Relational reasoning and generalization using non-symbolic neural networks

Atticus Geiger
Department of Linguistics
Stanford University

Alexandra Carstensen
Department of Psychology
Stanford University

Michael C. Frank
Department of Psychology
Stanford University

Christopher Potts
Department of Linguistics
Stanford University

Abstract

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 in non-symbolic neural networks. Drawing on recent developments in the study of such networks, we assess out-of-sample generalization of equality using both arbitrary representations and arbitrary representations that have been pretrained on separate tasks to imbue them with abstract structure. In this setting, we find that 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, we show that 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.

1 Introduction

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., Gentner and Goldin-Meadow, 2003). By far the most common case study of abstract relations has been equality.[1] 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*

[1] We use the term “equality” here, though different literatures have also used “identity.”

Preprint. Under review.

pair, AA, must be matched to a novel same pair, BB (Premack, 1983; KR Thompson et al., 2001). Preschool children struggle to learn the same regularities in a seemingly similar task (Walker et al., 2016). 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 (Avarguès-Weber et al., 2011), and infants as young as 3 months can generalize an identity pattern (Anderson et al., 2018). 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.

The learning task in all of these cases can be described by the predicate *same* (or equivalently, =). However, this symbolic vocabulary does not provide a lever to distinguish which of these tasks are trivial and which are immensely difficult. What would an explanation of this gradient look like? Ideally, we would want a model or set of models that describe under what conditions *same* is easy and under what conditions it is hard or unlearnable – and how learning proceeds in these hard cases. One perspective in the literature is that this type of learning task requires symbolic representations (Marcus et al., 1999; Premack, 1983) – but this view does not fully explain many of the cases on the table. For instance, explanations of newborn identity detection under this type of account are circular (Gervain et al., 2012): because newborns succeed, they then must have symbolic representations. Does this logic apply also to bees?

1.1 Models of equality learning

Addressing the empirical puzzles of equality learning 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. A range of models of this type have been studied (Alhama and Zuidema, 2019). Symbolic models of generalization (e.g., Frank and Tenenbaum 2011) can be used to make contact with data, but they presuppose the existence of a symbolic equality predicate and hence simply presuppose symbolic abilities in every case of success – an unsatisfying conclusion.

In contrast, neural network models provide a framework for arbitrary function learning that might in principle be used to understand the emergence of equality relations. Contra this proposal, however, Marcus et al. (1999) argued that a broad class of recurrent neural networks were unable to learn equality relations. Marcus et al.'s claims were subsequently challenged by Dienes et al. (1999), Seidenberg and Elman (1999b), Seidenberg and Elman (1999a), Elman (1999), and Negishi (1999), all of whom presented evidence that neural networks are able to learn (at least aspects of) the tasks that Marcus et al. posed. The subsequent debate (reviewed in Alhama and Zuidema, 2019) revealed a striking lack of consensus on some of the ground rules regarding both (1) what representations would qualify as non-symbolic and (2) what sort of generalization would be required to show that the learned function was suitably abstract. We review these issues below; in brief, we adopt (1) completely random distributed representations and (2) fully disjoint training and test vocabularies, with the goal of solving very challenging generalization tasks without building any relevant structure into our basic representations.

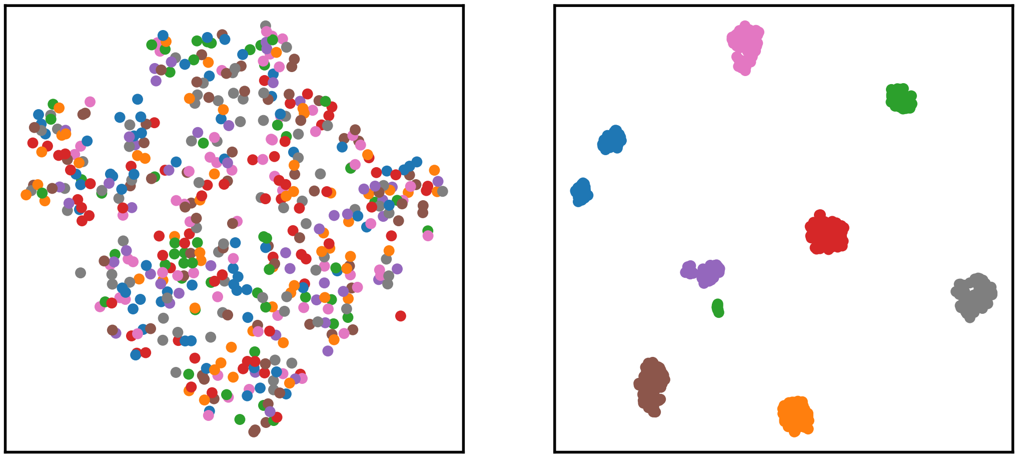
1.2 Representations

Representation schemes for neural network models can be placed in four broad classes. At one end of the spectrum, we have *localist representation* schemes, as in Figure 1b. Here, each row represents an entity, and every entity is a vector with a single 1 at its own column dimension, and in turn every column has exactly one 1 in it. There is no shared structure across objects; all are equally (un)related to each other as far as the model is concerned. These representations are a symbolic system in which all the symbols are meaningless unique identifiers.

In the middle of the spectrum, we find *distributed feature representations*, as in Figure 1c. Here, we build in more structure by designing a rich feature space. In this setting, the entities can have rich relationships to each other, as encoded in the shared structure given by the columns. We can again think of these representations as symbolic, but we would do this via the columns, by naming each column and then considering the entities to be analyzed as various values of these features, and two entities would be labeled as identical if they had exactly the same values for all features.

	🟥	🟦	⬟		⬢	red	blue	sides	green	d_1	d_2	d_3	d_4	d_5	d_1	d_2	d_3	d_4	d_5
🟥	1	0	0	⋯	0	1	0	4	0	0.1	−0.5	−0.2	−0.3	0.2	0.4	0.2	−0.5	−1.1	−0.3
🟦	0	1	0	⋯	0	0	1	4	0	0.2	0.4	−0.4	−0.1	−0.5	0.6	0.7	0.7	0.7	0.5
⬟	0	0	1	⋯	0	1	0	5	0	−0.3	0.0	−0.5	−0.3	0.1	0.8	−0.2	0.1	−0.0	0.5
⋮			⋮				⋮					⋮					⋮		
⬢	0	0	0	⋯	1	0	1	7	0	0.0	−0.3	0.1	0.3	−0.5	0.3	−1.4	0.1	0.4	−0.7

(a) (b) Localist. (c) Distributed. (d) Random. (e) Pretrained.



(f) The matrix in Figure 1d (before pretraining).

(g) The matrix in Figure 1e (after pretraining).

Figure 1: Representations options for the notional entities in Figure 1a.

At the other end of the spectrum, we have *completely random feature representations*, as in Figure 1d. Here, each entity is represented by a unique vector of random real-valued numbers. As with localist encoding, this defines all the entities as equally (un)related to each other, since column-wise patterns are unlikely and, to the extent that they are present, they exist completely by chance.

