we want to make predictions about test items with an entirely disjoint vocabulary from the training sample. The classifier function will get no feedback about these out-of-vocabulary items during training, and so it will never predict them during testing. To address this issue, we reformulate the prediction function. Our proposal is to have the model predict output vector representations – instead of discrete vocabulary items – at each timestep. During training, the model is trained to minimize the distance between these output predictions and the representations of the actual output entities. During assessment, we take the prediction to be the item in the entire vocabulary (training and assessment) whose representation is closest to the predicted vector (in terms of Euclidean distance). This fuzzy approach to prediction creates enough space for the model to predict sequences from an entirely new vocabulary.

Appendix provides an analytic solution to the ABA task using this model. To see how well the model performs in practice, we trained networks on sequences `<s> a b a </s>`, where `b` $\neq$ `a`. We show the network every such sequence during training, from an underlying vocabulary of 20 items (creating a total of 380 examples). To assess how well the model learns this pattern, we seed it with `<s> x` where `x` is an item from a disjoint vocabulary from that seen in training, and we say that a prediction is accurate if the model continues with `y x </s>`, where `y` is any character (from the training or assessment vocabulary) except `x`.

Results. Figure 3b shows our results. Unlike for the previous equality experiment, we found that we had to allow the model to experience multiple epochs of training on the same set in order to succeed and tens of thousands of training examples are necessary. We consider a range of representations (as in Model 1); the model was again successful with all representations, but in this experiment pretraining representations did not increase performance.



(a) Results for single layer feed-forward neural networks trained on our simple equality task. The 'no pretraining' model is provided random representations and the 'k-task pretraining' models are provided random representations are grounded in k binary property domains via pretraining learning tasks.



(b) Results for LSTM recursive neural networks trained on our sequential equality task. The 'no pretraining' model is provided random representations and the 'k-task pretraining' models are provided random representations are grounded in k binary property domains via pretraining learning tasks. All the training examples are presented at once over multiple epochs

Discussion. These sequential models are given no negative examples and they must predict into a totally new vocabulary. Despite these challenges, they succeed at learning the underlying patterns in our data. On the other hand, the learning process is slow and data-intensive. We hypothesized that grounding representations in property domains via pretraining might lead to noticeable speed-ups, as it did in for our simple same-different task, but we did not see this effect in practice. We speculate that there may be model variants that reduce these demands, given that learning is in principle possible in this architecture, but we leave them to future work.

Model 3: Hierarchical same–different relations

Given the strong results found for simple equality relations, we can ask whether more challenging equality problems are also learnable in our setting. The hierarchical equality task used by (2) is an interesting test case: given a pairs of pairs $((a, b), (c, d))$, the label is 1 if $(a = b) = (c = d)$, else 0. (2) suggested that the ability exemplified by this task – reasoning about hierarchical *same* and *different* relations – could represent a form of symbolic abstraction uniquely enabled by language. Given the non-symbolic nature of our models, our simulations provide a test of this hypothesis, though we should look critically at their ability to find good solutions with reasonable amounts of training data.

We can approach this task using the same model and methods as we used for equality, with the insubstantial change of providing the network four vector representations instead of two. We found that single layered feed-forward neural networks required nearly 100,000 training examples to solve this task. We hypothesized that the flat input representations $[a; b; c; d]$ might be suboptimal here. This task is intuitively hierarchical: if one works out the equality labels for each of the two pairs, then the further classification decision can be done entirely on that basis. Our current neural network might be too shallow to find this kind of decomposition. To address this, we use a two layer feed forward network.

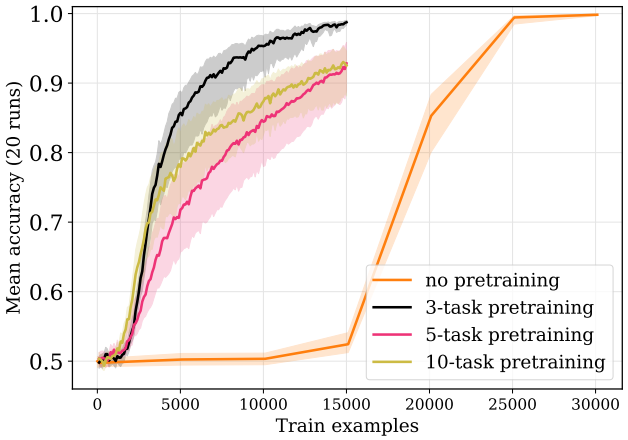
Results. Figure 4a shows our results. We again consider a range of representations and again the network succeeds across this range, with pretraining increasing performance as in Model 1. The network required more than 20,000 training instances to reach top performance and upwards of 10,000 examples with pretrained representations.

Discussion. This amount of training data is still vastly more data than human participants get in similar experiments, which typically involve short exposures in the range of dozens to hundreds of examples (e.g., 9, 38). Thus, it is worth asking whether there are other solutions that would be more data efficient and more in line with human capabilities. We next sought to capitalize even more on the hierarchical nature of this task by defining a modular pretraining regime in which previously learned capabilities are recruited for new tasks.

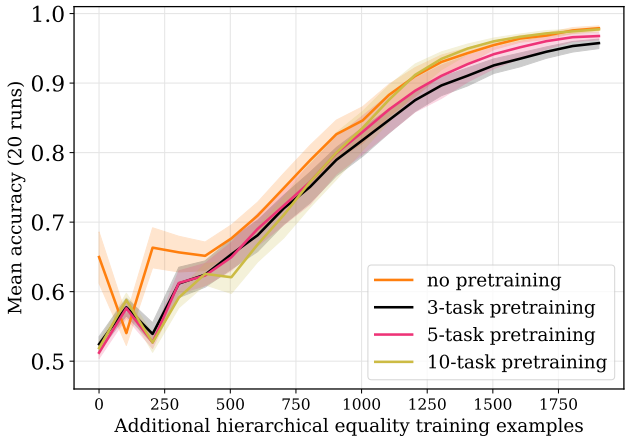
The critical role of experience. Our successful results training neural networks on simple equality suggested another strategy for solving the hierarchical equality task. Rather than requiring our networks to find solutions from scratch, we pretrained them on basic equality tasks and then used those parameters as a starting point for learning hierarchical equality. This set of simulations was conceptually similar to our previous experiments with pretrained input representations, but now we pretrained an entire subpart of the model, rather than just input representations.

The hierarchical equality task requires computing the equality relation three times: compute whether each pair of inputs are equal and then compute whether the truth-valued outputs of these first two computations are equal. We propose to use the same network pretrained on basic equality to perform all three equality computations.

Results and discussion. Figure 4b shows our results. All the models have above chance performance after being trained only on the simple equality task – that is, they achieve zero-shot generalization to the hierarchical task and after just a couple thousand examples, the models achieve with near perfect accuracy. It is remarkable that a model trained only on



(a) Results for two layer feed-forward neural networks trained on our hierarchical equality task. The 'no pretraining' model is provided random representations and the 'k-task pretraining' models are provided random representations are grounded in k binary property domains via pretraining learning tasks.



(b) Results for simple equality networks applied to the hierarchical equality task. The 'no pretraining' model is provided random representations and the 'k-task pretraining' models are provided random representations are grounded in k binary property domains via pretraining learning tasks.

equality between entities is able to get traction on a problem that requires determining whether equality holds between the truth values encoded in two learned representations. Pretrained representations did not increase performance.

