Supplementary materials for 'Relational reasoning and generalization using non-symbolic neural networks'

Anonymous CogSci 2020 submission

Contents

1 Model figures

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

Our model of equality is given by (1)–(2):

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

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 $\mathbf{softmax}(x)_i = \frac{\exp x_i}{\sum_j \exp x_j}$.

Fig. 1 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).

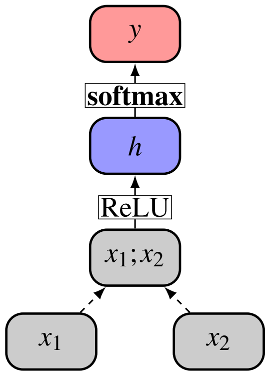


Figure 1: A single layer network computing equality.

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

The specific model we use for this is as follows:

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

for $t > 0$, with $h_0 = \mathbf{0}$. **LSTM** is a long short-term memory cell. Fig. 2 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 cosine similarity) is taken to be the prediction. This vector is then used as the input for timestep $t+1$.

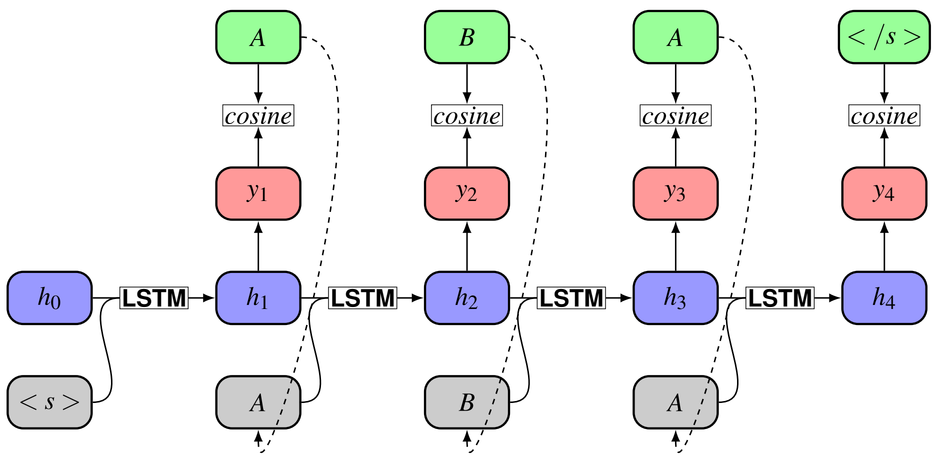


Figure 2: A recursive LSTM network producing ABA sequences.

1.3 Model 3a: 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 $[a;b;c;d]$.

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

Fig. 3 depicts this model.

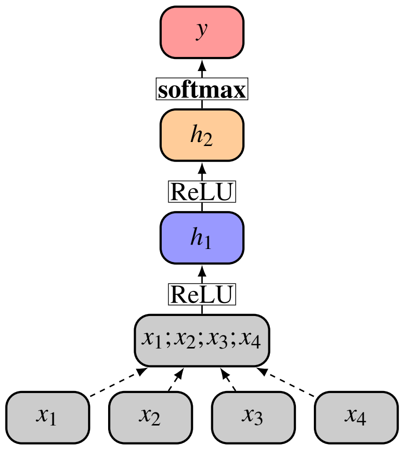


Figure 3: A two layer network computing hierarchical equality.

1.4 Model 3b: Pretraining for hierarchical same–different

Our pretraining model is as follows:

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

where W_{xh}, W_{hy}, b_h, and b_y are the parameters from the model in (1)–(2) already trained on basic equality. Fig. 4 depicts this model. The colors indicate where parameters are reused by the model.

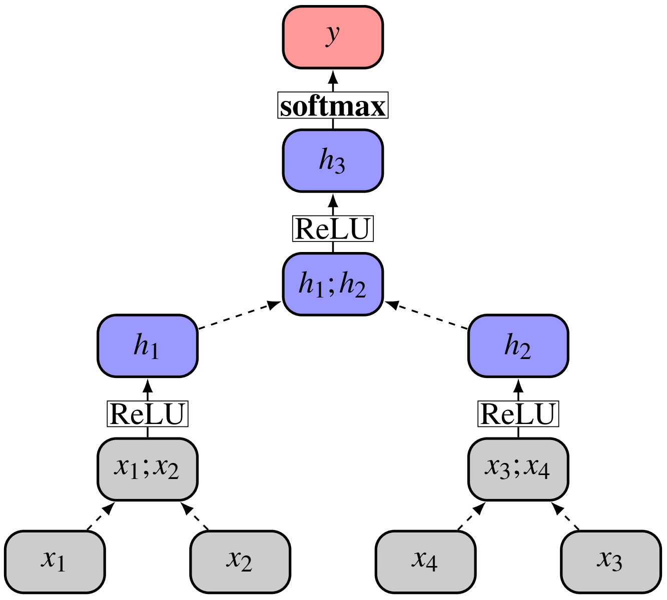


Figure 4: A single layer network pretrained on equality computing hierarchical equality.

