Supplementary Information for

Relational reasoning and generalization using non-symbolic neural networks

Atticus Geiger, Alexandra Carstensen, Michael C. Frank, and Christopher Potts

Corresponding Author: Atticus Geiger
E-mail: atticusg@stanford.edu

This PDF file includes:

Supplementary text
Figs. S1 to S9
SI References

Supporting Information Text

1. Model Details

A. Model 1: Same–different relation with feed-forward networks. Our model of equality is given by (1)–(2):

$$h = \mathrm{ReLU}([a; b]W_{xh} + b_h) \quad [1]$$
$$y = \mathbf{softmax}(hW_{hy} + b_y) \quad [2]$$

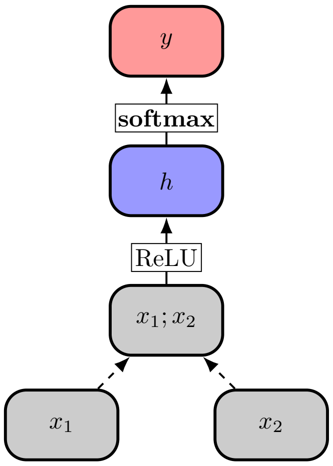


Fig. S1. A single layer network computing equality.

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 have features 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 ($\mathrm{ReLU}(x) = \max(0, x)$) is applied element-wise to this linear projection. This non-linearity is what gives the neural model more expressive power than a logistic regression (1, 2). 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.

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:

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

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 S1 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 (**??**)–(**??**).

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

$$h_t = \mathbf{LSTM}(x_t, h_{t-1}) \quad [4]$$
$$y_t = h_t W + b \quad [5]$$

This holds for $t > 0$, and we set $h_0 = \mathbf{0}$. **LSTM** is a long short-term memory cell (3).

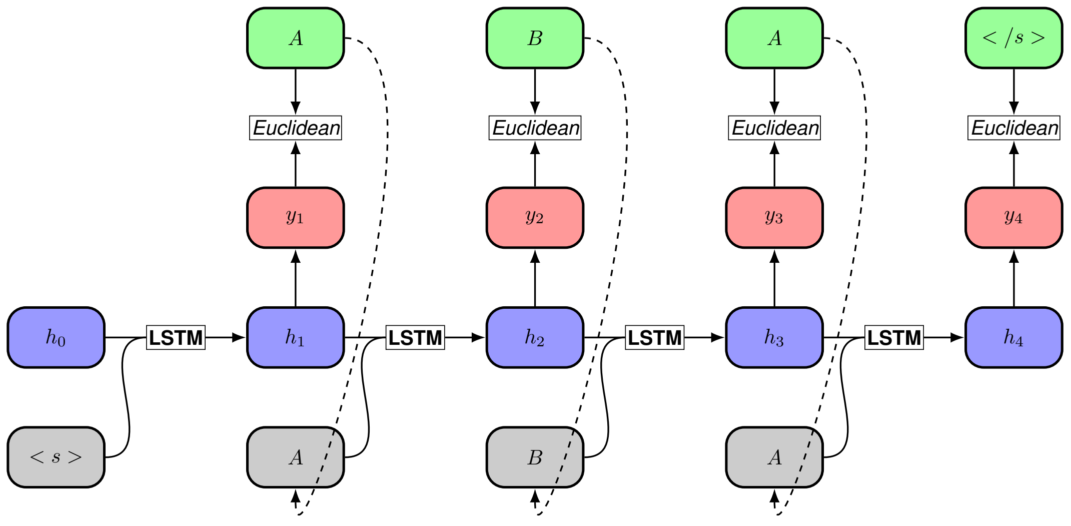


Fig. S2. A recursive LSTM network producing ABA sequences.

The input is a sequence of vectors $x_1, x_2, x_3, \dots$, 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:

$$\max(\theta) \qquad -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T_i}\left\|h_\theta\left(x^{i,0:t-1}\right) - x^{i,t}\right\|^2 \tag{6}$$

for N examples. Here, T_i is the length of example i. As before, θ abbreviates the parameters of the model as specified in (4)–(5). 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 S2 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 $t + 1$.