General Discussion

Equality is a key case study for understanding the origins of human relational reasoning. This case study has been puzzling for symbolic accounts of reasoning because such accounts do not provide a compelling explanation for why some equality tasks are so easy to learn and others are so hard. In addition, evidence of graded learning and generalization in non-human species suggests that a gradual learning account might provide more traction in explaining the empirical data (12). Inspired by this work, we revisited a long-standing debate about whether neural network models can learn equality relations from data (23). We presented a series of such models that succeeded even in stringent assessments, and, in the case of neural language models, even without being shown negative examples.

In some settings, our current models require more training instances than humans seem to need. However, our pretraining approach suggests a path forward: by using pretrained models as modular components, we can get traction on challenging tasks without any training specifically for those tasks. In some cases, even a small amount of additional training can make a substantial difference. Perhaps modular, pretrained components could serve as the basis for more complex cognitive abilities more generally.

One implication of these pretraining findings is that it should be possible to scaffold performance in complex, hierarchical equality tasks via training on simpler ones. Indeed, (14) show just this result in crows, consistent with our findings. Although we do not discount the potential role of linguistic labels in informing adult humans' expertise in such tasks (39), pretraining also provides a potential account of how infants and young children might succeed in a range of equality reasoning tasks without access to specific linguistic symbols like "same" (4, 8, 40).

More broadly still, our work suggests a possible way forward in understanding the acquisition of logical semantics. Graded logical functions like those our models learned here could form the foundation for a semantics of words like "same" (41). Such an option is appealing because it escapes from the circularity of defining the semantics of linguistic symbols as originating in a mental primitive SAME. A semantics for "same" requires defining its inputs and outputs as well as how it composes with other symbols. The assertion that there is a primitive identity computation does not provide these; it further fails to explain the flexibility that allows us to call two Toyota Corollas "the same" but two twin sisters "different." In contrast, the kinds of networks we propose here could in principle be conditioned contextually to provide flexible, context-sensitive interpretation of logical meaning.

Earlier debates about the nature of equality computations centered around the question of whether models included symbolic elements. We believe ours do not; but it is of course possible to quibble with this judgment. For example, since the supervisory signal used in Models 1 and 3 is generated based on a symbolic rule, perhaps that makes these models symbolic under some definition. (Of course, the same argument could be applied to the supervision signal that is provided to crows, baboons, and human children). We view this kind of argument as terminological, rather than substantive. In the end, our goal is an explicit learning theory for relational reasoning. Our hope is that the work described here takes a first step in this direction.

Materials and Methods

The code and data used to run the experiments in this paper are publicly available at: https://github.com/atticusg/NeuralRelationalReasoning

The simple equality model takes in input vectors, a, b, and uses a single linear transformation followed by a non-linearity to create a hidden representation h that is then used to create a probability distribution, y, over the two classes.

$$h = \text{ReLU}([a; b]W_{xh} + b_h) \quad [1]$$

$$y = \textbf{softmax}(hW_{hy} + b_y) \quad [2]$$

The sequential equality model takes in a sequence of input vectors, $x_1, x_2, \ldots$, and uses an LSTM cell to create a hidden representation h_t at each timestep t that is that is then linearly

projected into the input vector space providing a prediction for that timestep, y_t.

$$h_t = \mathbf{LSTM}(x_t, h_{t-1}) \quad [3]$$
$$y_t = h_t W + b \quad [4]$$

The first hierarchical equality model takes in input vectors, a, b, c, d, and applies a linear transformation followed by a non-linearity to create a hidden representation h_1 and then applies these two steps once more to create second hidden representation h_2 that is then used to create a probability distribution, y, over the two classes.

$$h_1 = \mathrm{ReLU}([a; b; c; d]W_{xh} + b_{h_1}) \quad [5]$$
$$h_2 = \mathrm{ReLU}(h_1 W_{hh} + b_{h_2}) \quad [6]$$
$$y = \mathbf{softmax}(h_2 W_{hy} + b_y) \quad [7]$$

The second hierarchical equality model takes in input vectors, a, b, c, d, and applies the simple equality model from equations (1)–(2) to the pairs (a, b) and (c, d) to produce hidden representations h_1 and h_2. Then the simple equality model is applied once again to the pair (h_1, h_2) to produce a final hidden representation h_3 that is used to create a probability distribution, y, over the two classes.

$$h_1 = \mathrm{ReLU}([a; b]W_{xh} + b_h) \quad [8]$$
$$h_2 = \mathrm{ReLU}([c; d]W_{xh} + b_h) \quad [9]$$
$$h_3 = \mathrm{ReLU}([h_1; h_2]W_{xh} + b_h) \quad [10]$$
$$y = \mathbf{softmax}(h_3 W_{hy} + b_y) \quad [11]$$

The parameters for the simple and hierarchical equality models are learned using back propagation with a cross entropy objective function defined as follows, for a set of N examples and K classes:

$$\max(\theta) \quad \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y^{i,k}\log(h_\theta(i)^k) \quad [12]$$

where θ abbreviates the model parameters, $y^{i,k}$ is the actual label for example i and class k, and $h_\theta(i)^k$ is the corresponding prediction.

The parameters for the sequential model are learned using back propagation with a squared mean error objective function defined as follows:

$$\max(\theta) \quad -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T_i}\left\|h_\theta\left(x^{i,0:t-1}\right) - x^{i,t}\right\|^2 \quad [13]$$

for N examples. Here, T_i is the length of example i. As before, θ abbreviates the parameters of the model as specified in (19)–(20). We use $h_\theta(x^{i,0:t-1})$ for the vector predicted by the model for example i at timestep t, which is compared to the actual vector at timestep t via squared Euclidean distance.

Hyperparameter searches and implementation details can be found in Appendix .

ACKNOWLEDGMENTS. This work is supported in part by a Facebook Robust Deep Learning for Natural Language Processing Research Award and by the McDonnell Foundation grant "The Nature and Origins of the Human Capacity for Abstract Combinatorial Thought".