2 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 network pretrained on basic equality and then used for hierarchical equality was implemented using TensorFlow and a cross entropy function was used to compute the prediction error. For all models, Adam optimizers 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.

3 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. A 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 behavior is 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. 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 resulting 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.

4 An analytic solution to identity with a feed forward network

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 and if the output o_2 is larger that o_1 then the other class is predicted. We 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, 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 that are left unspecified, b_1, b_2. We present a visualization in Figure 5 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, and 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.

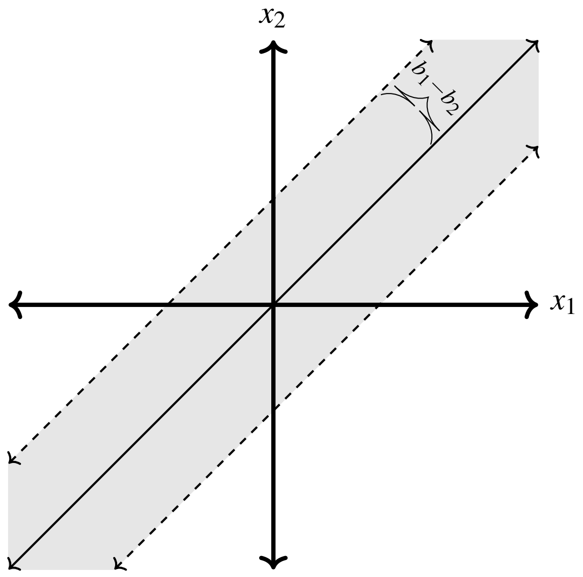


Figure 5: A visual representation of how the analytic solution to identity of a single layer feed forward network changes depending on the values of two bias terms, b_1, b_2.

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 boundary the network's predictions get closer and closer to the solution line, but 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; 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 you can get. 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 $(x,x+\varepsilon)$ which is not on the solution line $x_1 = x_2$. Then 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 can not be achieved. Then we can conclude our arbitrarily good solution is the best we can do.

5 An analytic solution to ABA sequences

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

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

where if n is the representation size and d is the network hidden dimension then $x_t \in \mathbb{R}^n, f_t, i_t, o_t, h_t, c_t \in \mathbb{R}^d, W \in \mathbb{R}^{n\times d}, U \in \mathbb{R}^{d\times d}, V \in \mathbb{R}^{d\times n}, b \in \mathbb{R}^d$ 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:

$$\begin{aligned}
f_t &= \sigma(x_t \mathbf{0}_{n\times d} + h_{t-1}\mathbf{0}_{d\times d} + \mathbf{N}_d)\\
i_t &= \sigma(x_t \mathbf{0}_{n\times d} + h_{t-1}\begin{bmatrix} -4\cdots -4\\ \mathbf{0}_{2n\times n}\end{bmatrix} + \mathbf{N}_d)\\
o_t &= \sigma(x_t \mathbf{0}_{n\times d} + h_{t-1}\begin{bmatrix} 1\dots 1\\ \mathbf{0}_{2n\times n}\end{bmatrix} + \mathbf{0}_d)\\
c_t &= f_t \circ c_{t-1} + i_t \circ \texttt{ReLu}(x_t \begin{bmatrix} 0 & & \\ \vdots & -I_{n\times n} & I_{n\times n}\\ 0 & & \end{bmatrix} + h_{t-1}\mathbf{0}_{d\times d} + \begin{bmatrix} N & 0\dots 0\end{bmatrix})
\end{aligned}$$

$$h_t = \texttt{ReLu}(o_t \circ (c_t))$$

$$y = h_t \begin{bmatrix} 0\ldots0 \\ -I_{n\times n} \\ I_{n\times n} \end{bmatrix}$$

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

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

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

$$f_1 = \sigma(\mathbf{N}_d) \approx \mathbf{1}_d \qquad o_1 = \sigma(\mathbf{0}_d) = \mathbf{0.5}_d \qquad i_1 = \sigma(\mathbf{N}_d) \approx \mathbf{1}_d$$

Then we compute the cell and hidden states at the first time step. 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.

$$c_1 = \mathbf{N}_d \circ \mathbf{0}_d + \mathbf{1}_d \circ \texttt{ReLu}(x_1 \begin{bmatrix} 0 & & \\ \vdots & -I_{n\times n} & I_{n\times n} \\ 0 & & \end{bmatrix} + \begin{bmatrix} N & 0\ldots0 \end{bmatrix}) = \texttt{ReLu}(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix})$$

$$h_1 = \mathbf{0.5}_d \texttt{ReLu}(\texttt{ReLu}(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix})) = \mathbf{0.5}_d \circ \texttt{ReLu}(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix})$$

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.