In between distributed and random representations are *pretrained* representations. These are typically obtained by starting with random representations and imbuing them with rich structure via a learning process. Figure 1e provides a simple example. This matrix is the results of pretraining the representations in Figure 1d against three independent binary classification objectives. (Appendix B provides technical details on our pretraining approach.) Superficially, the two matrices looks equally random. However, this is only because the pretraining process leads to very abstract kinds of structure. Intuitively, individual features can be encoded in multiple dimensions of these spaces, making it hard to discern. We can bring out this latent structure using visualizations such as those in Figure 1f and Figure 1g, which uses the t-SNE algorithm (van der Maaten and Hinton, 2008) to depict an approximation of the original high-dimensional matrices in two dimensions. We use color to distinguish the different label combinations determined by the three tasks. At left, we have the representations in Figure 1d, and at right we have the pretrained representations. The visualization reveals that pretraining has created structure akin to what we see transparently in distributed representations.

Pretraining need not be restricted to input representations. In principle, entire networks can be pretrained to solve specific tasks. This more general form of pretraining has the potential to find latent associations even between completely random inputs, since such networks can learn parameters that define intermediate representations with potentially as much structure as one would encode by hand in a featural representation. Such pretraining offers the possibility that networks might be used as modular components to solve more complex tasks.

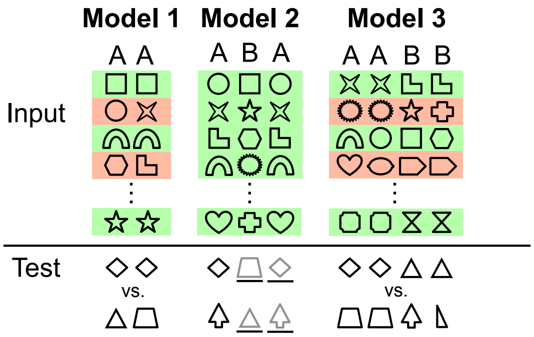


Figure 2: Relational reasoning tasks. Green and red mark positive and negative training examples, respectively. The sequential task (Model 2) uses only positive instances, and a model succeeds if, prompted with α, it produces a sequence $\beta\ \alpha$ for $\beta \neq \alpha$. For the hierarchical task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

Localist representations are transparently symbolic and useless in disjoint generalization tasks. In Appendix D, we show analytically that localist representations cannot lead to out-of-sample generalization. In contrast, distributed feature representations can be useful in such tasks, but they require grounding entities in property domains such as color, shape, or sound. Thus, such featural representations both require specialized domain knowledge and (arguably) build in symbolic structure (Marcus, 1999). We avoid these issues here by representing all entities using completely random distributed representations. This allows us to provide general results on relational learning, independent from the domains that entities are grounded in, and it forces our models to learn in a completely unstructured space. In addition, we report results for pretrained representations, showing that they consistently lead to even faster learning.

1.3 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 propose 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. Where pretrained inputs are used, V_T and V_A must come from the same representation space, but we still define V_T and V_A using completely different vocabularies. In adopting these methods, we get a clear picture of the system's capacity to generalize, and we can 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. In the language modeling context, we will see that it even requires us to departure from usual model formulations.

1.4 The current paper

This paper revisits the debate about the capacity of neural networks to solve relational reasoning tasks in ways that resemble human behavior. We consider three cases of identity-based reasoning that have featured prominently in discussions of the role of symbols in relational reasoning (Figure 2): (1) learning to discriminate pairs of objects that exemplify the relation *same* or *different*, (2) learning sequences with repeated *same* elements, as in Marcus et al. (1999), and (3) learning to distinguish hierarchical *same* and *different* relations in a context with pairs of pairs exemplifying these relations, as in Premack (1983).

To preview our results, our central finding is that even simple neural networks are effective at the tasks we pose, and that pretraining the input representations generally leads to faster learning. The sequence learning task shows additionally that explicit negative evidence is not always needed. However, for hierarchical equality, an unrealistic amount of training data is required. To address this concern, we consider the extended pretraining regimes mentioned in sec. 1.2, in which a network trained for equality is used as a modular component in a larger network. Pretraining greatly reduces the data demands of our models, even leading to success with no additional training in the hierarchical task.

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 entirely data-driven, non-symbolic learning processes.

2 Model 1: Same–different relations

In our first model, we consider whether a supervised, feed-forward classification model can learn equality (and inequality) relations in the strict setting we describe above: random representations with disjoint generalization examples.

2.1 Model

Our basic model for equality is a feed-forward neural network with a single hidden representation layer:

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

$$y = \mathbf{softmax}(hW_{hy} + b_y) \tag{2}$$

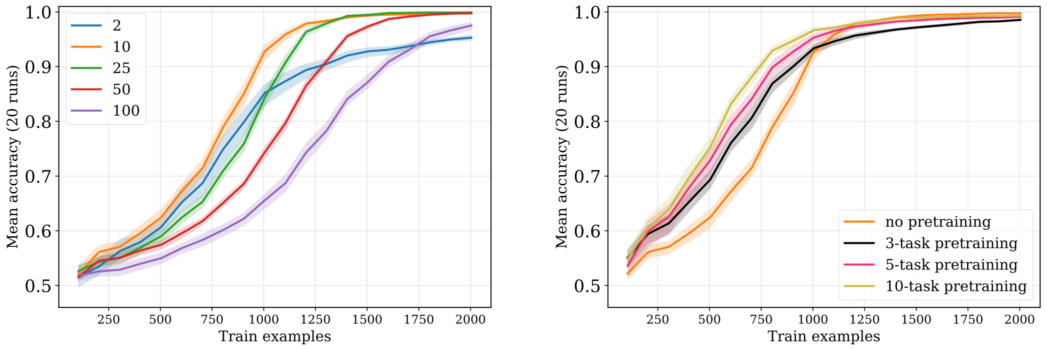
The input is a pair of vectors (a, b), each of dimension m, which correspond to the two stimulus objects. These vectors are random distributed representations that do not have features encoding properties of the objects or their identity. These are concatenated to form a single vector $[a; b]$ of dimension $2m$, 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 ($\mathrm{ReLU}(x) = \max(0, x)$) is applied element-wise to this linear projection. This non-linearity is what gives the neural model more expressive power than a logistic regression. 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: $\mathbf{softmax}(x)_i = \frac{\exp x_i}{\sum_j \exp x_j}$. 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 loss function. This function is defined as follows, for a corpus of N examples and K classes:

$$\max(\theta) \quad \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y^{i,k}\log(h_\theta(i)^k) \tag{3}$$



(a) Results for a model in which the hidden layer has dimensionality 100. The lines correspond to different dimensions for the entities a and b in the input pairs (a, b). These models largely pass 90% accuracy with 1,000 training examples and reach (near) perfection by 1,250.

(b) Results for pretraining on different numbers of tasks. The 'no pretraining' model is the best of the models at left (10-dimensional embeddings, 100-dimensional hidden layers), repeated to faculitate comparisons. The pretraining models use those same dimensionalities.

Figure 3: Same–different results with and without pretraining.

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.

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 E provides an analytic solution to the equality relation using this neural model. Here we illustrate with a small example network that maps all identity pairs to $[0.5, 0.5]$ and all non-identity pairs to $[y, 1-y]$ where $y > 0.5$, which supports a trivial classification rule:

$$\text{ReLU}\left([a; b]\begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix} + \mathbf{0}\right)\begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix} + \mathbf{0} \tag{4}$$

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. Further optimization details are in Appendix C.