1. Gentner D, Goldin-Meadow S (2003) *Language in mind: Advances in the study of language and thought.* (MIT press).
2. Premack D (1983) The codes of man and beasts. *Behavioral and Brain Sciences* 6(1):125–136.
3. KR Thompson R, Rattermann MJ, L Oden D (2001) Perception and judgement of abstract same-different relations by monkeys, apes and children: Do symbols make explicit only that which is implicit? *Hrvatska revija za rehabilitacijska istraživanja* 37(1):9–22.
4. Walker CM, Bridgers S, Gopnik A (2016) The early emergence and puzzling decline of relational reasoning: Effects of knowledge and search on inferring abstract concepts. *Cognition* 156:30–40.
5. Giurfa M, Zhang S, Jenett A, Menzel R, Srinivasan MV (2001) The concepts of 'sameness' and 'difference' in an insect. *Nature* 410(6831):930–933.
6. Avarguès-Weber A, Deisig N, Giurfa M (2011) Visual cognition in social insects. *Annual review of entomology* 56:423–443.
7. Anderson EM, Chang YJ, Hespos S, Gentner D (2018) Comparison within pairs promotes analogical abstraction in three-month-olds. *Cognition* 176:74–86.
8. Ferry AL, Hespos SJ, Gentner D (2015) Prelinguistic relational concepts: Investigating analogical processing in infants. *Child Development* 86(5):1386–1405.
9. Marcus GF, Vijayan S, Rao SB, Vishton PM (1999) Rule learning by seven-month-old infants. *Science* 283(5398):77–80.
10. Gervain J, Berent I, Werker JF (2012) Binding at birth: The newborn brain detects identity relations and sequential position in speech. *Journal of Cognitive Neuroscience* 24(3):564–574.
11. Frank MC, Tenenbaum JB (2011) Three ideal observer models for rule learning in simple languages. *Cognition* 120(3):360–371.
12. Wasserman E, Castro L, Fagot J (2017) Relational thinking in animals and humans: From percepts to concepts. in *APA HAndbook of Comparative Psychology: Perception, Learning, and Cognition*, eds. Call J, Burghardt GM, Pepperberg IM, Snowdon CT, Zentall T. (American Psychological Association) Vol. 2.
13. Cook RG, Wasserman EA (2007) Learning and transfer of relational matching-to-sample by pigeons. *Psychonomic Bulletin & Review* 14(6):1107–1114.
14. Smirnova A, Zorina Z, Obozova T, Wasserman E (2015) Crows spontaneously exhibit analogical reasoning. *Current Biology* 25(2):256–260.
15. Fagot J, Thompson RK (2011) Generalized relational matching by guinea baboons (papio papio) in two-by-two-item analogy problems. *Psychological Science* 22(10):1304–1309.
16. Castro L, Kennedy PL, Wasserman EA (2010) Conditional same-different discrimination by pigeons: Acquisition and generalization to novel and few-item displays. *Journal of Experimental Psychology: Animal Behavior Processes* 36(1):23.
17. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. *nature* 521(7553):436–444.
18. Dienes Z, Altmann GT, Gao SJ (1999) Mapping across domains without feedback: A neural network model of transfer of implicit knowledge. *Cognitive Science* 23(1):53–82.
19. Seidenberg MS, Elman JL (1999) Networks are not 'hidden rules'. *Trends in Cognitive Sciences* 3(8):288–289.
20. Seidenberg MS, Elman JL (1999) Do infants learn grammar with algebra or statistics? *Science* 284(5413):433–433.
21. Elman JL (1999) Generalization, rules, and neural networks: A simulation of Marcus et. al. HTML document.
22. Negishi M (1999) Do infants learn grammar with algebra or statistics? *Science* 284(5413):435.
23. Alhama RG, Zuidema W (2019) A review of computational models of basic rule learning: The neural-symbolic debate and beyond. *Psychonomic bulletin & review* 26(4):1174–1194.
24. Collobert R, et al. (2011) Natural language processing (almost) from scratch. *Journal of Machine Learning Research* 12:2493–2537.
25. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality in *Advances in Neural Information Processing Systems 26*, eds. Burges CJC, Bottou L, Welling M, Ghahramani Z, Weinberger KQ. (Curran Associates, Inc.), pp. 3111–3119.
26. Pennington J, Socher R, Manning CD (2014) GloVe: Global vectors for word representation in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. (Association for Computational Linguistics, Doha, Qatar), pp. 1532–1543.
27. Peters M, et al. (2018) Deep contextualized word representations in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. (Association for Computational Linguistics), pp. 2227–2237.
28. Devlin J, Chang MW, Lee K, Toutanova K (2019) BERT: Pre-training of deep bidirectional transformers for language understanding in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. (Association for Computational Linguistics, Minneapolis, Minnesota), pp. 4171–4186.
29. Marcus GF (2001) *The Algebraic Mind: Integrating Connectionism and Cognitive science.* (MIT Press).
30. Weyde T, Kopparti R (2019) Modelling identity rules with neural networks. Vol. 6, pp. 745–769.
31. Weyde T, Kopparti R (2018) Feed-forward neural networks need inductive bias to learn equality relations in *Proceedings of the NeurIPS 2018 Relation Representation Learning Workshop*.
32. Kopparti R, Weyde T (2020) Weight priors for learning identity relations.
33. Marcus GF (1999) Rule learning by seven-month-old infants and neural networks. Response to Altmann and Dienes. *Science* 284:875.
34. Brown R, Hanlon C (1970) Derivational complexity and order of development in speech in *Cognition and the development of language.*, ed. Hayes JR. (Wiley).
35. Chouinard MM, Clark EV (2003) Adult reformulations of child errors as negative evidence. *Journal of child language* 30(3):637–669.
36. Rabagliati H, Ferguson B, Lew-Williams C (2019) The profile of abstract rule learning in infancy: Meta-analytic and experimental evidence. *Developmental science* 22(1):e12704.
37. Hochreiter S, Schmidhuber J (1997) Long short-term memory. *Neural computation* 9(8):1735–1780.
38. Endress AD, Scholl BJ, Mehler J (2005) The role of salience in the extraction of algebraic rules. *Journal of Experimental Psychology: General* 134(3):406.
39. Gentner D (2003) Why we're so smart. *Language in mind: Advances in the study of language and thought* 195235.
40. Hochmann JR, Mody S, Carey S (2016) Infants' representations of same and different in match-and non-match-to-sample. *Cognitive psychology* 86:87–111.
41. Potts C (2019) A case for deep learning in semantics: Response to pater. *Language*.
42. Cybenko G (1989) Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* 2(4):303–314.
43. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. *Neural Networks* 2(5):359–366.
44. Kingma DP, Ba J (2014) Adam: A method for stochastic optimization in *Proceedings of the 3rd International Conference on Learning Representations*.

Model 1 Supplementary Methods

The input to our model is a pair of vectors (a, b), each of dimension m, which correspond to the two stimulus objects. These vectors are non-featural representations that do not have features encoding properties of the objects or their identity. These are concatenated to form a single vector $[a; b]$ of dimension $2m$, which is the simplest way of merging the two representations to form a single input.

This representation is multiplied by a matrix of weights W_{xh} of dimension $2m \times n$ and a bias vector b_h of dimension n is added to this result, where n is the hidden layer dimensionality. These two steps create a linear projection of the input representation, and the bias term is the value of this linear projection when the input representation is the zero vector. Then, the non-linear activation function ReLU ($\text{ReLU}(x) = \max(0, x)$) is applied element-wise to this linear projection. This non-linearity is what gives the neural model more expressive power than a logistic regression (42, 43). The result is the hidden representation h.

The hidden representation is the input to the classification layer: h is multiplied by a second matrix of weights W_{hy}, dimension $n \times 2$, and a bias term b_y (dimension 2) is added to this. This second bias term encodes the probabilities of each class when the hidden representation is 0. The result is fed through the softmax activation function: $\textbf{softmax}(x)_i = \frac{\exp x_i}{\sum_j \exp x_j}$. This creates a probability distribution over the classes (positive and negative). For a given input, the model computes this probability distribution and the input is categorized as the class with the higher probability.

During training, this model is presented with positive and negative labeled examples and the parameters W_{xh}, W_{hy}, b_y, and b_h are learned using back propagation with a cross entropy function. This function is defined as follows, for a set of N examples and K classes:

$$\max(\theta) \quad \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y^{i,k} \log(h_\theta(i)^k) \qquad [14]$$

where θ abbreviates the model parameters (W_{xh}, W_{hy}, b_y, b_h), $y^{i,k}$ is the actual label for example i and class k, and $h_\theta(i)^k$ is the corresponding prediction.

Model 2 Supplementary Methods

The input to this model is a sequence of vectors $x_1, x_2, x_3, \ldots$, each of dimension m, which correspond to a sequence of stimulus objects. These vectors are, again, non-featural representations that do not have features encoding properties of the objects or their identity.

At each timestep t, the input vector x_t is fed into the **LSTM** cell along with the previous hidden representation h_{t-1}. The defining feature of an **LSTM** is the ability to decide whether to store information from the current input, x_t, and whether to remember or forget the information from the previous timestep h_{t-t}. The output of the **LSTM** cell is the hidden representation for the current time step h_t. The dimension of the hidden representations is n. The hidden representation is multiplied by a matrix W with dimensionality $n \times m$ to produce y_t. This result, y_t, is a linear projection of the hidden representation into the input vector space, which is necessary because y_t is a prediction of what the next input, x_{t+1}, will be.