$$f_2 = \mathbf{N}_d \qquad o_2 = \sigma(y_1 \mathbf{0}_{n\times d} + h_1 \begin{bmatrix} 1\ldots1 \\ \mathbf{0}_{2n\times n} \end{bmatrix} + \mathbf{0}_d) =$$

$$\sigma(\mathbf{0.5}_d \circ \texttt{ReLu}(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix}) \begin{bmatrix} 1\ldots1 \\ \mathbf{0}_{2n\times n} \end{bmatrix} + \mathbf{0}_d) = \sigma(\mathbf{0.5}_d \circ \mathbf{N}_d) \approx \mathbf{1}_d$$

$$i_2 = \sigma(y_1 \mathbf{0}_{n\times d} + h_1 \begin{bmatrix} -4\cdots-4 \\ \mathbf{0}_{2n\times n} \end{bmatrix} + \mathbf{N}_d) =$$

$$\sigma(\mathbf{0.5}_d \circ \texttt{ReLu}(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix}) \begin{bmatrix} -4\cdots-4 \\ \mathbf{0}_{2n\times n} \end{bmatrix} + \mathbf{N}_d) = \sigma(\mathbf{0.5}_d \circ \textbf{-4}\mathbf{N}_d + \mathbf{N}_d) \approx \mathbf{0}_d$$

Then we compute the cell and hidden states for the second time step. 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.

$$c_2 = \mathbf{1}_d \circ \texttt{ReLu}(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix}) + \mathbf{0}_d \circ \texttt{ReLu}(x_2 \begin{bmatrix} 0 & & \\ \vdots & -I_{n\times n} & I_{n\times n} \\ 0 & & \end{bmatrix} + \begin{bmatrix} N & 0\ldots0 \end{bmatrix})$$
$$= \texttt{ReLu}(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix})$$

$$h_2 = \mathbf{1}_d \circ \texttt{ReLu}(\texttt{ReLu}(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix})) = \texttt{ReLu}(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix})$$

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.

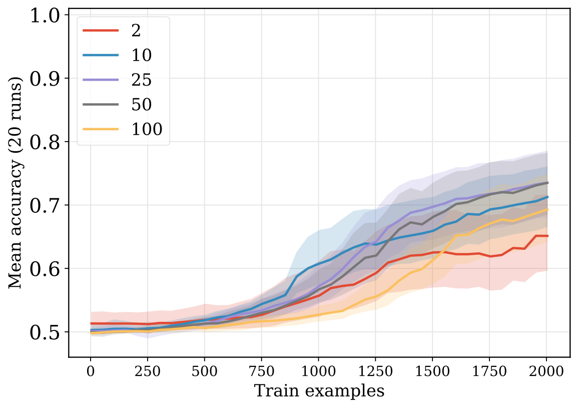
$$y_1 = h_1 \begin{bmatrix} 0\ldots0 \\ -I_{n\times n} \\ I_{n\times n} \end{bmatrix} = \mathbf{0.5}_d \circ \texttt{ReLu}(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix}) \begin{bmatrix} 0\ldots0 \\ -I_{n\times n} \\ I_{n\times n} \end{bmatrix} = \mathbf{0.5}_d \circ x_1$$
$$y_2 = h_2 \begin{bmatrix} 0\ldots0 \\ -I_{n\times n} \\ I_{n\times n} \end{bmatrix} = \texttt{ReLu}(\begin{bmatrix} N & -x_1 & x_1 \end{bmatrix}) \begin{bmatrix} 0\ldots0 \\ -I_{n\times n} \\ I_{n\times n} \end{bmatrix} = x_1$$

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

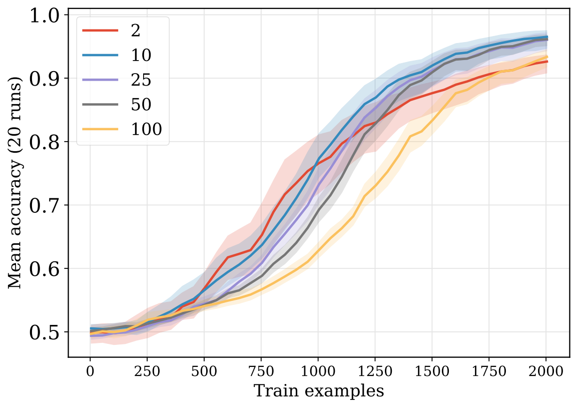
6 Additional results plots

6.1 Model 1 for basic same–different

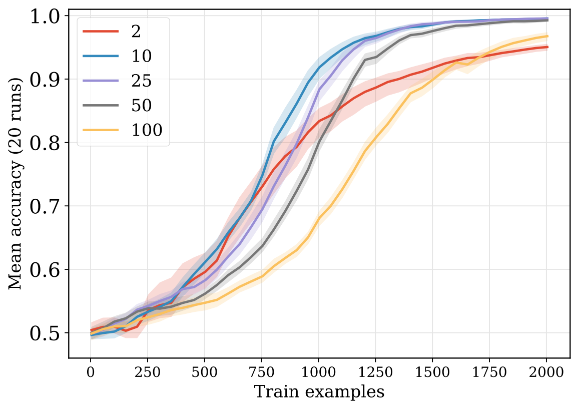
Fig. 6 explores a wider range of hidden dimensionalities for Model 1 applied to the basic same–differerent task. As in the paper, the lines correspond to different embedding dimensionalities.



