Supplemental Material for Benchmarks, Algorithms, and Metrics for Hierarchical Disentanglement

Andrew Slavin Ross [1] Finale Doshi-Velez [1]

A. Appendix

A.1. Training and Architecture Details

For Chopsticks, our encoders and decoders used two hidden layers of width 256, and our loss function $\mathcal{L}_x$ was defined as a zero-centered Gaussian negative log likelihood with $\sigma = 0.1$. For Spaceshapes, encoders and decoders used the 7-layer convolutional architecture from Burgess et al. (2018), and our loss function $\mathcal{L}_x$ was Bernoulli negative log likelihood. All models were implemented in Tensorflow; code is available at `https://github.com/dtak/hierarchical-disentanglement`.

For both models, the assignment loss $\mathcal{L}_a$ was set to mean-squared error, but only for assignments that were defined. This was implemented by setting undefined assignment components to -1, and then defining $\mathcal{L}_a(a, a') = \sum_i \mathbb{1}[a'_i \geq 0](a_i - a'_i)^2$.

All activation functions were set to ReLU ($\max(0, x)$) or Softplus ($\ln(1+e^x)$), e.g. for the initial smooth autoencoder, which was also trained with dimensionality equal to one plus the maximum intrinsic dimensionality of the dataset. We investigate varying this parameter in Fig. A.5 and find it can be much larger, and perhaps would have produced better results (though nearest neighbor calculation and local SVD computations would have been slower).

All models were trained for 50 epochs with a batch size of 256 on a dataset of size 100,000, split 90%/10% into train/test. We used the Adam optimizer with a learning rate starting at 0.001 and decaying by $\frac{1}{10}$ halfway and three-quarters of the way through training.

For COFHAE, we selected softmax temperature τ, the assignment penalty strength λ_1, and the adversarial penalty strength λ_2 based on *training set* reconstruction error and MIMOSA assignment accuracy. Splitting off a separate validation set was not necessary, as the most common problem we faced was poor convergence, not overfitting; the adversarial penalty would dominate and prevent the procedure from learning a model that could reconstruct X or A.

Specifically, for each restart, we ran COFHAE with τ in $\{\frac{1}{2}, \frac{2}{3}, 1\}$, λ_1 in $\{10, 100, 1000\}$, and λ_2 in $\{1, 10, 100\}$. We then selected the model with the lowest training MSE $\sum_n ||x_n - x'_n||_2^2$, but restricting ourselves to the 33.3% of models with the lowest assignment loss $\sum_n \mathcal{L}_a(a_n, a'_n)$.

For evaluating R^4 and R^4_c, we used gradient boosted decision trees, which were faster to train than neural networks.

A.2. Additional Chopsticks Details

In this section, we clarify the generative process behind the different variants of Chopsticks, and discuss alternatives.

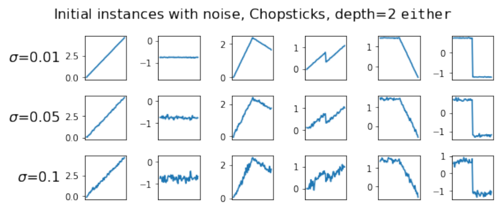
Chopsticks can be generated by the following Python code (the exact code we used is slightly different due to the need to save ground-truth factors):

```
def Bern(p):
  return int(np.random.uniform() < p)

def Unif(a,b):
  return a + np.random.uniform() * (b-a)

def stick_segment(variant, T):
  slope = Unif(-0.01, 0.01) * np.arange(T)
  inter = Unif( -0.2,  0.2) + np.zeros(T)
  if   variant == 'slope': return slope
  elif variant == 'inter': return inter
  elif variant == 'both': return slope+inter
  elif variant == 'either':
    return slope if Bern(0.5) else inter

def chopsticks(depth, variant, T=64):
  stick = stick_segment(variant, T)
  chop = Bern(1-np.power(2.0,-(depth-1)))
  if chop:
    stick2 = chopsticks(depth-1, variant, T//2)
    stick[T//2:] += stick2
  return stick
```