The objective function is as follows:

$$\max(\theta) \quad -\frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T_i} \left\| h_\theta\left(x^{i,0:t-1}\right) - x^{i,t} \right\|^2 \qquad [15]$$

for N examples. Here, T_i is the length of example i. As before, θ abbreviates the parameters of the model as specified in (3)–(4). We use $h_\theta(x^{i,0:t-1})$ for the vector predicted by the model for example i at timestep t, which is compared to the actual vector at timestep t via squared Euclidean distance (i.e., the mean squared error).

Model figures

Model 1: Same–different relation with feed-foward networks. Our model of equality is given by (16)–(17):

$$h = \text{ReLU}([a; b]W_{xh} + b_h) \qquad [16]$$

$$y = \textbf{softmax}(hW_{hy} + b_y) \qquad [17]$$

The input is a pair of vectors (a, b), each of dimension m, which correspond to two stimulus objects. These vectors are non-featural representations that do not hav efeatures encoding properties of the objects or their identities. These are concatenated to form a single vector $[a; b]$ of dimension $2m$, which is the simplest way of merging the two representations to form a single input.

This representation is multiplied by a matrix of weights W_{xh} of dimension $2m \times n$ and a bias vector b_h of dimension n is added to this result, where n is the hidden layer dimensionality. These two steps create a linear projection of the input representation, and the bias term is the value of this linear projection when the input representation is the zero vector. Then, the non-linear activation function ReLU ($\text{ReLU}(x) = \max(0, x)$) is applied element-wise to this linear projection. This non-linearity is what gives the neural model more expressive power than a logistic regression (42, 43). The result is the hidden representation h.

The hidden representation is the input to the classification layer: h is multiplied by a second matrix of weights W_{hy}, dimension $n \times 2$, and a bias term b_y (dimension 2) is added to this. This second bias term encodes the probabilities of each class when the hidden representation is 0. The result is fed through the softmax activation function: $\textbf{softmax}(x)_i = \frac{\exp x_i}{\sum_j \exp x_j}$. This creates a probability distribution over the classes (positive and negative). For a given input, the model computes this probability distribution and the input is categorized as the class with the higher probability.

The parameters W_{xh}, W_{hy}, b_y, and b_h are learned using back propagation with a cross entropy function. This function is defined as follows, for a set of N examples and K classes:

$$\max(\theta) \quad \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y^{i,k} \log(h_\theta(i)^k) \qquad [18]$$

where θ abbreviates the model parameters (W_{xh}, W_{hy}, b_y, b_h), $y^{i,k}$ is the actual label for example i and class k, and $h_\theta(i)^k$ is the corresponding prediction.

Figure 5 provides a visual depiction of the model. The gray boxes correspond to embedding representations, the purple box is the hidden representation h, and the red box is the output distribution y. Dotted arrows depict concatenation, and solid arrows depict the dense relations corresponding to the matrix multiplications (plus bias terms) in (1)–(2).

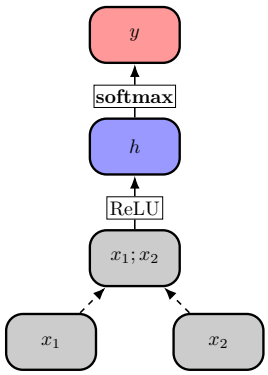


Fig. 5. A single layer network computing equality.

Model 2: Sequential same–different (ABA task). The specific model we use for this is as follows:

$$h_t = \textbf{LSTM}(x_t, h_{t-1}) \qquad [19]$$

$$y_t = h_t W + b \qquad [20]$$

This holds for $t > 0$, and we set $h_0 = \mathbf{0}$. **LSTM** is a long short-term memory cell (37). Full details on these cells are given in Appendix .

The input is a sequence of vectors $x_1, x_2, x_3, \ldots$, each of dimension m, which correspond to a sequence of stimulus objects. These vectors are, again, non-featural representations that do not have features encoding properties of the objects or their identity.

At each timestep t, the input vector x_t is fed into the **LSTM** cell along with the previous hidden representation h_{t-1}. The defining feature of an **LSTM** is the ability to decide whether to store information from the current input, x_t, and whether to remember or forget the information from the previous timestep h_{t-t}. The output of the **LSTM** cell is the hidden representation for the current time step h_t. The dimension of the hidden representations is n. The hidden representation is multiplied by a matrix W with dimensionality $n \times m$ to produce y_t. This result, y_t, is a linear projection of the hidden representation into the input vector space, which is necessary because y_t is a prediction of what the next input, x_{t+1}, will be.

The objective function is as follows:

$$\max(\theta) \quad -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T_i}\left\|h_\theta\left(x^{i,0:t-1}\right) - x^{i,t}\right\|^2 \qquad [21]$$

for N examples. Here, T_i is the length of example i. As before, θ abbreviates the parameters of the model as specified in (19)–(20). We use $h_\theta(x^{i,0:t-1})$ for the vector predicted by the model for example i at timestep t, which is compared to the actual vector at timestep t via squared Euclidean distance (i.e., the mean squared error).

Figure 6 depicts this model. At each timestep t, a vector y_t (red) is predicted based on the input representation at t (gray) and the hidden representation at t (purple). During training, this is compared with the actual vector for timestep $t+1$ (green). During testing, the predicted vector y_i is compared with every item in the union of the train and assessment vocabularies, and the closest vector (according to Euclidean distance) is taken to be the prediction. This vector is then used as the input for timestep $t+1$.

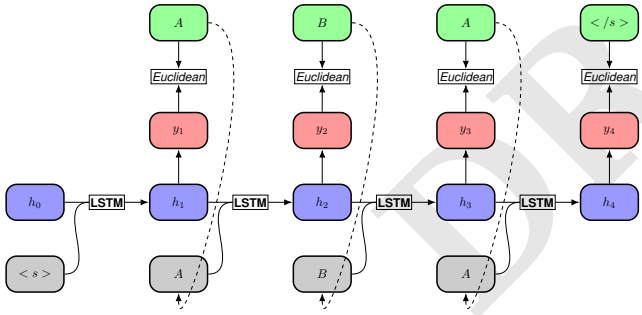


Fig. 6. A recursive LSTM network producing ABA sequences.

Model 3a: A single layer feed forward network . The only change required to equations (16)–(17) is that we create inputs $[a; b; c; d]$: the flat concatenation of all the elements of the two pair of vectors. This change in turn leads W_{xh} to have dimensionality $4m \times n$. The objective function is again defined using a cross entropy function, as in equation 18. Appendix provides a full picture of these learning trends. These models are able to find nearly perfect solutions, but vastly more training data is required for this task than was required for simple equality, and the network configuration matters much more. For example, our model with 10-dimensional entity representations and 100-dimensional hidden representations reached near perfect accuracy, but only with over 95,000 training instances. A comparable model with 50-dimensional entity representations failed to get traction at all with this amount of training data, and pretraining led to only minor improvements. For this reason, we also perform experiments with a deeper neural network.

Model 3b: A deeper feed-forward network for hierarchical same–different. This model extends (16)–(17) with an additional hidden layer and a larger input dimensionality, corresponding to the input pair of pairs $((a, b), (c, d))$ being flattened into a single concatenated representation $[a; b; c; d]$:

$$h_1 = \text{ReLU}([a; b; c; d]W_{xh} + b_{h_1}) \qquad [22]$$
$$h_2 = \text{ReLU}(h_1 W_{hh} + b_{h_2}) \qquad [23]$$
$$y = \textbf{softmax}(h_2 W_{hy} + b_y) \qquad [24]$$

The objective function is again defined using a cross entropy function, as in equation 18.