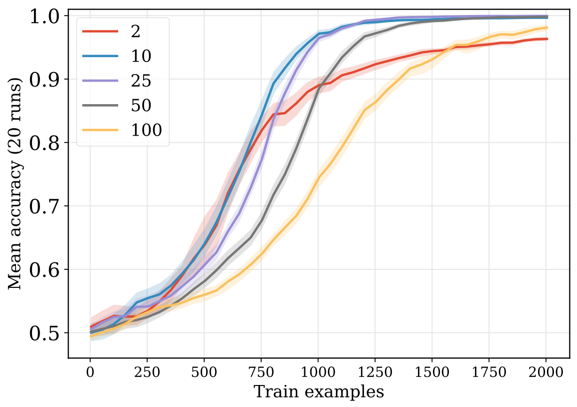
(a) Hidden dimensionality 2.



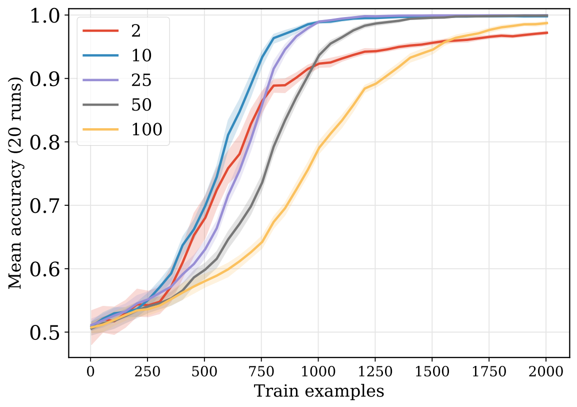
(b) Hidden dimensionality 10.



(c) Hidden dimensionality 25.



(d) Hidden dimensionality 50.



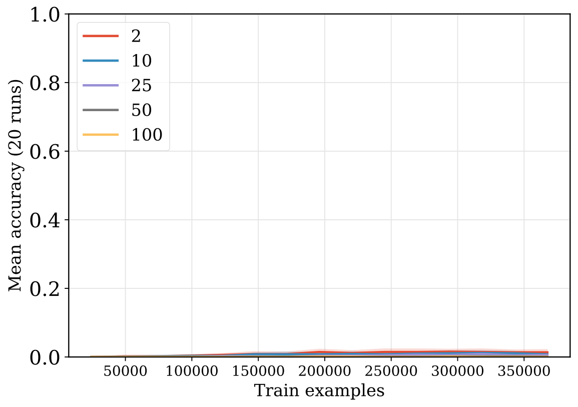
(e) Hidden dimensionality 100.

Figure 6: Results for Model 1 for basic same–different.

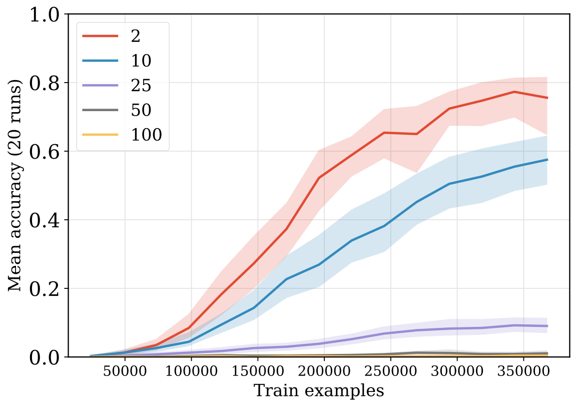
6.2 Model 2 for sequential same–different

Fig. 7–Fig. 9 explore 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. Here, we report results for three different vocabulary sizes. The full training set is presented to the model in multiple epochs.

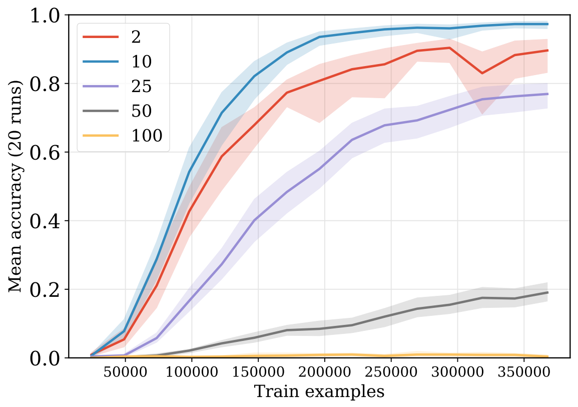
6.2.1 Vocabulary size: 50 (2,450 training examples)



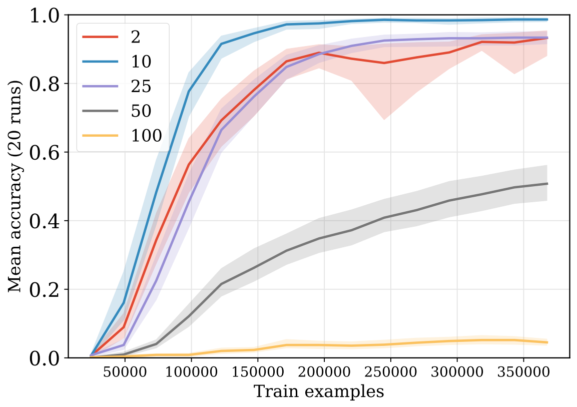
(a) Hidden dimensionality 2.