2.2 Results

Figure 3a presents typical results. This is for the case where the hidden layer dimensionality is 100, and we plot results for different embedding dimensionalities m and different amounts of training data. The picture is comparable with hidden dimensionalities at 10, 25, and 50, but those models require more training data to reach (near) perfect performance (Appendix G.1).

2.3 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.

While all the networks in Figure 3a reach above-chance performance almost immediately, they require upwards of 1,000 examples to truly solve these tasks. To try to speed up learning, we ran the above experiments using pretrained representations as well, holding out one of the pretraining classes as our test set as described above. Figure 3b summarizes the results of these experiments for 10-dimensional embeddings and 100-dimensional hidden representations, as this seems to be the network that learns the fastest with random inputs. We see a clear speed-up, with more pretraining tasks resulting in the largest gains.

3 Model 2: Sequential same–different (ABA task)

Our first model is simple and successfully learns equality. However, this model was 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., Marcus, 1999). While this argument likely holds true for language learning (in which supervision is generally agreed not to be direct; Brown and Hanlon, 1970), it is not necessarily true for learning more generally. Nevertheless, learning of sequential ABA rules without negative feedback is possible for infants (Marcus et al., 1999; Rabagliati et al., 2019). Our next model explores whether neural network models can learn this task in a challenging regime with no negative supervision.

3.1 Model

To explore learning with only positive instances, we adopt a neural language model. The hope is that these models can learn the underlying principles that govern their inputs, for the purposes of generating new sequences or discriminating among different kinds of input.

Neural language models are sequential models. 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 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 using an entirely different vocabulary. The classifier function will get no feedback about these out-of-vocabulary items during training, and so it will never predict them during assessment. To address this issue, we need to 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.

The specific model we use for this is as follows:

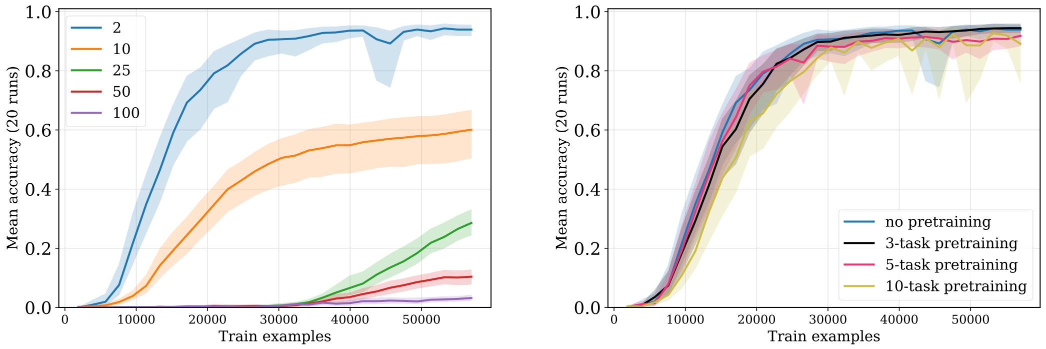
$$h_t = \mathbf{LSTM}(x_t, h_{t-1}) \tag{5}$$

$$y_t = h_t W + b \tag{6}$$

This holds for $t > 0$, and we set $h_0 = \mathbf{0}$. **LSTM** is a long short-term memory cell (Hochreiter and Schmidhuber, 1997). Full details on these cells are given in sec. F.

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, random distributed 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.



(a) Results for a model in which each h_t in Eq. (5) has dimensionality 100. The lines correspond to the dimensionality of the input representations (x_t in Eq. (5)). All the training examples are presented at once over multiple epochs.

(b) Results for pretraining on different numbers of tasks. The 'no pretraining' model is the best of the models at left in terms of speed and overall performance (2-dimensional embeddings, 100-dimensional hidden representations). The pretraining models use those same dimensionalities. Pretraining does not lead to a noticeable change in speed or accuracy for this task.

Figure 4: Sequential same–different results with and without pretraining.

The objective function is as follows:

$$\max(\theta) \quad \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T_i}\left\|h_\theta\left(x^{i,0:t-1}\right)-x^{i,t}\right\|^2 \tag{7}$$

for N examples. Here, T_i is the length of example i. As before, θ abbreviates the parameters of the model as specified in (5)–(6). 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).

As noted above, this is an unusual formulation for a language model. The usual version essentially treats every timestep as involving a classification decision, with a cross-entropy loss. We cannot adopt this because of our goal of using unseen vocabulary items at test time.

Appendix F provides an analytic solution to the ABA task using this model. To see how well it does 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`.

3.2 Results

Figure 4a shows the results for a model with a 100-dimensional hidden representation. As before, the results are comparable for smaller networks. Unlike for the previous equality experiments, we found that we had to allow the model to experience multiple epochs of training on the same set in order to succeed, and hence the x-axis counts examples in terms of epochs of learning on our fixed set of 380 examples.

3.3 Discussion

These models succeed at learning the underlying patterns in our data, even though they use only random representations. They are given no negative instances, and they must predict into a totally new vocabulary. The shortcoming we see here is that the learning process is admittedly slow and data-intensive. We hypothesized that pretraining leads to noticeable speed-ups, as it did in sec. 2, but we did not see this in practice; Figure 4b shows that pretraining does not help with learning. However, we speculate that there may be variants of our model that reduce these demands, given that learning is in principle possible in this architecture.

4 Model 3: Hierarchical same–different relations

Given the strong results for equality, we can ask whether more challenging equality problems are also learnable in our setting. An interesting test-case in this regard is the hierarchical equality task used by Premack (1983): given a pairs of pairs $((a, b), (c, d))$, the label is 1 if $(a = b) = (c = d)$, else 0. Premack (1983) 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.

4.1 Model

We can approach this task using the same model and methods as we used for equality. The only change required to equations (1)–(2) is that we create inputs $[a; b; c; d]$: the flat concatenation of all the elements of the two pair of vectors. This in turn leads W_{xh} to have dimensionality $4m \times n$.

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 reaches near perfect accuracy, but only with over 95,000 training instances, and a comparable model with 50-dimensional entity representations failed to get traction at all with this amount of training data. Appendix G.3 provides a full picture of thee learning trends. Pretraining does lead to faster and more stable learning, but the improvements are relatively minor.

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 can simply add another intermediate layer:

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

4.2 Results

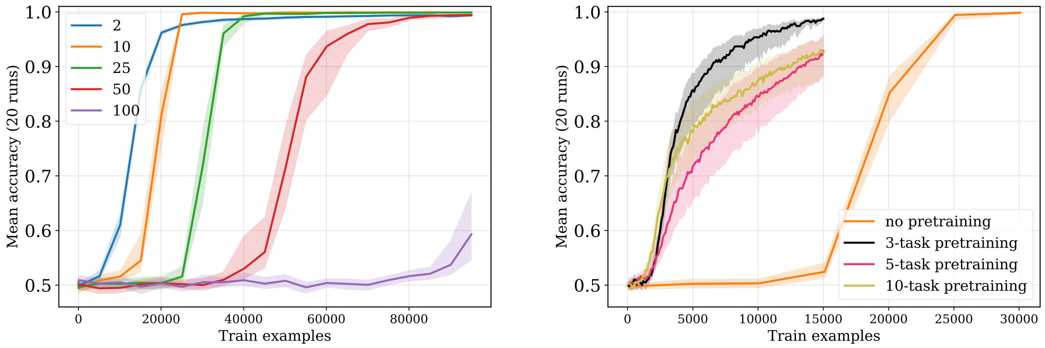
Figure 5a shows that these deeper networks learn faster and are more robust to different network configurations than our single-layer variant. Intuitively, these models are better structured to find hierarchical solutions to this hierarchical problem.