C. Model 3a: A single layer feed forward network . 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 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 3. SI 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.

D. Model 3b: A deeper feed-forward network for hierarchical same–different. This model extends (1)–(2) 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 $[a;b;c;d]$:

$$h_1 = \text{ReLU}([a;b;c;d]W_{xh} + b_{h_1}) \quad [7]$$
$$h_2 = \text{ReLU}(h_1 W_{hh} + b_{h_2}) \quad [8]$$
$$y = \textbf{softmax}(h_2 W_{hy} + b_y) \quad [9]$$

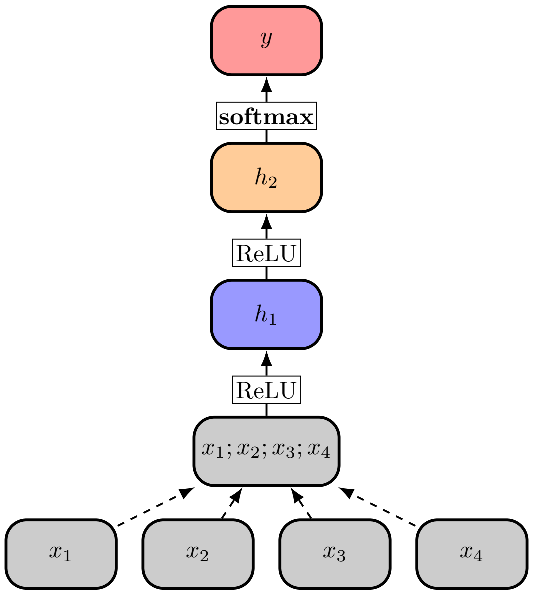


Fig. S3. A two layer network computing hierarchical equality.

The objective function is again defined using a cross entropy function, as in equation 3.

Figure S3 depicts this model.

E. Model 3c: Pretraining for hierarchical same–different.

Our pretraining model is as follows:

$$h_1 = \text{ReLU}([a;b]W_{xh} + b_h) \quad [10]$$
$$h_2 = \text{ReLU}([c;d]W_{xh} + b_h) \quad [11]$$
$$h_3 = \text{ReLU}([h_1;h_2]W_{xh} + b_h) \quad [12]$$
$$y = \textbf{softmax}(h_3 W_{hy} + b_y) \quad [13]$$

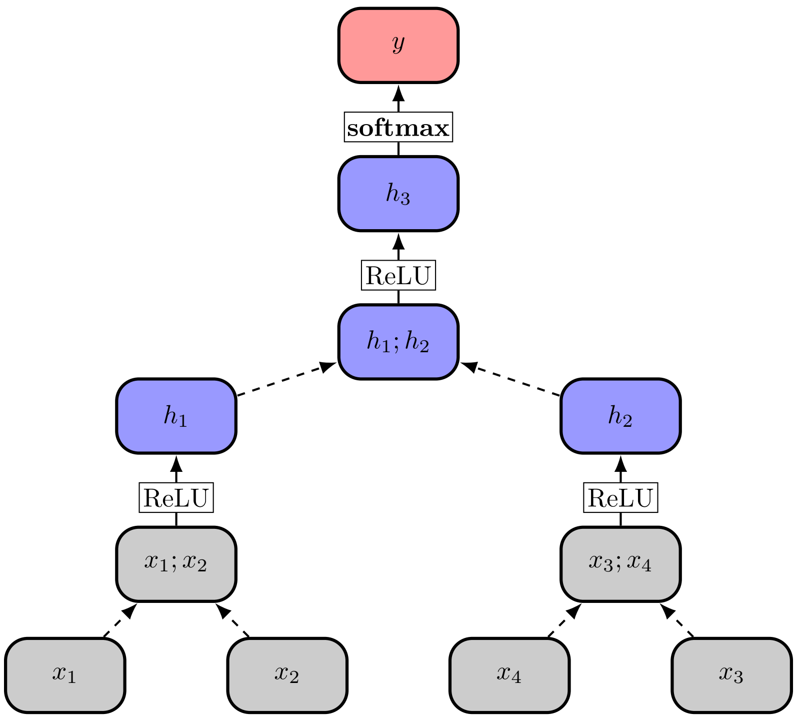


Fig. S4. A single layer network pretrained on equality computing hierarchical equality.