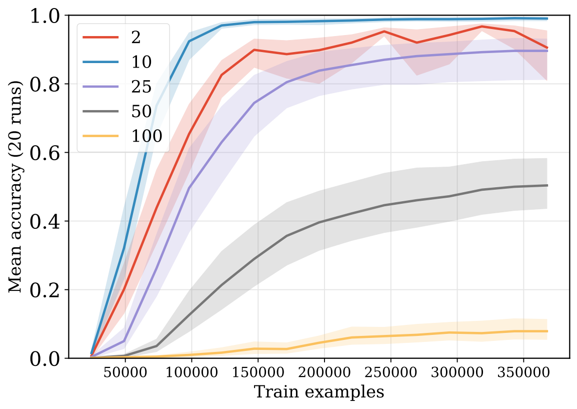
(b) Hidden dimensionality 10.



(c) Hidden dimensionality 25.



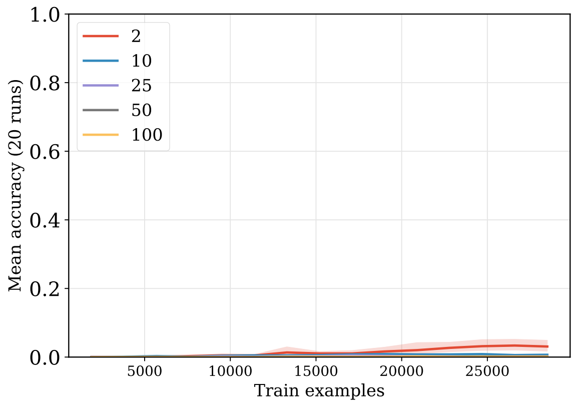
(d) Hidden dimensionality 50.



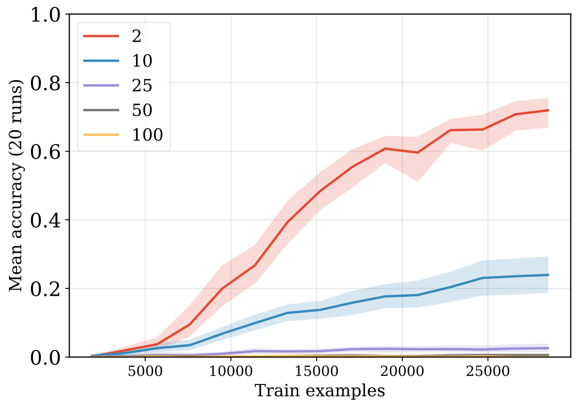
(e) Hidden dimensionality 100.

Figure 7: Results for Model 2 for basic same–different, with a vocabulary size of 50.

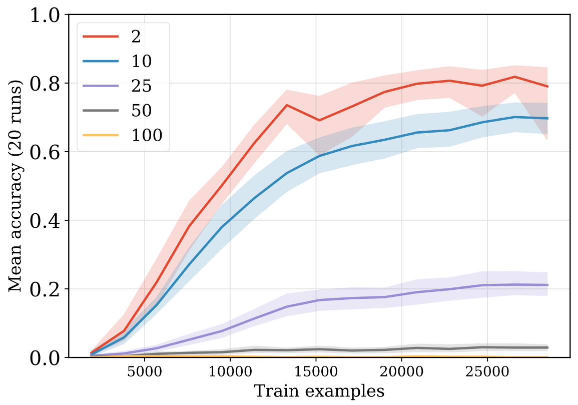
6.2.2 Vocabulary size: 20 (380 training examples)



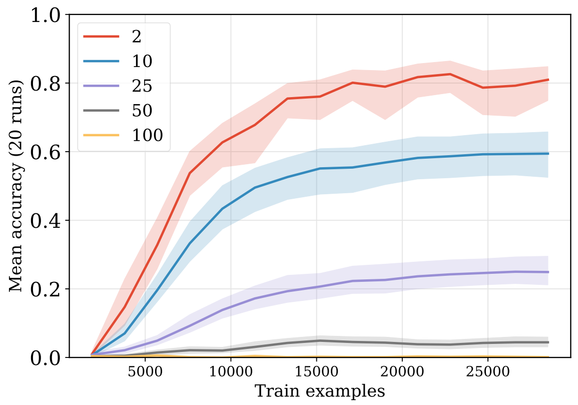
(a) Hidden dimensionality 2.