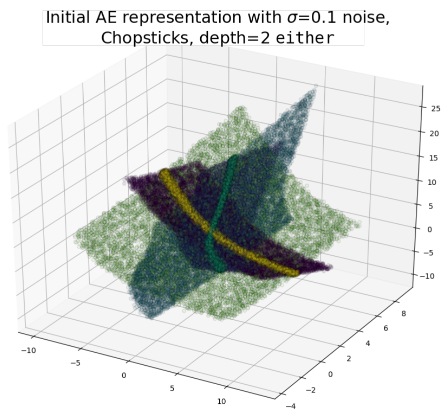
For all variants, at depth $d \geq 1$, we sample a linear "stick", and then "chop" it with probability $1 - 2^{-(d-1)}$. If we chop the stick, then we recursively generate a new stick of half the length, which we add to the second half of the current stick. We choose chop probabilities in this way so that, on average, we have equal counts of samples at each depth.

[1]Harvard University, Cambridge, MA, USA. Correspondence to: Andrew Slavin Ross <andrew_ross@g.harvard.edu>.

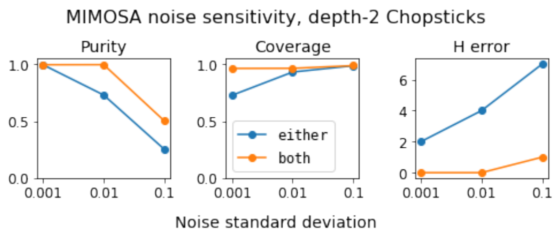
Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).



(a) Chopsticks instances corrupted by Gaussian noise.

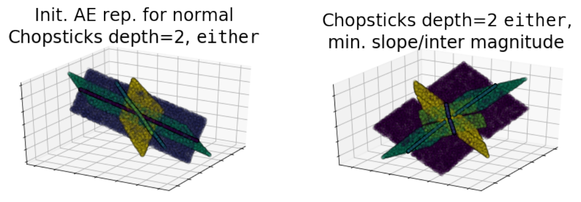


(b) Effect of noise on an initial AE representation.

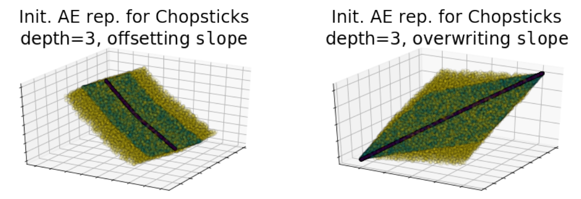


(c) Effect of noise on MIMOSA for two dataset variants.

Figure A.1. Illustration of the sensitivity of MIMOSA to data noise. In preliminary experiments, we find that noise poses the greatest problem for identifying the lowest-dimensional components, e.g. the 1D components in (b) that end up being classified as 2D or 3D. Tuning parameters would help, but we lack labels to cross-validate.



(a) Effect of setting a minimum slope/intercept magnitude.



(b) Effect of overwriting rather than offsetting slope.

Figure A.2. Initial AE representations for alternative variants of Chopsticks. Setting a minimum slope/intercept magnitude (a) causes representations to contain gaps. Overwriting rather than offsetting slope (b) changes the angle of lower- within higher-dimensional manifolds. Neither change breaks MIMOSA, which can still find the right hierarchy as long as manifolds remain similarly embedded with similar local SVD angles over gaps.

Although this framework already gives us a wide diversity of datasets, we could consider others. One option is to add noise, which hurts MIMOSA, though it depends on the dataset (Fig. A.1). Another option is to sample slopes and intercepts non-uniformly or even over non-convex sets; see e.g. Fig. A.2a, where we set slope/intercept magnitudes at least a threshold away from 0, which introduces gaps into the initial representation. In general, MIMOSA continues to return the right hierarchy for all such slope/intercept distributions, though COFHAE disentanglement tends to drop when variables are sampled from Gaussians, likely because of symmetry (with proper rescaling, we can rotate factorized Gaussians in any direction and preserve factorization; the same is not true for uniforms, though Locatello *et al.* (2018) show there must exist analogous, if more complex, transformations). Yet another possibility is to overwrite the slope and/or intercept when recursing, rather than offsetting them (Fig. A.2b). Overwriting slope does not affect MIMOSA performance, though it changes the orientation of lower-dimensional within higher-dimensional manifolds, which can affect COFHAE), but overwriting the intercept can break the geometrical nesting of manifolds at large slopes. Although we considered many of these options, we ultimately decided it was pedagogically best for our benchmark to distribute instances over maximally simple (but still arbitrarily deep) underlying manifold structures.