Figure 7 depicts this model.

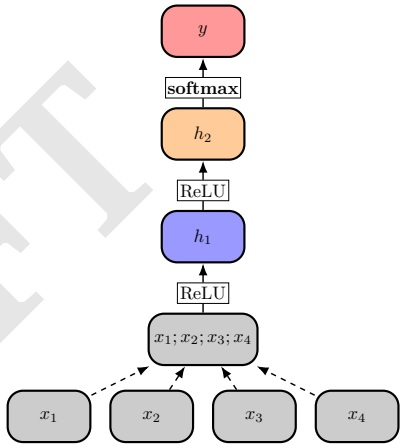


Fig. 7. A two layer network computing hierarchical equality.

Model 3c: Pretraining for hierarchical same–different. Our pretraining model is as follows:

$$h_1 = \text{ReLU}([a; b]W_{xh} + b_h) \qquad [25]$$
$$h_2 = \text{ReLU}([c; d]W_{xh} + b_h) \qquad [26]$$
$$h_3 = \text{ReLU}([h_1; h_2]W_{xh} + b_h) \qquad [27]$$
$$y = \textbf{softmax}(h_3 W_{hy} + b_y) \qquad [28]$$

where W_{xh}, W_{hy}, b_h, and b_y are the parameters from the model in equations (16)–(17) already trained on basic equality. Crucially, the same parameters, W_{xh} and b_h, are used three times: twice to compute representations encoding whether a pair of input entities are equal (h_1, h_2), and once to compute a representation (h_3) encoding whether the truth values encoded by h_1 and h_2 are equal. This final representation is then used to compute a probability distribution over two classes, and the class with the higher probability is predicted by the model.

The objective function is again defined using a cross entropy function, as in equation 18.

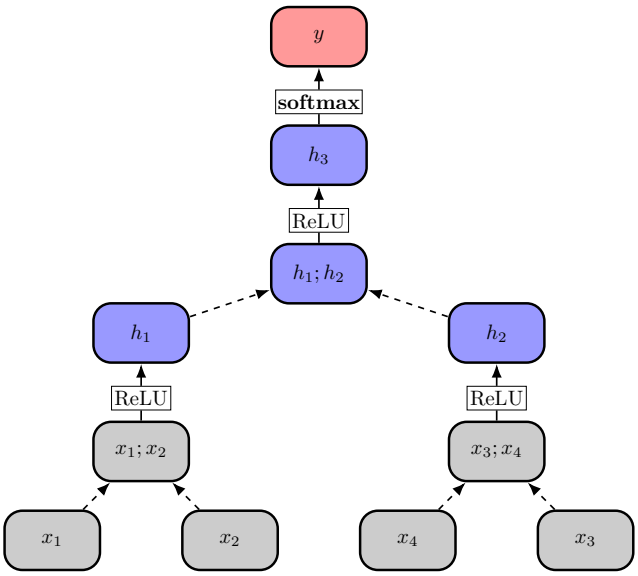


Fig. 8. A single layer network pretrained on equality computing hierarchical equality.

Pretraining

Our pretraining model closely resembles the models used for our main experiments. It defines a feed-forward network with a multi-task objective. For an example i and task j:

$$h_i = \text{ReLU}\left(E[i]W_{xh} + b_h\right) \quad [29]$$

$$y_{i,j} = \textbf{softmax}(h^i W^j_{hy} + b^j_y) \quad [30]$$

where $E[i]$ is the vector representation for example i in the embedding matrix E. The overall objective of the model is to maximize the sum of the task objective functions. For N examples, J tasks, and K_j the number of classes for task j:

$$\max(\theta) \quad \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{J}\sum_{k=1}^{K_j} y^{i,j,k}\log\left(h_\theta(i)^{j,k}\right) \quad [31]$$

where $y^{i,j,k}$ is the correct label for example i in task j and $p^{i,j,k}$ is the predicted value for example i in task j.

For the experiments in the paper, we initialize E randomly and then, for pretraining on J tasks, we create a random binary vector of length J for each row in E. Each dimension (task) in J is independent of the others.

Our motivation for pretraining is to update the embedding E so that its representations contain rich structure that can be used by subsequent models. To achieve this, we backpropagate errors through the network and into E.

In our experiments, we always pretrain for 10 iterations. This choice is motivated primarily by computational costs; additional pretraining iterations greatly increase experiment run-times, though they do have the potential to imbue the representations with even more useful structure.

Model optimization details

The feed forward networks for basic and hierarchical equality were implemented using the multi-layer perception from sklearn and a cross entropy function was used to compute the prediction error. The recursive LSTM network for the sequential ABA task was implemented using PyTorch and a mean squared error function was used to compute the prediction error. The networks for pretraining representations and for the hierarchical equality task were also implemented using PyTorch, with cross-entropy loss functions used to compute the prediction errors. For all models, Adam optimizers (44) were used. For all models, a hyperparameter search was run over learning rate values of {0.00001, 0.0001, 0.001} and l2 normalization values of {0.0001, 0.001, 0.01} for each hidden dimension and input dimension mentioned in the paper.

Localist and binary feature representations prevent generalization

The method of representation impacts whether there is a natural notion of similarity between entities and the ability of models to generalize to examples unseen in training. These two attributes are deeply related; if there is a natural notion of similarity between vector representations, then models can generalize to inputs with representations that are similar to those seen in training.

In order to discuss how representation impacts generalization, we will need explain some properties of how neural models are trained. Standard neural models, including all models in the paper, begin with applying a linear layer to the input vector where no two input units are connected to the same weight. An easily observed fact about the back-propagation learning algorithm is that, if a unit of the input vector is always zero during training, then any weights connected to that unit and only that unit will not change from their initialized values during training. This means that, when a standard neural model is evaluated on an input vector that has a non-zero value for a unit that was zero throughout training, untrained weights are used and behavior is unpredictable.

Localist representations are orthogonal and equidistant from one another so there is no notion of similarity and consequently standard neural models have no ability to generalize to new examples. No two representations share a non-zero unit, and so when models are presented with inputs unseen in training, untrained weights are used and the resulting behavior is again unpredictable.

Distributed representations with binary features also limit generalization, though less severely than localist representations. Localist representations prevent generalization to entities unseen during training, while binary feature representations prevent generalization to features unseen during training. For example, if color and shape are represented as binary features, and a red square and blue circle are seen in training, then a model could generalize to the unseen entities of a blue circle or a red square. However, if no entity that is a circle is seen during training, then the binary feature representing the property of being a circle is zero throughout training and untrained weights are used when the model is presented with a entity that is a circle during testing, which results in unpredictable behavior.

Distributed representations with analog features do not inhibit generalization in the same way. If height is represented as a binary feature, then a single unit represents all height values and is always non-zero. Non-featural representations similarly do not inhibit generalization, because all units for all representations are non-zero and the network can learn parameters that create complex associations between these entities and its task labels.

An analytic solution to identity with a feed forward network

We now show that our feed forward networks can solve the same–different problem we pose, in the following sense: for any set of inputs, we can find parameters θ that perfectly classify those inputs. At the same time, we also show that there are always additional inputs for which θ makes incorrect predictions.