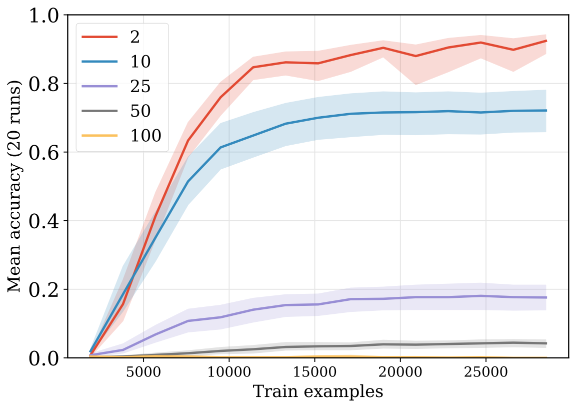
(b) Hidden dimensionality 10.



(c) Hidden dimensionality 25.



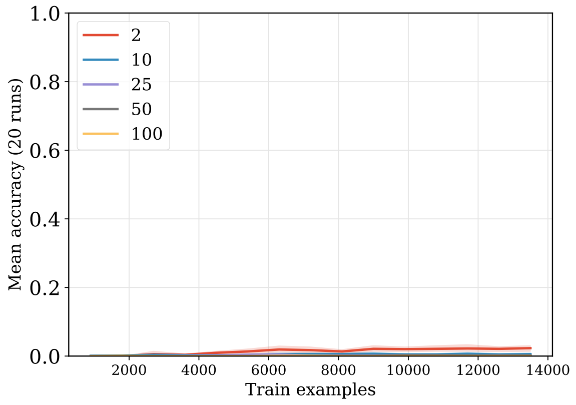
(d) Hidden dimensionality 50.



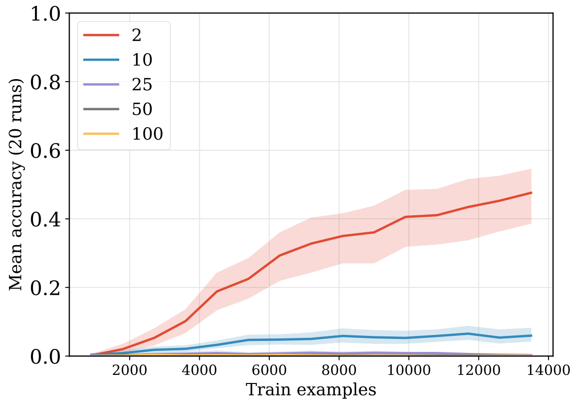
(e) Hidden dimensionality 100.

Figure 8: Results for Model 2 for basic same–different, with a vocabulary size of 20 (380 training examples per epoch)

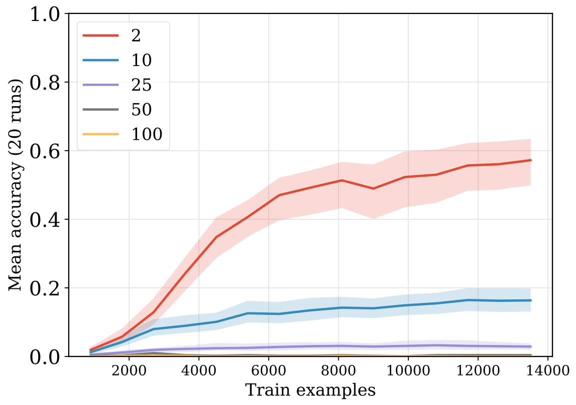
6.2.3 Vocabulary size: 10 (90 training examples)



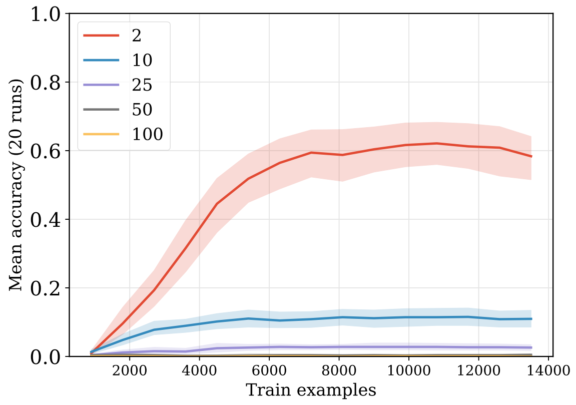
(a) Hidden dimensionality 2.



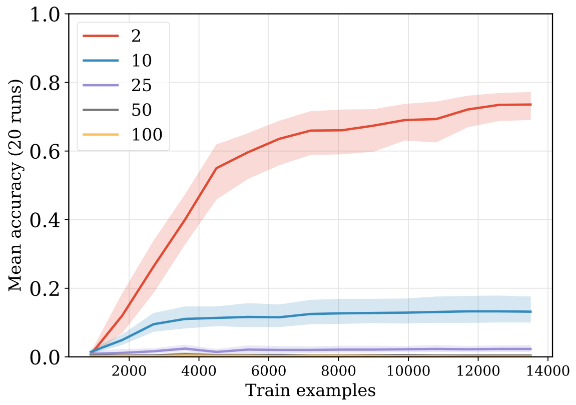
(b) Hidden dimensionality 10.



(c) Hidden dimensionality 25.



(d) Hidden dimensionality 50.