4.3 Discussion

These results are encouraging, but we still require more than 20,000 training instances to reach top performance. As Figure 5b shows, pretraining helps substantially with this shortcoming, but we still require upwrds of 10,000 examples to fully solve the problem. This is still vastly more data than human participants get in similar experiments. Thus, it is worth asking whether there are other solutions that would be more data efficient and thus more in line with human capabilities. In the next section, we seek 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.

4.4 The critical role of experience

Our successful results training neural networks on simple equality suggest another strategy for solving the hierarchical equality task. Rather than requiring our networks to find solutions from scratch, we



(a) Results for a network with two 100-dimensional hidden layers. Nearly all the networks solve the task, but they require very large training sets to do so.

(b) Results for pretraining on different numbers of tasks. The 'no pretraining' model is the best of the models att left in terms of speed (10-dimensional embeddings, 100-dimensional hidden representations). It is repeated for comparison, but we show only up to 30,000 examples to reveal more detail in the learning trends. Pretaining greatly increases the speed of learning.

Figure 5: Hierarchical same–different results with and without pretraining.

can pretrain them to do basic equality and then use those parameters as a starting point for learning hierarchical equality. This is conceptually similar to our previous experiments with pretraining, but now we seek to pretrain an entire subpart of the model, rather than just its inputs.

4.4.1 Model

The hierarchical equality task requires computing the equality relation three times: compute whether the first two inputs are equal, compute whether the second two inputs are equal, then compute whether the truth value 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. More precisely, we define

$$h_1 = \mathrm{ReLU}([a; b]W_{xh} + b_h) \tag{11}$$

$$h_2 = \mathrm{ReLU}([c; d]W_{xh} + b_h) \tag{12}$$

$$h_3 = \mathrm{ReLU}([h_1; h_2]W_{xh} + b_h) \tag{13}$$

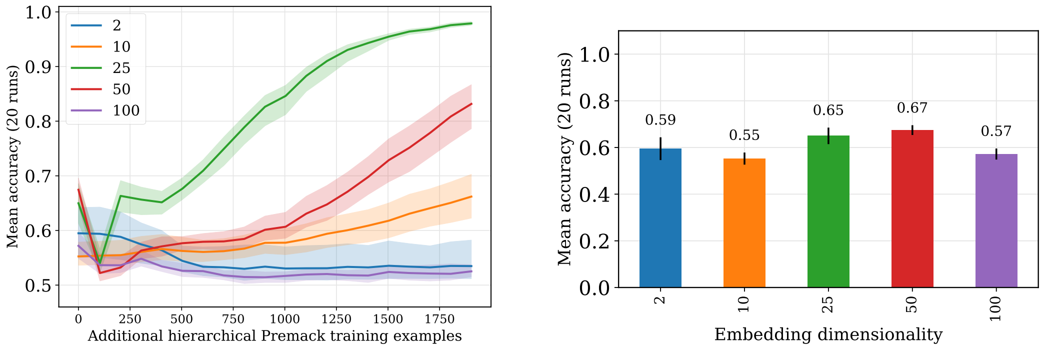
$$y = \mathbf{softmax}(h_3 W_{hy} + b_y) \tag{14}$$

where W_{xh}, W_{hy}, b_h, and b_y are the parameters from the model in equations (1)–(2) 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.

4.4.2 Results and discussion

Figure 6 shows that this model succeeds very quickly at this task. As we see in Figure 6a, after just a few thousand examples, the majority of the model configurations perform with near perfect accuracy. The findings in Figure 6b are even more striking: 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. It is remarkable that a model trained only on 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.

It might be possible to effectively combine network pretraining with input pretraining as in the previous experiments. An initial exploration of this idea is presented in Appendix G.5. While we have not yet found a way to use this combination of pretraining regimes to improve over Figure 6, we are optimistic about such combinations.



(a) Basic equality networks applied to the hierarchical equality task. Each line represents a different embedding dimensionality, which is constrained in this model to match the hidden dimensionality. Even with no additional training instances for this task, all models achieve greater than chance accuracy, and even modest amounts of additional training on the task lead to excellent performance.

(b) The results from the plot at left where the x-axis is 0, that is, where we are testing zero-shot generalization from the pretrained equality network to the hierarchical setting. All the models perform well above chance in this setting, with the 50-dimensioanl version achieving a mean of 67% accuracy.

Figure 6: Modular network results for the hierarchical same–different task.

5 General discussion

We presented a series of neural network models for learning and generalizing equality relations, showing that these non-symbolic models succeed even in stringent assessments, and, in the case of neural language models, even without being shown negative examples. We support these results with analytic insights into why networks are able to find solutions to these tasks.

In some settings, our current models require more training instances than humans seem to need. However, our pretraining approach to the hierarchical equality task 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, and even a small amount of additional training can be transformative. We hypothesize that these modular pretrained components might serve as the basis for even more complex cognitive abilities.

Acknowledgements

This work is supported in part by a Facebook Robust Deep Learning for Natural Language Processing Research Award.

References

Raquel G Alhama and Willem Zuidema. 2019. A review of computational models of basic rule learning: The neural-symbolic debate and beyond. *Psychonomic bulletin & review*, 26(4):1174–1194.

Erin M Anderson, Yin-Juei Chang, Susan Hespos, and Dedre Gentner. 2018. Comparison within pairs promotes analogical abstraction in three-month-olds. *Cognition*, 176:74–86.

Aurore Avarguès-Weber, Nina Deisig, and Martin Giurfa. 2011. Visual cognition in social insects. *Annual review of entomology*, 56:423–443.

Roger Brown and Camille Hanlon. 1970. Derivational complexity and order of development in speech. In J. R. Hayes, editor, *Cognition and the development of language*. Wiley.

Zoltán Dienes, Gerry TM Altmann, and Shi-Ji Gao. 1999. Mapping across domains without feedback: A neural network model of transfer of implicit knowledge. *Cognitive Science*, 23(1):53–82.

Jeffrey L. Elman. 1999. Generalization, rules, and neural networks: A simulation of Marcus et. al. HTML document.

Michael C Frank and Joshua B Tenenbaum. 2011. Three ideal observer models for rule learning in simple languages. *Cognition*, 120(3):360–371.

Dedre Gentner and Susan Goldin-Meadow. 2003. *Language in mind: Advances in the study of language and thought*. MIT press.

Judit Gervain, Iris Berent, and Janet F Werker. 2012. Binding at birth: The newborn brain detects identity relations and sequential position in speech. *Journal of Cognitive Neuroscience*, 24(3):564–574.

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

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*.

Roger KR Thompson, Mary Jo Rattermann, and David L Oden. 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.

Laurens van der Maaten and Geoffrey E. Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

Gary F Marcus. 1999. Rule learning by seven-month-old infants and neural networks. Response to Altmann and Dienes. *Science*, 284:875.