A.3. Computing H-error

Our H-error metric is meant to quantify the "edit distance" between two dimension hierarchies H and $\hat{H}$, but in a way that is invariant to merge-up and push-down operations, as well as reorderings of child groups. To implement it, we first convert H and $\hat{H}$ to a canonical form where each dimension group is labeled by the minimum downstream dimension of its leaves (which equals the dimension of the manifold component at the matching location in the original enclosure hierarchy), which renders us invariant to merge-up and push-down operations. We then reorder children in terms of the (sorted) concatenation of their downstream labels, which renders us invariant to child ordering in most cases. Finally, we apply the Zhang-Shasha algorithm for tree edit distance between ordered, labeled trees (Zhang and Shasha, 1989; Paassen *et al.*, 2015) to get our final H-error.

A.4. Complexity and Runtimes

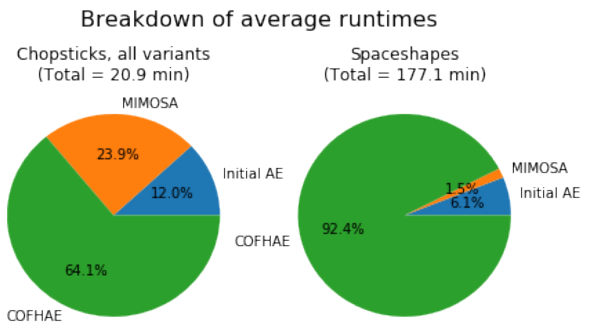


Figure A.3. Mean runtimes and percentage breakdowns for COFHAE and MIMOSA on Chopsticks and Spaceshapes, based on Tensorflow implementations running on single GPUs (exact model varies between Tesla K80, Tesla V100, GeForce RTX 2080, etc). Runtimes tend to be dominated by COFHAE, which is similar in complexity to existing adversarial representation learning methods (e.g. FactorVAE).

Per Fig. A.3, the total runtime of our method is dominated by COFHAE, an adversarial autoencoder method which has the same complexity as FactorVAE (Kim and Mnih, 2018) (linear in dataset size N and number of training epochs, and strongly affected by GPU speed).

MIMOSA could theoretically take more time, however, as the complexity of constructing a ball tree (Omohundro, 1989) for nearest neighbor queries is $O(|Z|N \log N)$. As such, initial dimensionality reduction is critical; in our Spaceshapes experiments, $|Z|$ is 7, whereas $|X|$ is 4096.

Other MIMOSA steps can also take time. With a `num_nearest_neighbors` of k, the complexity of running local SVD on every point in the dataset is $O(N(|Z|^2k + |Z|k^2 + k^3))$, providing another reason to reduce initial dimensionality and keep neighborhood size manageable (though ideally k should increase with $|Z|$ to robustly learn local manifold directions). Iterating over the dataset in BuildComponent and computing cosine similarity will also have complexity at least $O(Nkd^3(d + |Z|))$ for components of local dimensionality d, and detecting component boundaries can actually have complexity $O(Nke^d)$ (if this is implemented, as in our experiments, by checking if projected points are contained in their neighbors' convex hulls—though we also explored a much cheaper $O(Nk^2d)$ strategy of checking for the presence of neighbors in all principal component directions that worked almost as well).

Although these scaling issues are worth noting, MIMOSA was still relatively fast in our experiments, where runtimes were dominated by neural network training (Fig. A.3).

A.5. MIMOSA Hyperparameters

In this section, we list and describe all hyperparameters for MIMOSA, along with values that we used for our main results. We also present sensitivity analyses in Fig. A.5.

MIMOSA initial autoencoder (Algorithm 1, line 1)

- `initial_dim` - the dimensionality of the initial smooth autoencoder. As the sensitivity analysis in Fig. A.5 shows, this does not need to be as low as the intrinsic dimensionality of the data, which MIMOSA will estimate, and ideally should be a little larger. We defaulted to using the maximum intrinsic dimensionality plus 1; in a real-world context where this information is not available, it can be estimated by starting at `initial_dim` $= |X|$ and reducing until initial autoencoder reconstruction error starts increasing.
- Training and architectural details appropriate for the data modality (e.g. convolutional layers for images). See §A.1 for our choices.