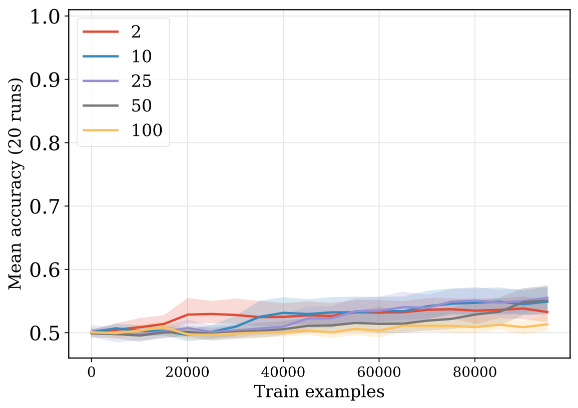


(e) Hidden dimensionality 100.

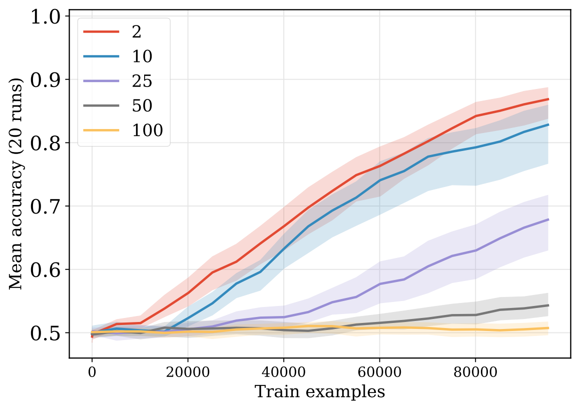
Figure 9: Results for Model 2 for basic same–different, with a vocabulary size of 10.

6.3 Model 1 for hierarchical same–different

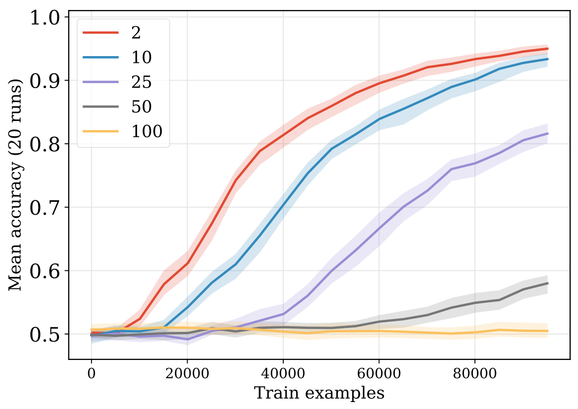
Fig. 10 shows the results of applying the model in (1)–(2) to the hierarchical same–different task. The only change from that model is that the inputs have dimensionality $4m$, since the four distinct representations in task inputs are simply concatenated.



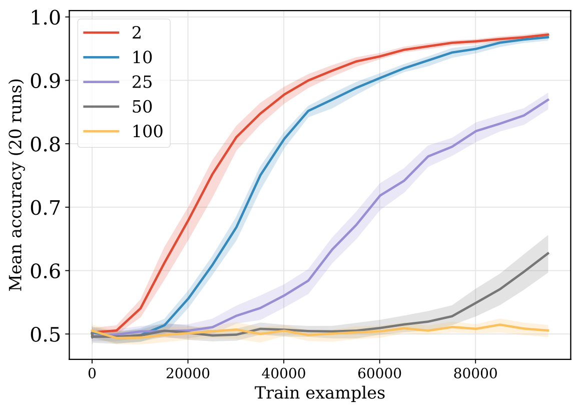
(a) Hidden dimensionality 2.



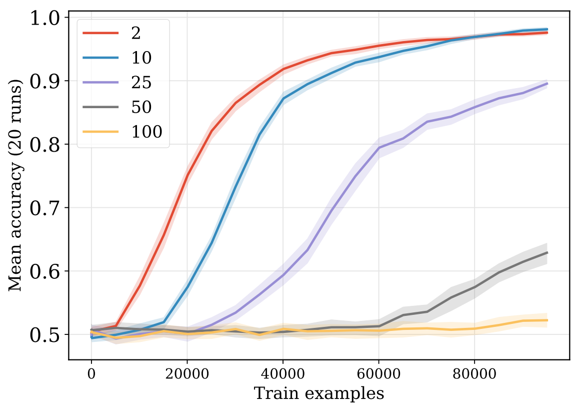
(b) Hidden dimensionality 10.



(c) Hidden dimensionality 25.



(d) Hidden dimensionality 50.

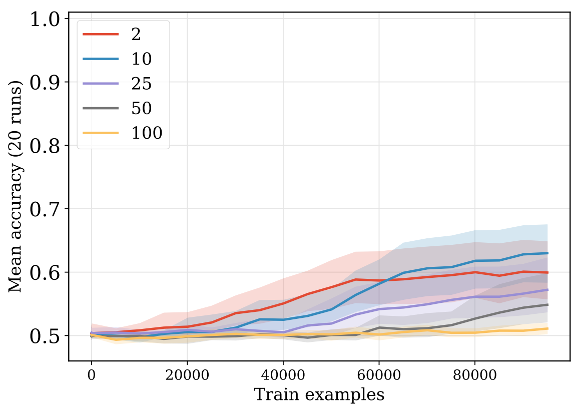


(e) Hidden dimensionality 100.