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.

The objective function is again defined using a cross entropy function, as in equation 3.

2. 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:

$$h_i = \text{ReLU}\left(E[i]W_{xh} + b_h\right) \qquad [14]$$

$$y_{i,j} = \textbf{softmax}(h^i W^j_{hy} + b^j_y) \qquad [15]$$

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:

$$\max(\theta) \quad \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{J}\sum_{k=1}^{K_j} y^{i,j,k}\log\left(h_\theta(i)^{j,k}\right) \qquad [16]$$

where $y^{i,j,k}$ is the correct label for example i in task j and $h_\theta(i)^{j,k}$ 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 epochs where each epoch consists of an example for each item in the vocabulary. This choice is motivated primarily by computational costs; additional pretraining epochs greatly increase experiment run-times, though they do have the potential to imbue the representations with even more useful structure. Pretraining with the optimal hyperparameter settings always led to perfect accuracy on the pretraining tasks.

3. 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 (4) were used. For all models, we used a batch size of 1 and ran a hyperparameter search 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. We considered hidden dimensions of {2, 10, 25, 50, 100} and input dimensions of {2, 10, 25, 50, 100}. In the main text, we graph results for the single best hyperparameter setting. Later in the supplemental material under the heading 'Additional results plots', we show how model performance is affected by changes in hidden dimensionality and input dimensionality.

4. 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.

Property 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.

Property representations with analog features do not inhibit generalization in the same way. If height is represented as an analog 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.

5. 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 S5 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.

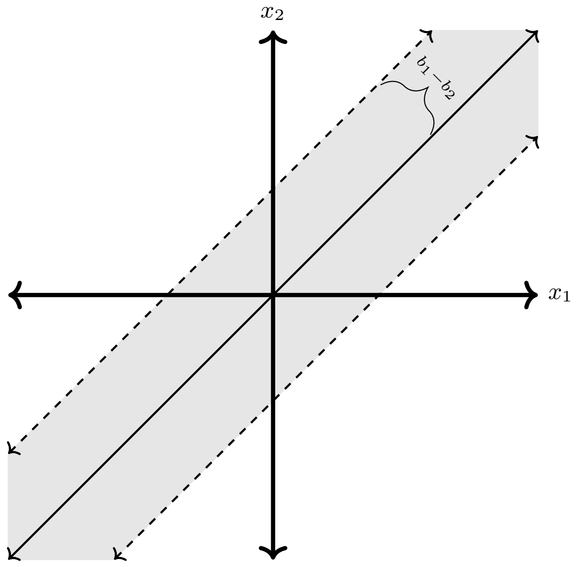


Fig. S5. A visual representation of how the analytic solution to identity of a single layer feed forward network changes depending on the values of two bias terms, b_1, b_2.

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.

6. 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

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

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:

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

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.

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

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

$$f_2 = \mathbf{1}_d$$

$$\begin{aligned}
o_2 &= \sigma(x_2 \mathbf{0}_{n\times d} + h_1 \begin{bmatrix} 1\ldots 1 \\ \mathbf{0}_{2n\times n} \end{bmatrix} + \mathbf{0}_d) \\
&= \sigma(\mathbf{0.5}_d \circ \mathrm{ReLU}\left(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix}\right) \begin{bmatrix} 1\ldots 1 \\ \mathbf{0}_{2n\times n} \end{bmatrix} + \mathbf{0}_d) \\
&= \sigma(\mathbf{0.5}_d \circ \mathbf{N}_d) \approx \mathbf{1}_d
\end{aligned}$$

$$\begin{aligned}
i_2 &= \sigma(x_2 \mathbf{0}_{n\times d} + h_1 \begin{bmatrix} -4\cdots -4 \\ \mathbf{0}_{2n\times n} \end{bmatrix} + \mathbf{N}_d) \\
&= \sigma(\mathbf{0.5}_d \circ \mathrm{ReLU}\left(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix}\right) \begin{bmatrix} -4\cdots -4 \\ \mathbf{0}_{2n\times n} \end{bmatrix} + \mathbf{N}_d) \\
&= \sigma(\mathbf{0.5}_d \circ \textbf{-4}\mathbf{N}_d + \mathbf{N}_d) \approx \mathbf{0}_d
\end{aligned}$$

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.