LocalSVD (Algorithm 3)

- `num_nearest_neighbors` - the neighborhood size for local SVD and later traversal. We used 40. Must be larger than `initial_dim`; could also be replaced with a search radius.
- `ransac_frac` - the fraction of neighbors to refit SVD. We used $2/3$. Note that we do not run traditional, multi-step RANSAC (Fischler and Bolles, 1981), but a more efficient two-step approximation, where we define the loss term based an aggregation of reconstruction errors across dimensions. Another (less efficient but potentially more robust) option would be to iteratively re-fit SVD using the points with lowest reconstruction error at each dimension, and check if the resulting eigenvalues meet our cutoff criteria.
- `eig_cumsum_thresh` - the minimum fraction of variance SVD dimensions must explain to determine local dimensionality. We used 0.95. For noisy or sparse data, it might be useful to reduce this parameter.
- `eig_decay_thresh` - the minimum multiplicative factor by which SVD eigenvalues must decay to determine local dimensionality. We used 4. It might also be useful to reduce this parameter for sparse data.

Note that our LocalSVD algorithm can be seen as a faster version of Multiscale SVD (Little *et al.*, 2009), which is used in an analogous way by Mahapatra and Chandola (2017), but would require repeatedly computing singular value decompositions over different search radii for each point.

BuildComponent (Algorithm 5)

- `cos_simil_thresh` - neighbors' local SVDs must be this similar to add to the component. This corre-

Algorithm 3 LocalSVD(Z)

```
1: Run SVD on $Z$ (a design matrix of dimension num_nearest_neighbors by initial_dim)
2: if ransac_frac $< 1$ then
3:     for each dimension $d$ from $1$ to initial_dim $- 1$ do
4:         for each point $z_n$ do
5:             Compute the reconstruction error for $z_n$ using the only first $d$ SVD dimensions
6:         end for
7:     end for
8:     Take the norm of reconstruction errors across dimensions, giving a vector of length num_nearest_neighbors
9:     Re-fit SVD on points whose error-norms are less than the $100 \times$ ransac_frac percentile value.
10: end if
11: for each dimension $d$ from $1$ to initial_dim $- 1$ do
12:     Check if the cumulative sum of the first $d$ eigenvalues is at least eig_cumsum_thresh
13:     Check if the ratio of the $d$th to the $d + 1$st eigenvalue is at least eig_decay_thresh
14:     if both of these conditions are true then
15:         return only the first $d$ SVD components
16:     end if
17: end for
18: return the full set of SVD components otherwise
```

Algorithm 4 TangentPlaneCos(U, V)

```
1: if $U$ and $V$ are equal-dimensional then
2:     return $|\det(U \cdot V^T)|$
3: else
4:     return $0$
5: end if
```

Algorithm 5 BuildComponent(z_i, neighbors, svds)

```
1: Initialize component to $z_i$ and neighbors $z_j$ not already in other components where $\text{TangentPlaneCos}(\text{svds}_i, \text{svds}_j) \geq$ cos_simil_thresh (Algorithm 4).
2: while the component is still growing do
3:     Add all points $z_k$ for which at least contagion_num of their neighbors $z_\ell$ are already in the component with $\text{TangentPlaneCos}(\text{svds}_k, \text{svds}_\ell) \geq$ cos_simil_thresh.
4:     Skip adding any $z_k$ already in another component.
5: end while
6: return the set of points in the component
```

Algorithm 6 MergeComponents(components, svds)

```
1: Discard components smaller than min_size_init.
2: for each component $c_i$ do
3:     Construct a local ball tree for the points in $c_i$.
4:     Set $c_i$.edges to points not contained in the convex hull of their neighbors in local SVD space.
5: end for
6: Initialize edge overlap matrix $M$ of size |components| by |components| to 0.
7: for each ordered pair of equal-dimensional components $(c_i, c_j)$ do
8:     Set $M_{ij}$ to the fraction of points in $c_i$.edges for which the closest point in $c_j$.edges has local SVD tangent plane similarity above cos_simil_thresh.
9: end for
10: Average $M$ with its transpose to symmetrize.
11: Merge all components $c_i \neq c_j$ of equal dimensionality $d$ where $M_{ij} \geq$ min_common_edge_frac($d$).
12: Discard components smaller than min_size_merged.
13: return the merged set of components
```