Gary F Marcus, Sugumaran Vijayan, S Bandi Rao, and Peter M Vishton. 1999. Rule learning by seven-month-old infants. *Science*, 283(5398):77–80.

Michiro Negishi. 1999. Do infants learn grammar with algebra or statistics? *Science*, 284(5413):435.

David Premack. 1983. The codes of man and beasts. *Behavioral and Brain Sciences*, 6(1):125–136.

Hugh Rabagliati, Brock Ferguson, and Casey Lew-Williams. 2019. The profile of abstract rule learning in infancy: Meta-analytic and experimental evidence. *Developmental science*, 22(1):e12704.

Mark S Seidenberg and Jeff L Elman. 1999a. Do infants learn grammar with algebra or statistics? *Science*, 284(5413):433–433.

Mark S Seidenberg and Jeffrey L Elman. 1999b. Networks are not 'hidden rules'. *Trends in Cognitive Sciences*, 3(8):288–289.

Caren M Walker, Sophie Bridgers, and Alison Gopnik. 2016. The early emergence and puzzling decline of relational reasoning: Effects of knowledge and search on inferring abstract concepts. *Cognition*, 156:30–40.

A Model figures

A.1 Model 1: Same–different relation with feed-foward networks

Our model of equality is given by (15)–(16):

$$h = \mathrm{ReLU}([a;b]W_{xh} + b_h) \tag{15}$$

$$y = \mathbf{softmax}(hW_{hy} + b_y) \tag{16}$$

where a and b have dimension m (the embedding dimension), W_{xh} is a weight matrix of dimension $2m \times n$, b_h is a bias vector of dimension n, W_{hy} is a weight matrix of dimension $n \times 2$, b_y is a bias vector of dimension 2, $\mathrm{ReLU}(x) = \max(0, x)$, and $\mathbf{softmax}(x)_i = \frac{\exp x_i}{\sum_j \exp x_j}$.

Figure 7 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).

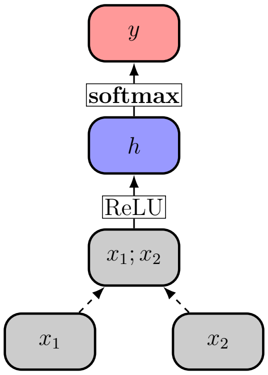


Figure 7: A single layer network computing equality.

A.2 Model 2: Sequential same–different (ABA task)

The specific model we use for this is as follows:

$$h_t = \textbf{LSTM}(x_t, h_{t-1}) \tag{17}$$
$$y_t = h_t W + b \tag{18}$$

for $t > 0$, with $h_0 = \mathbf{0}$. **LSTM** is a long short-term memory cell. Figure 8 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 $t + 1$.

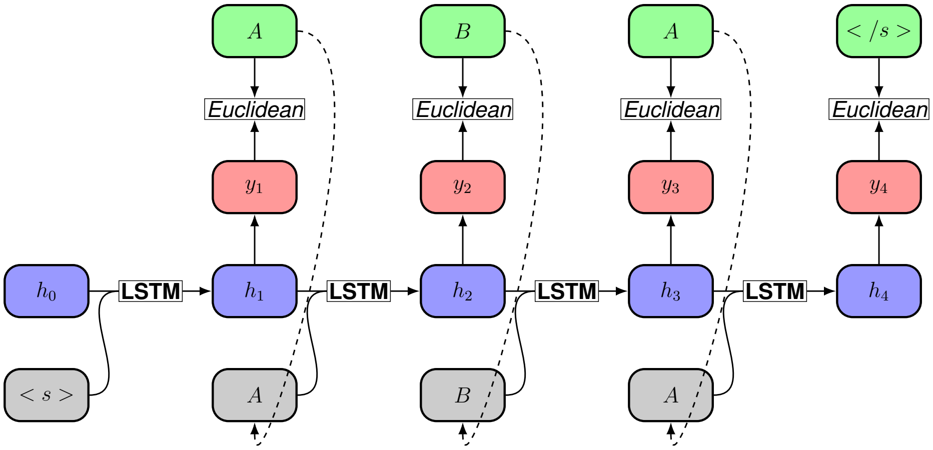


Figure 8: A recursive LSTM network producing ABA sequences.

A.3 Model 3a: A deeper feed-forward network for hierarchical same–different

This model extends (15)–(16) 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 $[a;b;c;d]$:

$$h_1 = \text{ReLU}([a;b;c;d]W_{xh} + b_{h_1}) \tag{19}$$
$$h_2 = \text{ReLU}(h_1 W_{hh} + b_{h_2}) \tag{20}$$
$$y = \textbf{softmax}(h_2 W_{hy} + b_y) \tag{21}$$

Figure 9 depicts this model.

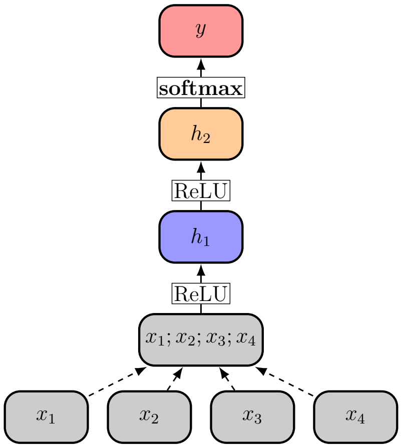


Figure 9: A two layer network computing hierarchical equality.

A.4 Model 3b: Pretraining for hierarchical same–different

Our pretraining model is as follows:

$$h_1 = \text{ReLU}([a;b]W_{xh} + b_h) \tag{22}$$
$$h_2 = \text{ReLU}([c;d]W_{xh} + b_h) \tag{23}$$
$$h_3 = \text{ReLU}([h_1;h_2]W_{xh} + b_h) \tag{24}$$
$$y = \textbf{softmax}(h_3 W_{hy} + b_y) \tag{25}$$

where W_{xh}, W_{hy}, b_h, and b_y are the parameters from the model in (15)–(16) already trained on basic equality. Figure 10 depicts this model. The colors indicate where parameters are reused by the model.

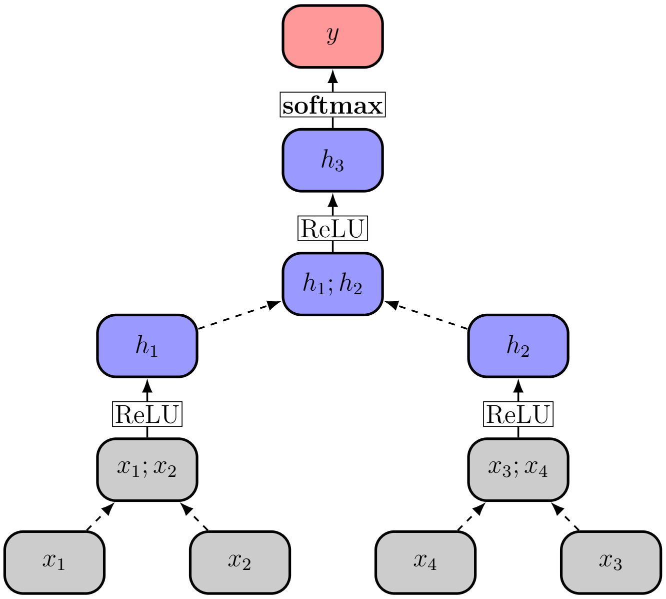


Figure 10: A single layer network pretrained on equality computing hierarchical equality.