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

$$\begin{aligned}
h_2 &= \mathbf{1}_d \circ \mathrm{ReLU}\left(\mathrm{ReLU}(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix})\right) \\
&= \mathrm{ReLU}\left(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix}\right)
\end{aligned}$$

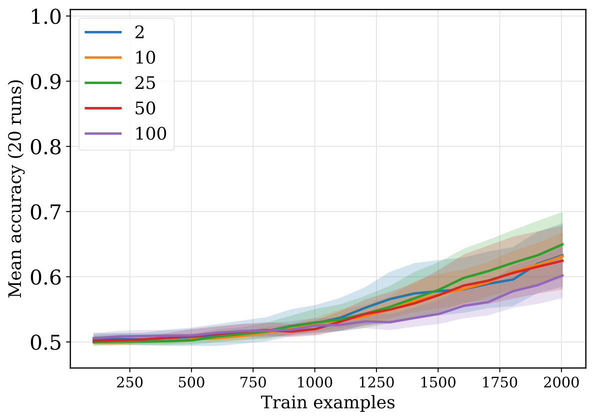
With the hidden states for the first and second time steps, we can compute the output values and find that the output at the first time step is the initial input vector scaled by one half and the output at the second time step is the initial input vector.

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

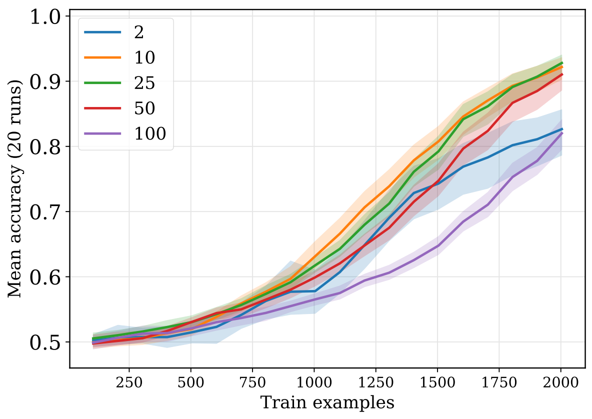
Then, because $y_1 = \mathbf{0.5}_d \circ x_1 \neq x_1$ and $y_2 = x_1$, this network produces ABA sequences.

7. Additional results plots

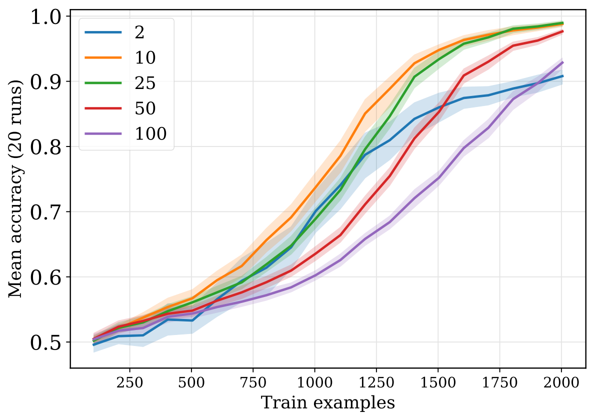
A. Model 1 for basic same–different. Figure S6 explores a wider range of hidden dimensionalities for Model 1 applied to the basic same–differerent task. The lines correspond to different embedding dimensionalities.



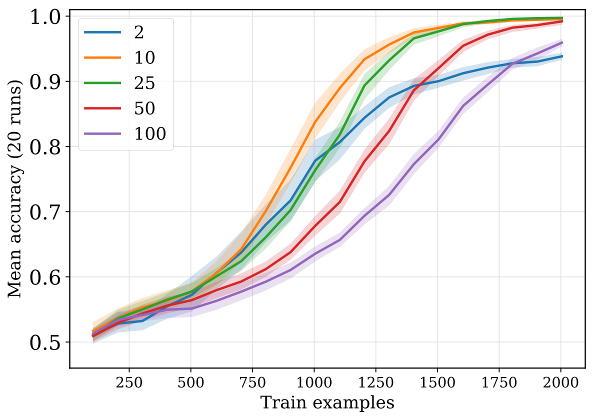
(a) Hidden dimensionality 2.