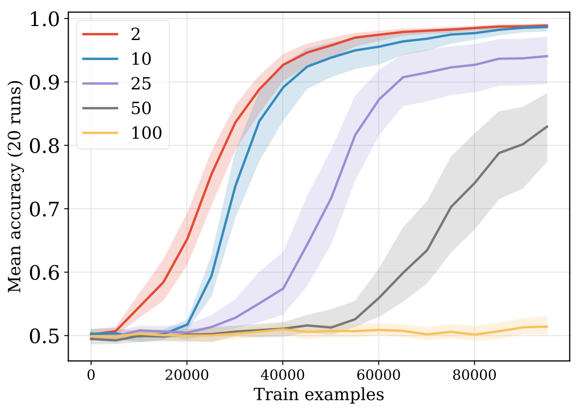
Figure 10: Results for Model 1 applied to the hierarchical same–different task.

6.4 Model 3a for hierarchical same–different

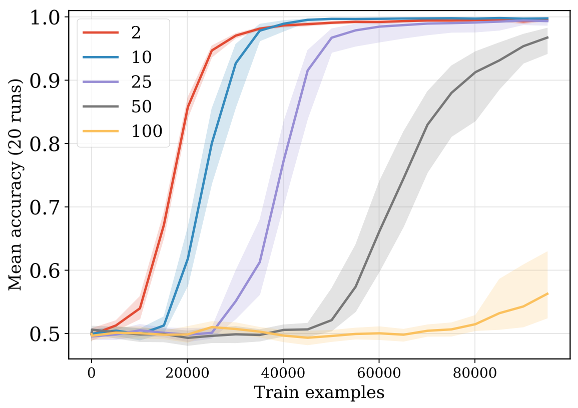
Fig. 11 shows the results of applying the model in (5)–(7) to the hierarchical same–different task.



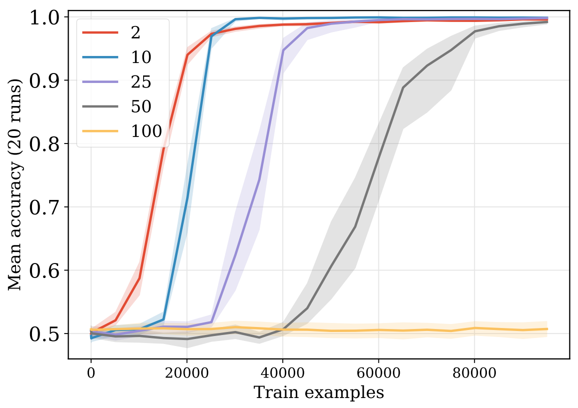
(a) Hidden dimensionality 2.



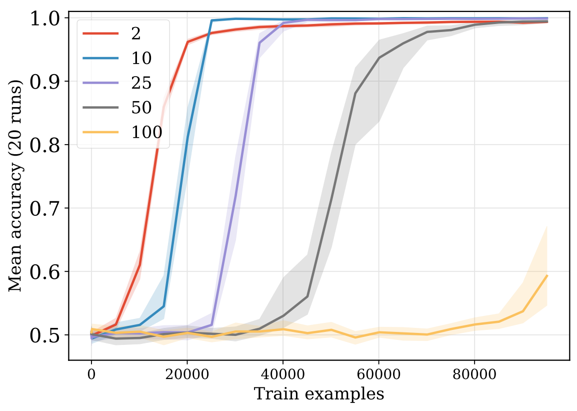
(b) Hidden dimensionality 10.



(c) Hidden dimensionality 25.



(d) Hidden dimensionality 50.



(e) Hidden dimensionality 100.

Figure 11: Results for Model 3a applied to the hierarchical same–different task.

6.5 Model 3b for hierarchical same–different

Fig. 12 shows the results of applying the model in (8)–(11) to the hierarchical same–different task. For this model, the embedding and hidden dimensionalities must be the same.

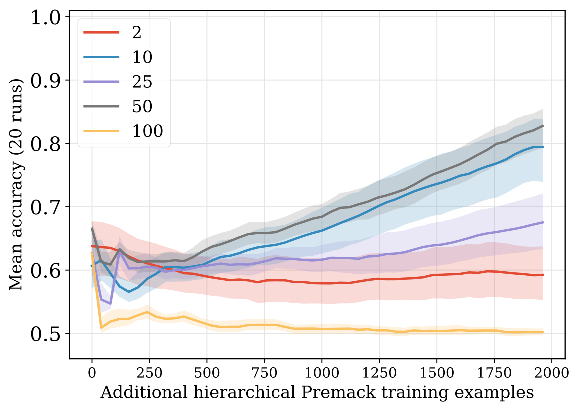


Figure 12: Results for Model 3b applied to the hierarchical same–different task.