Here are the parameters of a feed forward neural network that performs a binary classification task

$$\texttt{ReLu}(\begin{pmatrix}x_1\\x_2\end{pmatrix}^T\begin{pmatrix}W^{11} & W^{12}\\W^{21} & W^{22}\end{pmatrix})\begin{pmatrix}v^{11} & v^{12}\\v^{21} & v^{22}\end{pmatrix}+\begin{pmatrix}b_1 & b_2\end{pmatrix}=\begin{pmatrix}o_1 & o_2\end{pmatrix}$$

where, if n is the dimension of entity embeddings used, then

$$x, y, v^{11}, v^{12}, v^{21}, v^{22} \in \mathbb{R}^{n\times 1}$$

$$W^{11}, W^{12}, W^{21}, W^{22} \in \mathbb{R}^{n\times n}$$

$$b_1, b_2, o_1, o_2 \in \mathbb{R}$$

Given an input (x_1, x_2), if the output o_1 is larger than o_2, then one class is predicted; if the output o_2 is larger that o_1, then the other class is predicted. When the two outputs are equal, the network has predicted that both classes are equally likely and we can arbitrarily decide which class is predicted. In this case, the output o_1 predicts the two inputs, x_1 and x_2, are in the identity relation and the output o_2 predicts the two inputs are not. Now we specify parameters

to provide an analytic solution to the identity relation using this network

$$\texttt{ReLu}(\begin{pmatrix}x_1\\x_2\end{pmatrix}^T \begin{pmatrix}I & -I\\-I & I\end{pmatrix}) \begin{pmatrix}\vec{1} & \vec{0}\\\vec{1} & \vec{0}\end{pmatrix} + \begin{pmatrix}b_1 & b_2\end{pmatrix} = \begin{pmatrix}o_1 & o_2\end{pmatrix}$$

where I is the identity matrix, $-I$ is the negative identity matrix, and $\vec{1}$ and $\vec{0}$ are the two vectors in $\mathbb{R}^n$ that have all zeros and all ones, respectively. The output values, given an input, are

$$o_1 = \sum_{i=1}^{n} |(x_1)_i - (x_2)_i| + b_1 \qquad o_2 = b_2$$

where two parameters are left unspecified, b_1, b_2. We present a visualization in Figure 9 of how the analytic solution to identity of this network changes depending on the values of two bias terms. In this example, the network receives two one-dimensional inputs, x_1 and x_2. If the ordered pair of inputs is in the shaded area on the graph, then they are predicted to be in the identity relation. If in the unshaded area, they are predicted not to be. The dotted line is where the network predicts the two classes to be equally likely.

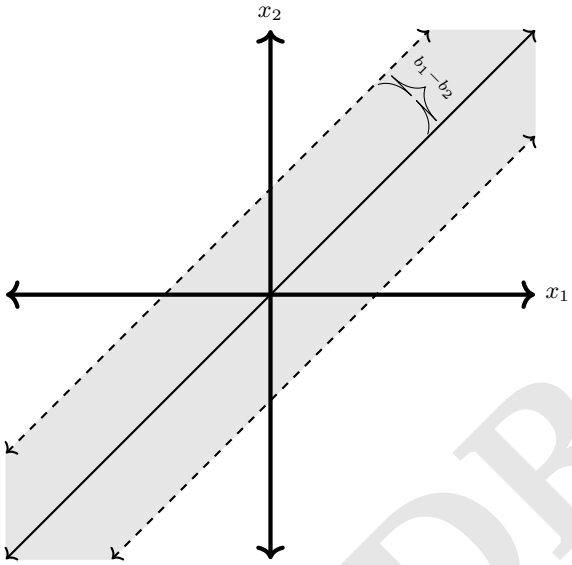


Fig. 9. A visual representation of how the analytic solution to identity of a single layer feed forward network changes depending on the values of two bias terms, b_1, b_2.

The network predicts x_1 and x_2 to be in the identity relation if $\sum_{i=1}^{n} |x_i - y_i| < b_1 - b_2$ which is visualized as the points between two parallel lines above and below the solution line $x_1 = x_2$. As the difference $b_1 - b_2$ gets smaller and smaller, the two lines that bound the network's predictions get closer and closer to the solution line. However, as long as $b_1 - b_2$ is positive, there will always be inputs of the form $(r, r + (b_1 - b_2)/2)$ that are false positives. For any set of inputs, we can find bias values that result in the network correctly classifying those inputs, but for any bias values, we can find an input that is incorrectly classified by those values. In other words, we have an arbitrarily good solution that is never perfect. We provide a proof below that there is no perfect solution and so this is the best outcome possible. However, if we were to decide that, if the network predicts that an input is equally likely in either class, then this input is predicted to be in the identity relation, we could have a perfect solution with $b_1 = b_2$.

Here is proof that a perfect solution is not possible. A basic fact from topology is that the set $\{x : f(x) < g(x)\}$ is an open set if f and g are continuous functions. Let N_{o_1} and N_{o_2} be the functions that map an input (x_1, x_2) to the output values of the neural network, o_1 and o_2, respectively. These functions are continuous. Consequently, the set $C = \{(x_1, x_2) : N_{o_2}(x_1, x_2) < N_{o_1}(x_1, x_2)\}$, which is the set of inputs that are predicted to be in the equality relation, is open.

With this fact, we can show that, if the neural network correctly classifies any point on the solution line $x_1 = x_2$, then it must incorrectly classify some point not on the solution line. Suppose that C contains some point (x, x). Then, by the definition of an open set, C contains some ϵ ball around (x, x), and therefore C contains $(x, x + \epsilon)$, which is not on the solution line $x_1 = x_2$. Thus, C can never be equal to the set $\{(x_1, x_2) : x_1 = x_2\}$. So, because C is the set of inputs classified as being in the equality relation by the neural network, a perfect solution cannot be achieved. Thus, we can conclude our arbitrarily good solution is the best we can do.

An analytic solution to ABA sequences

Here are the parameters of a long short term memory recursive neural network (LSTM):

$$\begin{aligned}
f_t &= \sigma(x_t W_f + h_{t-1} U_f + b_f)\\
i_t &= \sigma(x_t W_i + h_{t-1} U_i + b_i)\\
o_t &= \sigma(x_t W_o + h_{t-1} U_o + b_o)\\
c_t &= f_t \circ c_{t-1} + i_t \circ \text{ReLU}(x_t W_c + h_{t-1} U_c + b_c)\\
h_t &= o_t \circ \text{ReLU}(c_t)\\
y_t &= h_t V
\end{aligned}$$

where, if n is the representation size and d is the network hidden dimension, then

$$\begin{aligned}
x_t \in \mathbb{R}^n, f_t, i_t, o_t, h_t, c_t \in \mathbb{R}^d\\
W \in \mathbb{R}^{n \times d}, U \in \mathbb{R}^{d \times d}\\
V \in \mathbb{R}^{d \times n}, b \in \mathbb{R}^d
\end{aligned}$$

and σ is the sigmoid function. The initial hidden state h_0 and initial cell state c_0 are both set to be the zero vector. We say that an LSTM model with specified parameters has learned to produce ABA sequences if the following holds: when the network is seeded with some entity vector representation as its first input, x_1, then the output y_1 is not equal to x_1 and at the next time step the output y_2 is equal to x_1.