Algorithm 7 ConstructHierarchy(components)

```
 1: for each component $c_i$ do
 2:     Set $c_i$.neighbor_lengthscale to the average distance of points to their nearest neighbors inside the component (computed using the local ball tree from Algorithm 6)
 3: end for
 4: for each pair of different-dimensional components $(c_i, c_j)$, $c_i$ higher-dimensional do
 5:     Compute the average distance from points in $c_i$ to their nearest neighbors in $c_j$ (via ball tree).
 6:     Divide this average distance by $c_i$.neighbor_lengthscale.
 7:     if the resulting ratio $\leq$ neighbor_lengthscale_mult then
 8:         Set $c_j \in c_i$ ($c_j$ is enclosed by $c_i$)
 9:     end if
10: end for
11: Create a root node with edges to all components which do not enclose others.
12: Transform the component enclosure DAG into a tree (where enclosing components are children of enclosed components) by deleting edges which:
        1. are redundant because an intermediate edge exists, e.g. if $c_1 \in c_2 \in c_3$, we delete the edge between $c_1$ and $c_3$.
        2. are ambiguous because a higher-dimensional component encloses multiple lower-dimensional components (i.e. has multiple parents). In that case, preserve only the edge with the lowest distance ratio.
13: Convert the resulting component enclosure tree into a dimension hierarchy:
        1. If the root node has only one child, set it to be the root. Otherwise, begin with a dimension group with a single categorical dimension whose options point to groups containing each child.
        2. For the rest of the component tree, add continuous dimensions until the total number of continuous dimensions up to the root equals the component's dimensionality.
        3. If a component has children, add a categorical dimension that includes those child groups as options (recursing down the tree), along with an empty group ($\boxed{\emptyset}$) option.
14: return the dimension hierarchy
```

Algorithm 8 HAE_θ.encode$(x; \tau)$

```
1: Encode $x$ using any neural network architecture as a flat vector $z_{pre}$, with size equal to the number of continuous variables plus the number of categorical options in $\text{HAE}_\theta$.hierarchy.
2: Associate each group of dimensions in the flat vector with variables in the hierarchy.
3: For all of the categorical variables, pass their options through a softmax with temperature $\tau$.
4: Use the softmax outputs to recursively mask all components of $z_{pre}$ corresponding to variables *below* each option in $\text{HAE}_\theta$.hierarchy.
5: return the masked representation, separated into discrete $a'$, continuous $z$, as well as the mask $m$ (for determining active dimensions later).
```

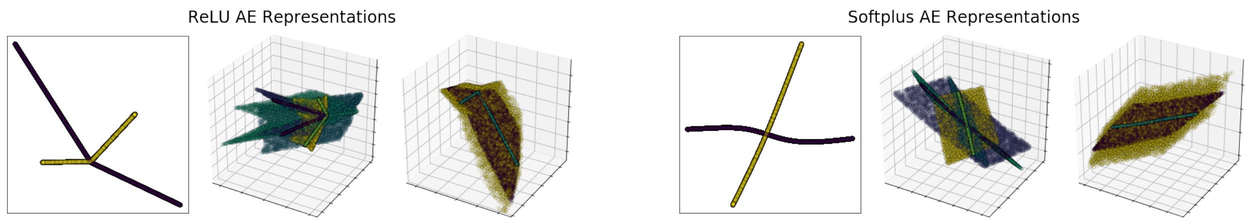


Figure A.4. Comparison of the latent spaces learned by MIMOSA initial autoencoders with ReLU (left) vs. Softplus (right) activations on three versions of Chopsticks (depth=1 `either`, depth=2 `either`, and depth=3 `slope`). Each plot shows encoded data samples colored by their ground-truth location in the dimension hierarchy. Because ReLU activations are non-differentiable at 0, the resulting latent manifolds contain sharp corners where local SVD directions change discontinuously, causing issues for BuildComponent and MergeComponents within MIMOSA (Algorithms 5 and 6). Representations learned by autoencoders with smooth activation functions work much better.