B Pretraining

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

$$h_i = \mathrm{ReLU}\left(E[i]W_{xh} + b_h\right) \tag{26}$$

$$y_{i,j} = \mathbf{softmax}(h^i W_{hy}^j + b_y^j) \tag{27}$$

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

$$\max(\theta) \quad \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{J}\sum_{k=1}^{K_j} y^{i,j,k}\log\left(h_\theta(i)^{j,k}\right) \tag{28}$$

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.

C 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 (Kingma and Ba, 2014) 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.

D 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. Random distributed 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.

E 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 11 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.

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

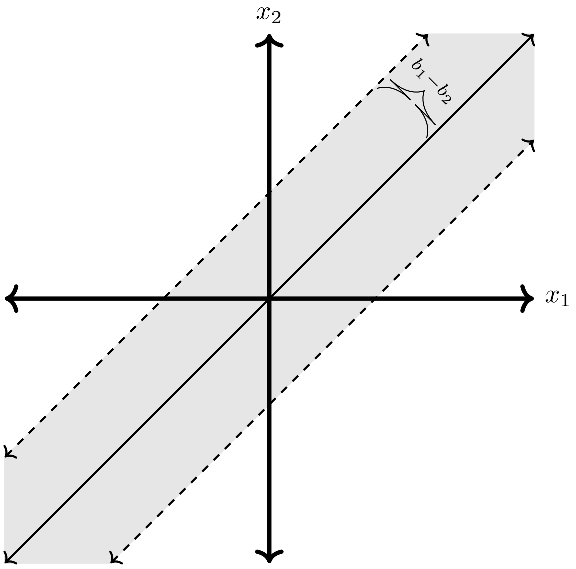


Figure 11: A visual representation of how the analytic solution to identity of a single layer feed forward network changes depending on the values of two bias terms, b_1, b_2.

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.

F An analytic solution to ABA sequences

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

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

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:

$$\begin{aligned}
f_t &= \sigma(x_t \mathbf{0}_{n \times d} + h_{t-1} \mathbf{0}_{d \times d} + \mathbf{N}_d) \\
i_t &= \sigma\left(x_t \mathbf{0}_{n \times d} + h_{t-1} \begin{bmatrix} -4 \cdots -4 \\ \mathbf{0}_{2n \times n} \end{bmatrix} + \mathbf{N}_d\right) \\
o_t &= \sigma\left(x_t \mathbf{0}_{n \times d} + h_{t-1} \begin{bmatrix} 1 \dots 1 \\ \mathbf{0}_{2n \times n} \end{bmatrix} + \mathbf{0}_d\right) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \text{ReLU}\left(x_t \begin{bmatrix} 0 & & \\ \vdots & -I_{n \times n} & I_{n \times n} \\ 0 & & \end{bmatrix} + h_{t-1} \mathbf{0}_{d \times d} + [N \quad 0 \dots 0] \right) \\
h_t &= o_t \circ \text{ReLU}(c_t) \\
y &= h_t \begin{bmatrix} 0 \dots 0 \\ -I_{n \times n} \\ I_{n \times n} \end{bmatrix}
\end{aligned}$$

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.

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

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:

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

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.

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

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.

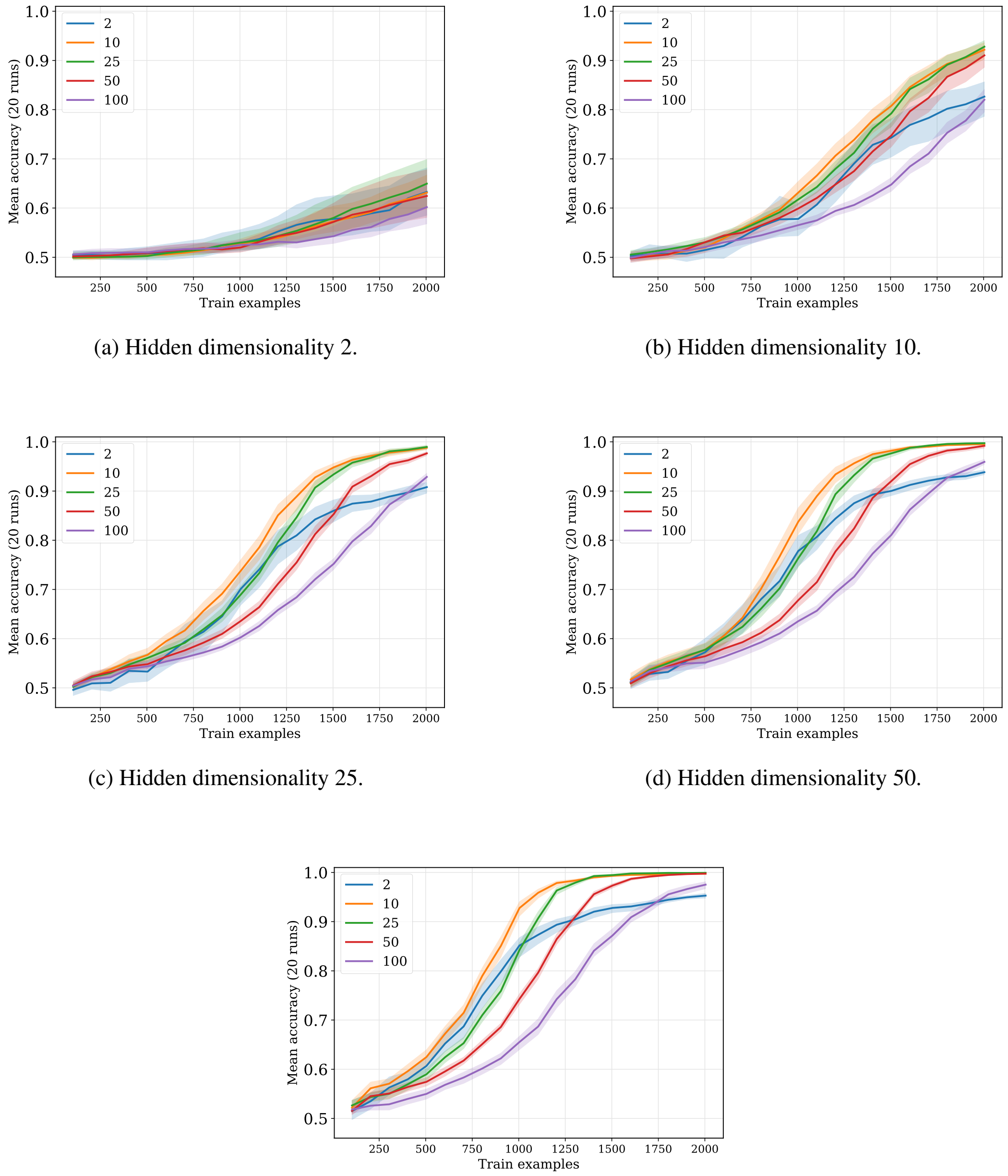
$$
\begin{aligned}
y_1 &= h_1 \begin{bmatrix} 0 \dots 0 \\ -I_{n\times n} \\ I_{n\times n} \end{bmatrix} = \mathbf{0.5}_d \circ \mathrm{ReLU}\left([N \quad -x_1 \quad x_1]\right) \begin{bmatrix} 0 \dots 0 \\ -I_{n\times n} \\ I_{n\times n} \end{bmatrix} = \mathbf{0.5}_d \circ x_1 \\
y_2 &= h_2 \begin{bmatrix}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

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

G Additional results plots

G.1 Model 1 for basic same–different

Figure 12 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 12e is repeated from Figure 3a.