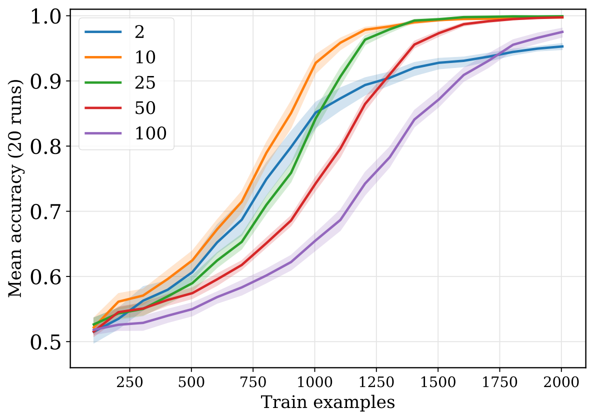
(b) Hidden dimensionality 10.



(c) Hidden dimensionality 25.



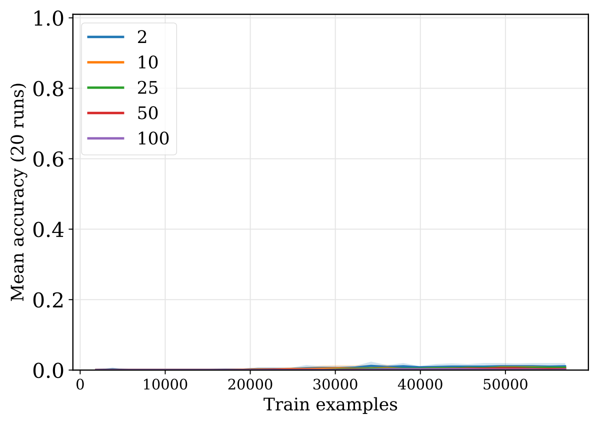
(d) Hidden dimensionality 50.



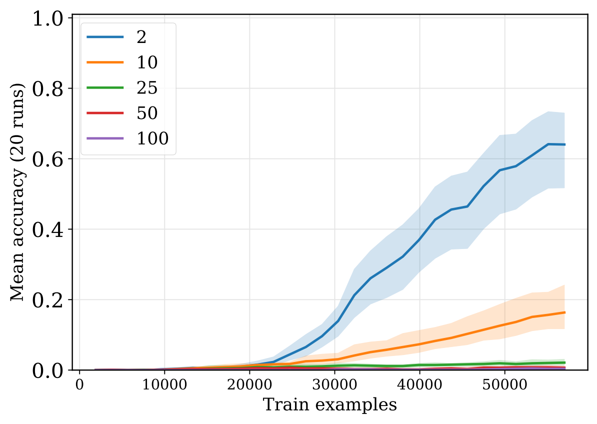
(e) Hidden dimensionality 100.

Fig. S6. Results for Model 1 for basic same–different. Lines correspond to different input dimensions. These models were provided random input representations.

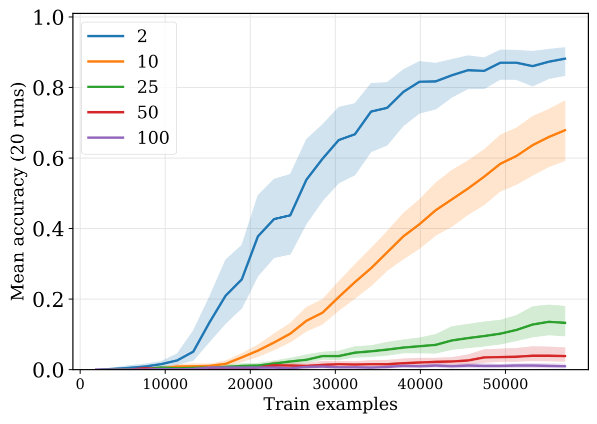
B. Model 2 for sequential same–different. Figure S7 explores a wider range of hidden dimensionalities for Model 2 applied to the sequential ABA task. The lines correspond to different embedding dimensionalities. The full training set is presented to the model in multiple epochs.