We let $d = 2n + 1$ and assign the following parameters, which provide an analytic solution to producing ABA sequences:

$$f_t = \sigma(x_t \mathbf{0}_{n \times d} + h_{t-1} \mathbf{0}_{d \times d} + \mathbf{N}_d)$$

$$i_t = \sigma(x_t \mathbf{0}_{n \times d} + h_{t-1} \begin{bmatrix}-4 \cdots -4\\ \mathbf{0}_{2n \times n}\end{bmatrix} + \mathbf{N}_d)$$

$$o_t = \sigma(x_t \mathbf{0}_{n \times d} + h_{t-1} \begin{bmatrix}1 \ldots 1\\ \mathbf{0}_{2n \times n}\end{bmatrix} + \mathbf{0}_d)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \text{ReLU}\left(x_t \begin{bmatrix}0\\ \vdots & -I_{n \times n} & I_{n \times n}\\ 0\end{bmatrix} + h_{t-1} \mathbf{0}_{d \times d} + \begin{bmatrix}N & 0 \ldots 0\end{bmatrix}\right)$$

$$h_t = o_t \circ \text{ReLU}(c_t)$$

$$y = h_t \begin{bmatrix}0 \ldots 0\\ -I_{n \times n}\\ I_{n \times n}\end{bmatrix}$$

Where $\mathbf{0}_{j \times k}$ is the $j \times k$ zero matrix, $\mathbf{m}_k$ is a k dimensional vector with each element having the value m, $I_{n \times n}$ is the $n \times n$ identity matrix, and N is some very large number. Now we show that these parameters achieve an increasingly good solution as N increases. When a value involves the number N, we will simplify the computation by saying what that value is equal to as N approaches infinity. We begin with an arbitrary input x_1 and the input and hidden state intialized to zero vectors:

$$h_0 = \mathbf{0}_d \qquad c_0 = \mathbf{0}_d$$

The gates at the first time step are easy to compute, as the cell state and hidden state are zero vectors so the gates are equal to the sigmoid function applied to their respective bias vectors. The forget gate is completely open, the output gate is partially open, and the input gate is fully open:

$$\begin{aligned}
f_1 &= \sigma(\mathbf{N}_d) \approx \mathbf{1}_d\\
o_1 &= \sigma(\mathbf{0}_d) = \mathbf{0.5}_d\\
i_1 &= \sigma(\mathbf{N}_d) \approx \mathbf{1}_d
\end{aligned}$$

Then we compute the cell and hidden states at the first timestep. The cell state encodes the information of the input vector, so it can be used to recover the vector at a later time step and receives no information from the previous cell state despite the forget gate being open, because the previous cell state is a zero vector. The hidden state is the cell state scaled by one half.

$$
\begin{aligned}
c_1 &= \mathbf{N}_d \circ \mathbf{0}_d + \\
&\quad \mathbf{1}_d \circ \mathrm{ReLU}\left(x_1 \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \quad -I_{n\times n} \quad I_{n\times n} \Big] + \begin{bmatrix} N & 0 \dots 0 \end{bmatrix} \right) \\
&= \mathrm{ReLU}\left(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix} \right) \\
h_1 &= \mathbf{0.5}_d \, \mathrm{ReLU}\left(\mathrm{ReLU}(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix}) \right) \\
&= \mathbf{0.5}_d \circ \mathrm{ReLU}(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix})
\end{aligned}
$$

At the next time step, the forget gate remains fully open, the open gate changes from partially open to fully open, and the input gate changes from fully open to fully closed:

$$
\begin{aligned}
f_2 &= \mathbf{N}_d \\
o_2 &= \sigma(y_1 \mathbf{0}_{n\times d} + h_1 \begin{bmatrix} 1 \dots 1 \\ \mathbf{0}_{2n\times n} \end{bmatrix} + \mathbf{0}_d) \\
&= \sigma(\mathbf{0.5}_d \circ \mathrm{ReLU}\left(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix} \right) \begin{bmatrix} 1 \dots 1 \\ \mathbf{0}_{2n\times n} \end{bmatrix} + \mathbf{0}_d) \\
&= \sigma(\mathbf{0.5}_d \circ \mathbf{N}_d) \approx \mathbf{1}_d \\
i_2 &= \sigma(y_1 \mathbf{0}_{n\times d} + h_1 \begin{bmatrix} -4 \cdots -4 \\ \mathbf{0}_{2n\times n} \end{bmatrix} + \mathbf{N}_d) \\
&= \sigma(\mathbf{0.5}_d \circ \mathrm{ReLU}\left(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix} \right) \begin{bmatrix} -4 \cdots -4 \\ \mathbf{0}_{2n\times n} \end{bmatrix} + \mathbf{N}_d) \\
&= \sigma(\mathbf{0.5}_d \circ \mathbf{\text{-}4N}_d + \mathbf{N}_d) \approx \mathbf{0}_d
\end{aligned}
$$

Then we compute the cell and hidden states for the second timestep. Because the forget gate is completely open and the input gate is completely closed, the cell state remains the same. Because the output gate is completely open, the hidden state is the same as the cell state.

$$
\begin{aligned}
c_2 &= \mathbf{1}_d \circ \mathrm{ReLU}\left(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix} \right) + \\
&\quad \mathbf{0}_d \circ \mathrm{ReLU}(x_2 \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \quad -I_{n\times n} \quad I_{n\times n} \Big] + \begin{bmatrix} N & 0 \dots 0 \end{bmatrix}) \\
&= \mathrm{ReLU}\left(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix} \right) \\
h_2 &= \mathbf{1}_d \circ \mathrm{ReLU}\left(\mathrm{ReLU}(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix}) \right) \\
&= \mathrm{ReLU}\left(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix} \right)
\end{aligned}
$$

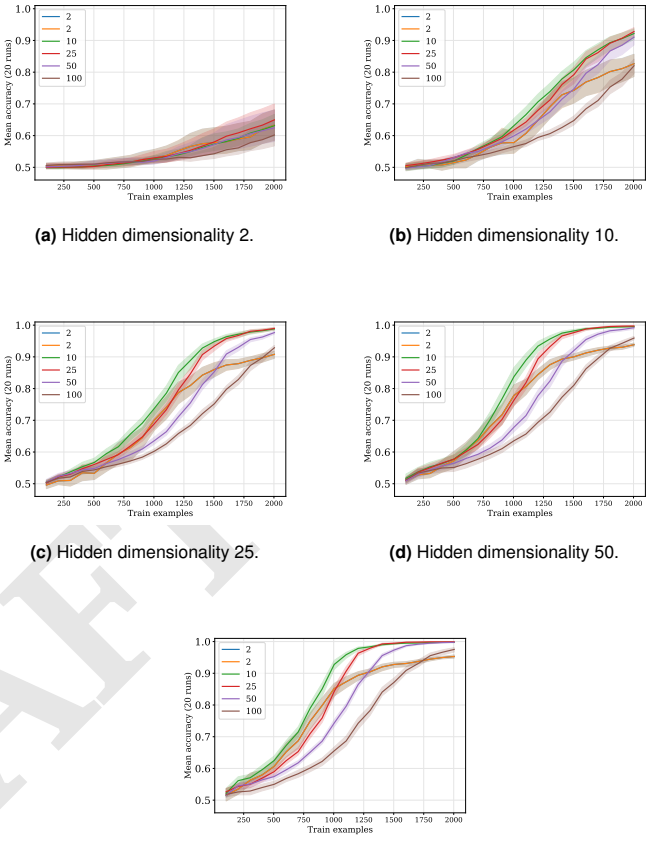
With the hidden states for the first and second time steps, we can compute the output values and find that the output at the first time step is the initial input vector scaled by one half and the output at the second time step is the initial input vector.

$$
\begin{aligned}
y_1 &= h_1 \begin{bmatrix} 0 \dots 0 \\ -I_{n\times n} \\ I_{n\times n} \end{bmatrix} = \mathbf{0.5}_d \circ \mathrm{ReLU}\left(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix} \right) \begin{bmatrix} 0 \dots 0 \\ -I_{n\times n} \\ I_{n\times n} \end{bmatrix} \\
&= \mathbf{0.5}_d \circ x_1 \\
y_2 &= h_2 \begin{bmatrix} 0 \dots 0 \\ -I_{n\times n} \\ I_{n\times n} \end{bmatrix} = \mathtt{ReLu}(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix}) \begin{bmatrix} 0 \dots 0 \\ -I_{n\times n} \\ I_{n\times n} \end{bmatrix} = x_1
\end{aligned}
$$

Then, because $y_1 = \mathbf{0.5}_d \circ x_1 \neq x_1$ and $y_2 = x_1$, this network produces ABA sequences.