(a) Hidden dimensionality 2.

(b) Hidden dimensionality 10.

(c) Hidden dimensionality 25.

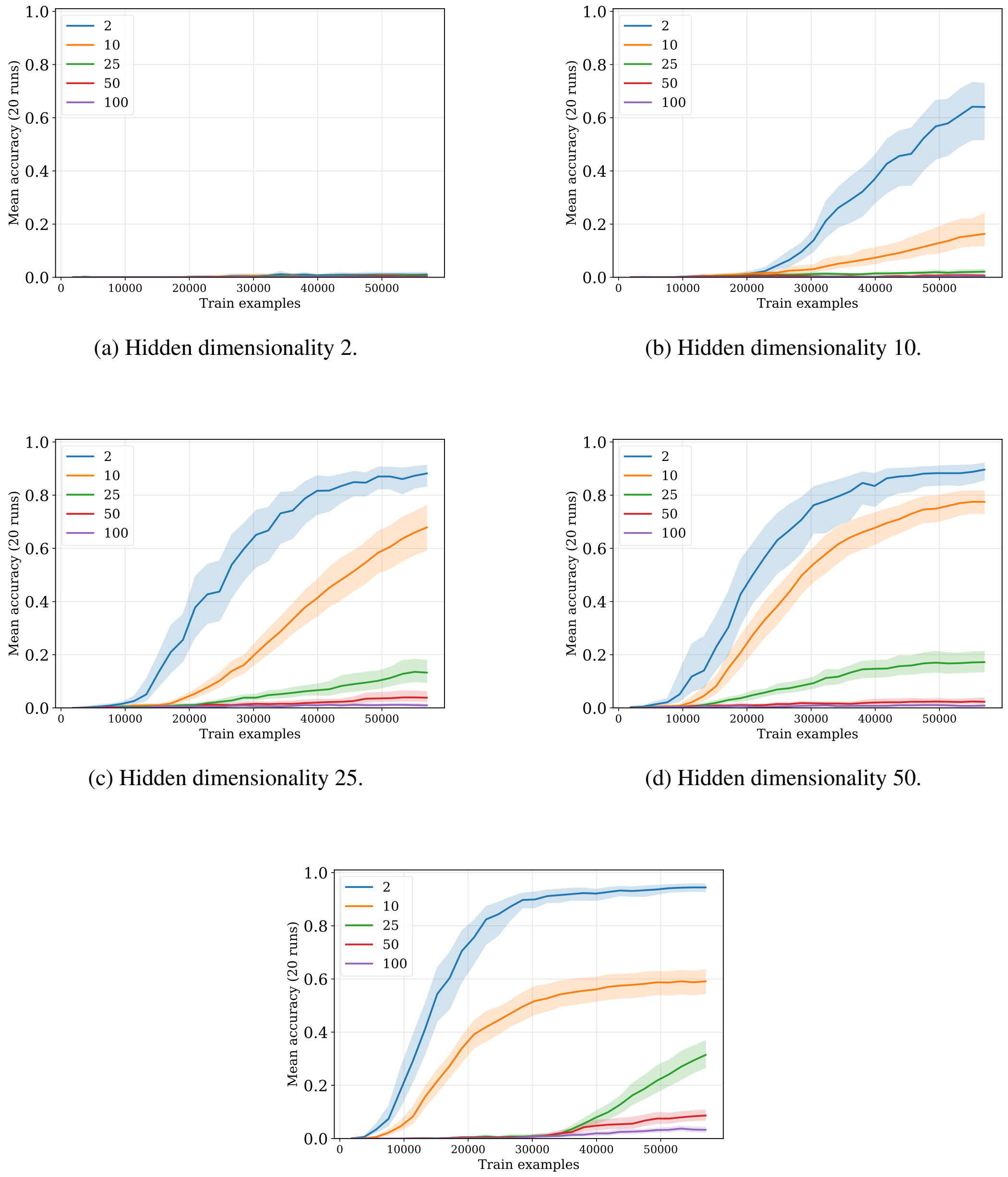
(d) Hidden dimensionality 50.

(e) Hidden dimensionality 100.

Figure 12: Results for Model 1 for basic same–different.

G.2 Model 2 for sequential same–different

Figure 13 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 13e is repeated from Figure 4b.



(a) Hidden dimensionality 2.

(b) Hidden dimensionality 10.

(c) Hidden dimensionality 25.

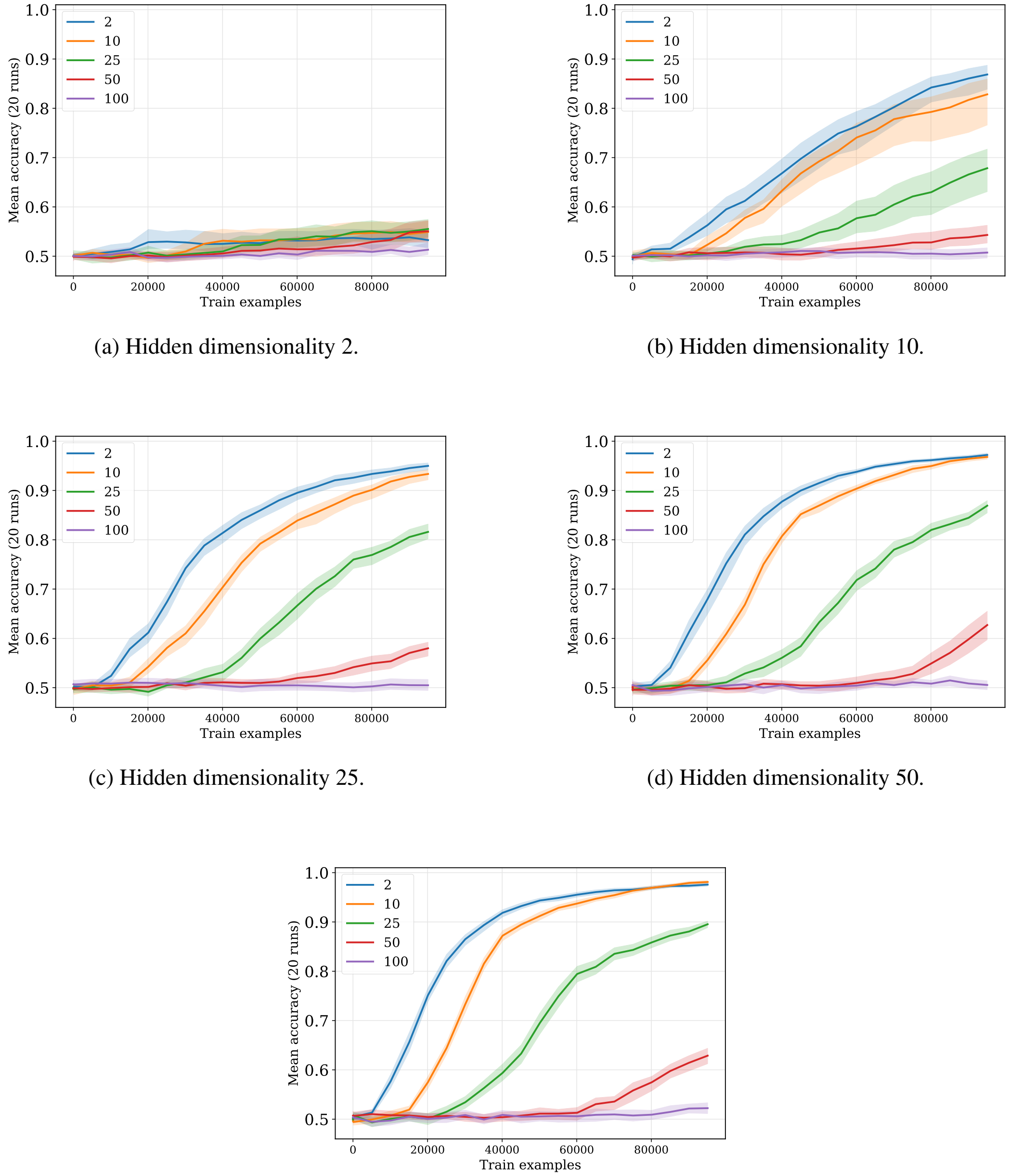
(d) Hidden dimensionality 50.