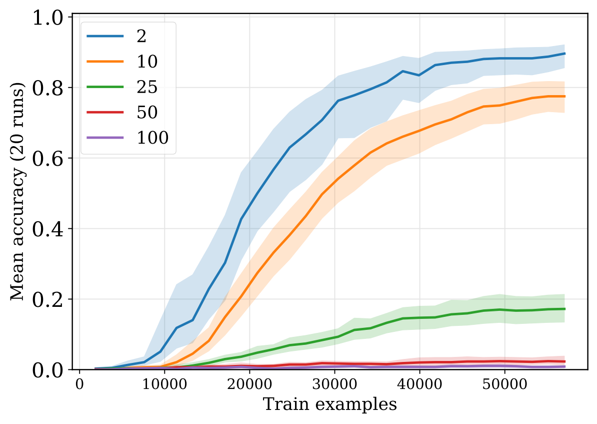
(a) Hidden dimensionality 2.



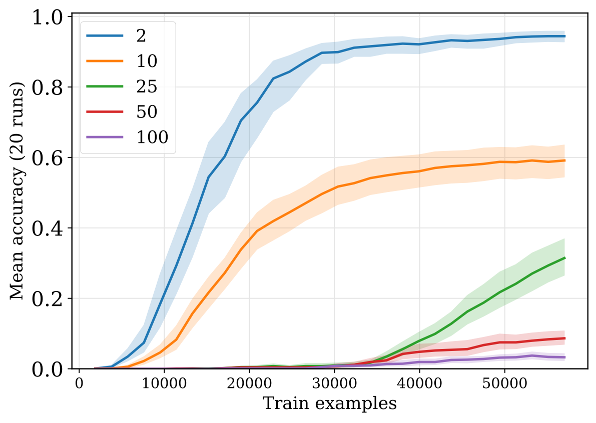
(b) Hidden dimensionality 10.



(c) Hidden dimensionality 25.



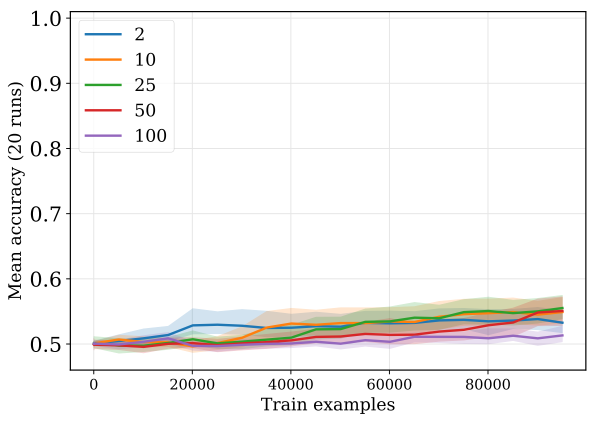
(d) Hidden dimensionality 50.



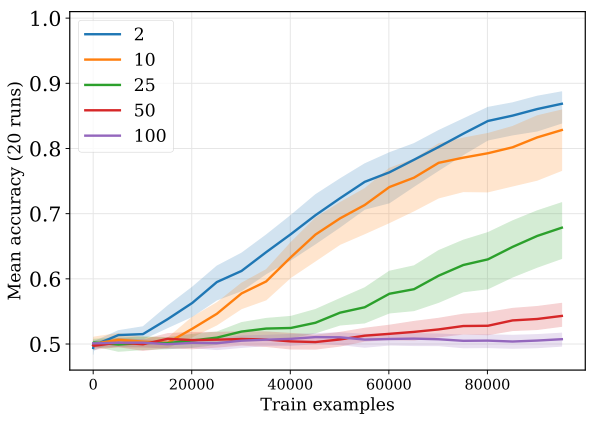
(e) Hidden dimensionality 100.

Fig. S7. Results for Model 2 for basic same–different, with a vocabulary size of 20. Lines correspond to different input dimensions. These models were provided random input representations.