Additional results plots

Model 1 for basic same–different. Figure 10 explores a wider range of hidden dimensionalities for Model 1 applied to the basic same–differerent task. As in the paper, the lines correspond to different embedding dimensionalities. Figure 10e is repeated from Figure **??**.



(a) Hidden dimensionality 2.

(b) Hidden dimensionality 10.

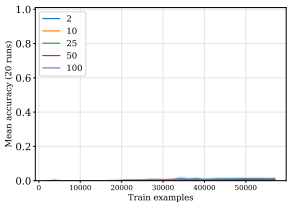
(c) Hidden dimensionality 25.

(d) Hidden dimensionality 50.

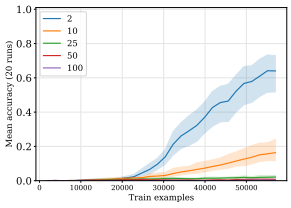
(e) Hidden dimensionality 100.

Fig. 10. Results for Model 1 for basic same–different.

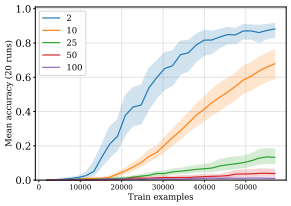
Model 2 for sequential same–different. Figure 11 explores a wider range of hidden dimensionalities for Model 2 applied to the sequential ABA task. As in the paper, the lines correspond to different embedding dimensionalities. The full training set is presented to the model in multiple epochs. Figure 11e is repeated from Figure 3b.



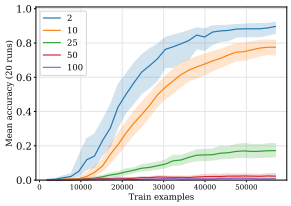
(a) Hidden dimensionality 2.



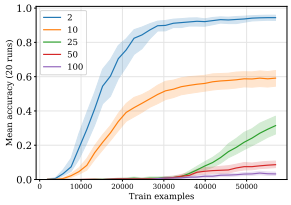
(b) Hidden dimensionality 10.



(c) Hidden dimensionality 25.



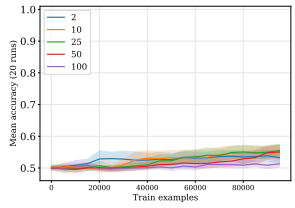
(d) Hidden dimensionality 50.



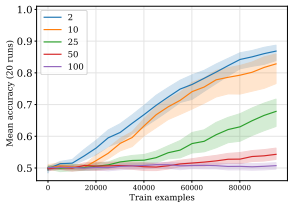
(e) Hidden dimensionality 100.

Fig. 11. Results for Model 2 for basic same–different, with a vocabulary size of 20.

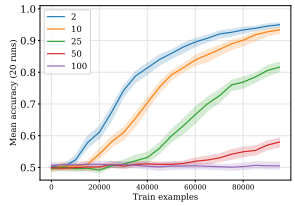
Model 1 for hierarchical same–different. Figure 12 shows the results of applying the model in (16)–(17) to the hierarchical same–different task. The only change from that model is that the inputs have dimensionality $4m$, since the four distinct representations in task inputs are simply concatenated.



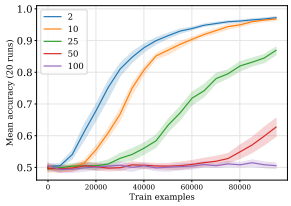
(a) Hidden dimensionality 2.



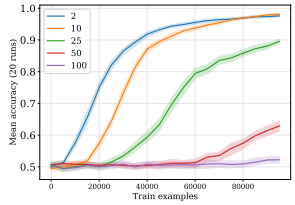
(b) Hidden dimensionality 10.



(c) Hidden dimensionality 25.



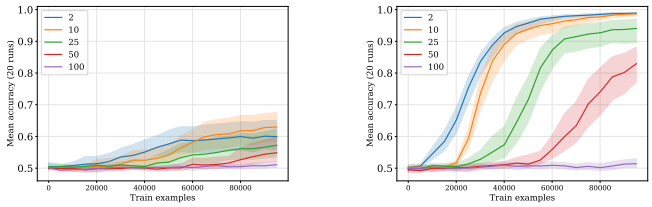
(d) Hidden dimensionality 50.



(e) Hidden dimensionality 100.

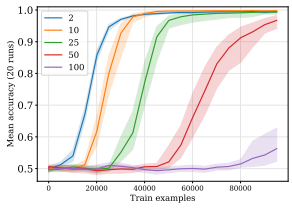
Fig. 12. Results for Model 1 applied to the hierarchical same–different task.

Model 3a for hierarchical same–different. Figure 13 shows the results of applying the model in (22)–(24) to the hierarchical same–different task. Figure 13e is repeated from Figure **??**.

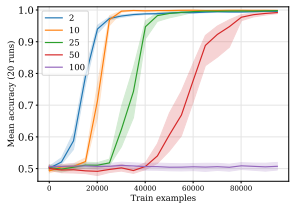


(a) Hidden dimensionality 2.

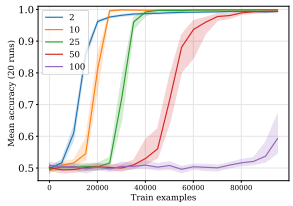
(b) Hidden dimensionality 10.



(c) Hidden dimensionality 25.



(d) Hidden dimensionality 50.



(e) Hidden dimensionality 100.

Fig. 13. Results for Model 3a applied to the hierarchical same–different task.

Pretraining input representations and equality parameters. In Section , we showed that pretraining an entire basic equality network led to faster learning in the hierarchical same–different task. This kind of network pretraining can be combined with the input-level pretrained we explored elsewhere in the paper (details in Appendix). Figure 14 reports initial experiments for this combination of pretraining regimes. Although the results are not better than the ones we report in the paper, this still seems like a promising idea, though one for which optimal solutions might be hard to find.

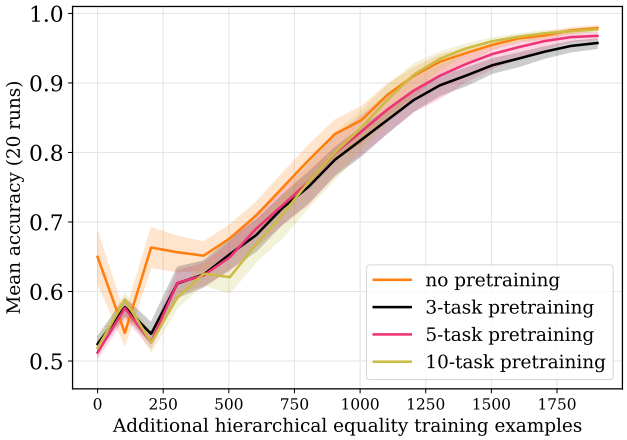


Fig. 14. Results for the hierarchical same–different task for a model in which both the input representations and the basic equality network are pretrained. The 'no pretrain' model is the best one from Figure 4b (25-dimensional embeddings, 100-dimensional hidden representations). The input-pretrained models use this same configuration.