(e) Hidden dimensionality 100.

Figure 13: Results for Model 2 for basic same–different, with a vocabulary size of 20.

G.3 Model 1 for hierarchical same–different

Figure 14 shows the results of applying the model in (15)–(16) 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.

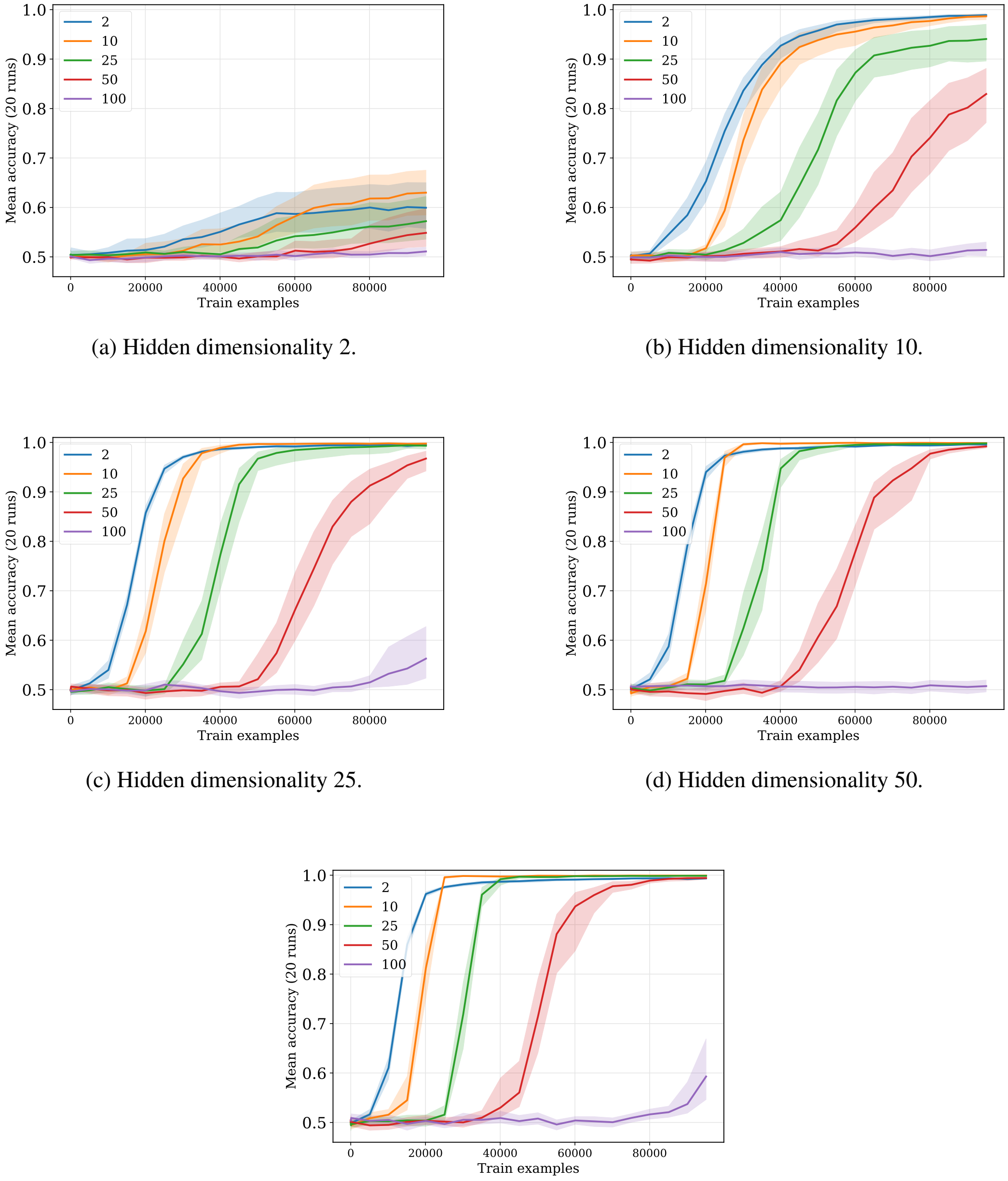
(d) Hidden dimensionality 50.

(e) Hidden dimensionality 100.

Figure 14: Results for Model 1 applied to the hierarchical same–different task.

G.4 Model 3a for hierarchical same–different

Figure 15 shows the results of applying the model in (19)–(21) to the hierarchical same–different task. Figure 15e is repeated from Figure 5a.



(a) Hidden dimensionality 2.

(b) Hidden dimensionality 10.

(c) Hidden dimensionality 25.

(d) Hidden dimensionality 50.

(e) Hidden dimensionality 100.

Figure 15: Results for Model 3a applied to the hierarchical same–different task.

G.5 Pretraining input representations and equality parameters

In sec. 4.4, 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 B). Figure 16 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.

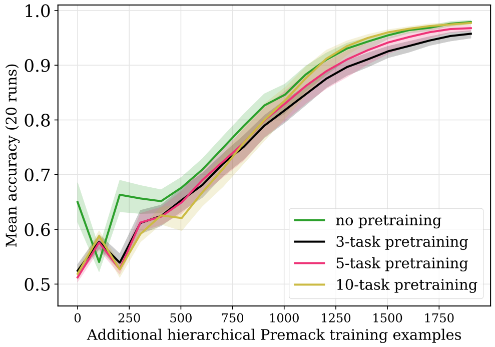


Figure 16: Results for the hierarchical same–different task for a model in which both the input representations and the basic equality network are pretrained. The ‘no pretrain’ model is the best one from Figure 6a (25-dimensional embeddings, 100-dimensional hidden representations). The input-pretrained models use this same configuration.