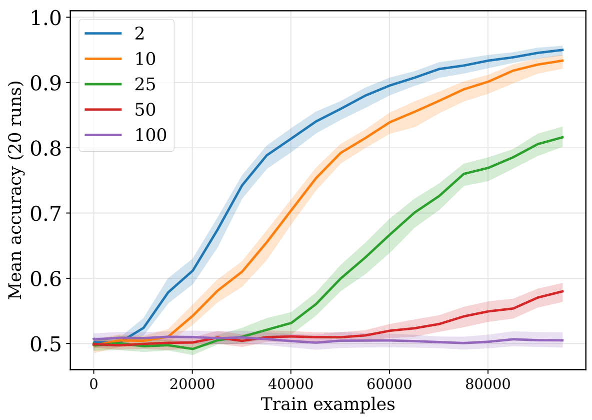
C. Model 3a for hierarchical same–different. Figure S8 shows the results of applying the model in (1)–(2) to the hierarchical same–different task. The lines correspond to different embedding dimensionalities. The only change from that model is that the inputs have dimensionality $4m$, since the four distinct representations in task inputs are simply concatenated.



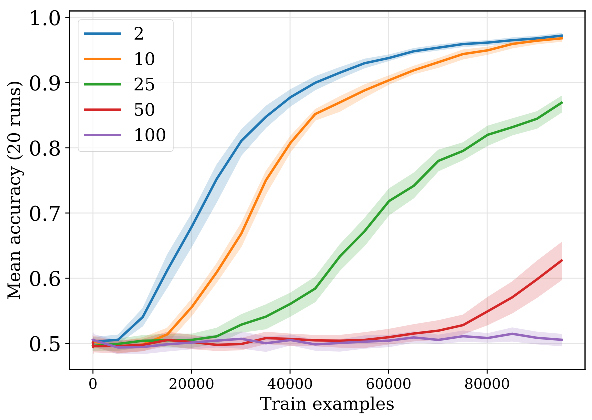
(a) Hidden dimensionality 2.



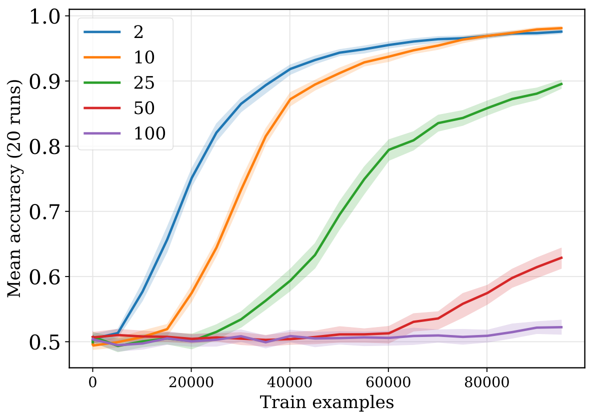
(b) Hidden dimensionality 10.



(c) Hidden dimensionality 25.



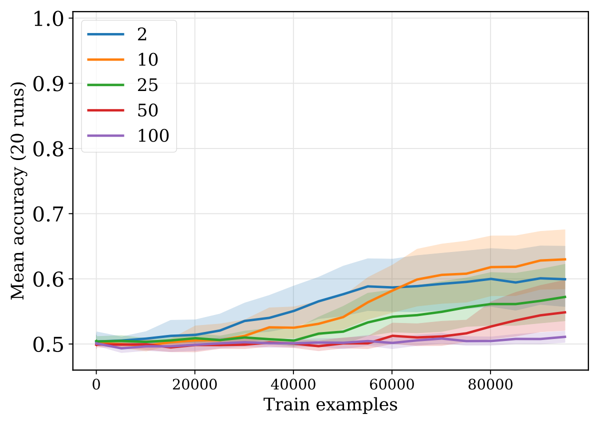
(d) Hidden dimensionality 50.



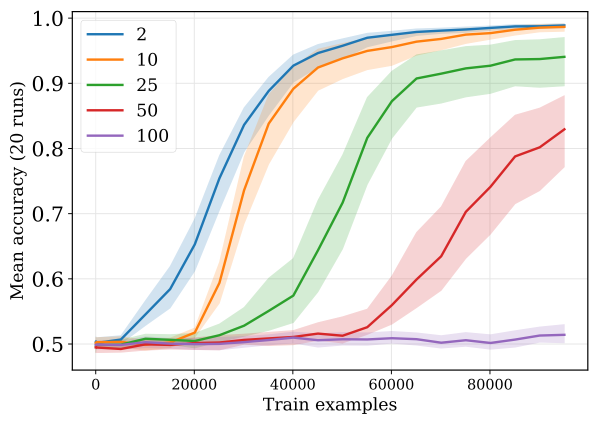
(e) Hidden dimensionality 100.

Fig. S8. Results for Model 1 applied to the hierarchical same–different task. Lines correspond to different input dimensions. These models were provided random input representations.

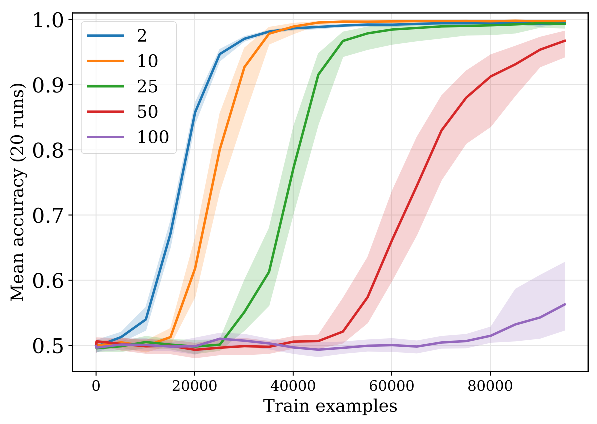
D. Model 3b for hierarchical same–different. Figure S9 shows the results of applying the model in (7)–(9) to the hierarchical same–different task. The lines correspond to different embedding dimensionalities.



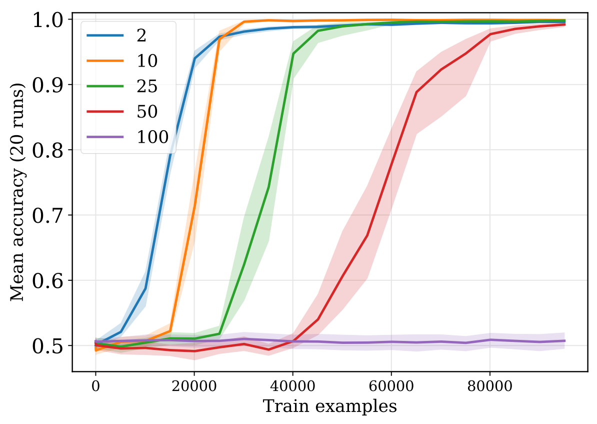
(a) Hidden dimensionality 2.



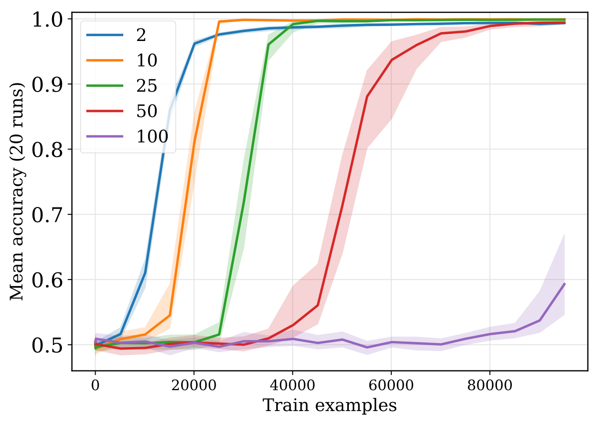
(b) Hidden dimensionality 10.



(c) Hidden dimensionality 25.



(d) Hidden dimensionality 50.



(e) Hidden dimensionality 100.

Fig. S9. Results for Model 3a applied to the hierarchical same–different task. Lines correspond to different input dimensions. These models were provided random input representations.

References

1. Cybenko G (1989) Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* 2(4):303–314.
2. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. *Neural Networks* 2(5):359–366.
3. Hochreiter S, Schmidhuber J (1997) Long short-term memory. *Neural computation* 9(8):1735–1780.
4. Kingma DP, Ba J (2014) Adam: A method for stochastic optimization in *Proceedings of the 3rd International Conference on Learning Representations.*