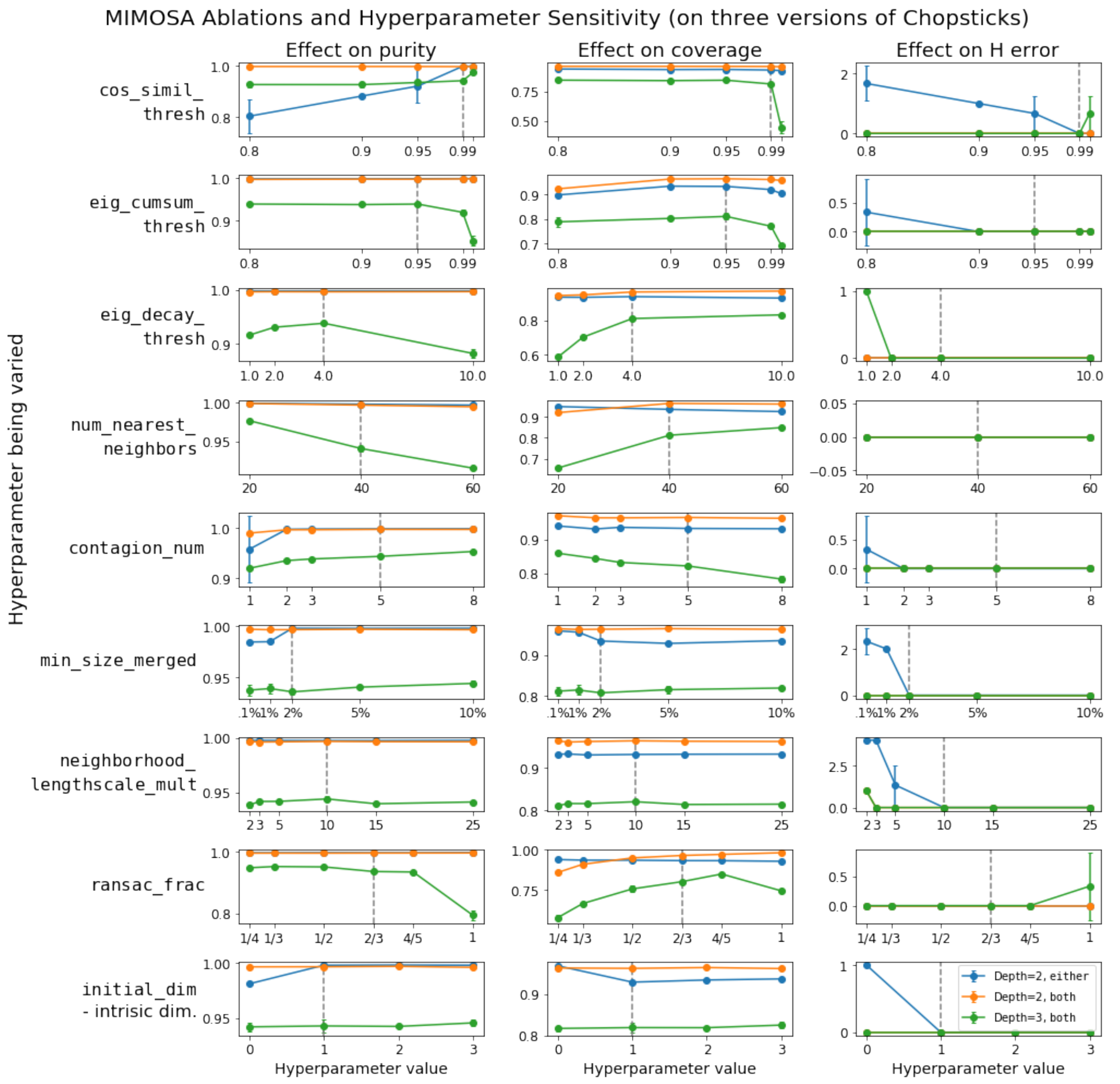


Figure A.5. Effect of varying different hyperparameters (and ablating different robustness techniques) on MIMOSA. Default values are shown with vertical gray dotted lines, and for each hyperparameter (top to bottom), average coverage (left), purity (middle), and H error (right) when deviating from defaults are shown for three versions of the Chopsticks dataset. Results suggest both a degree of robustness to changes (degradations tend not to be severe for small changes), but also the usefulness of various components; for example, results markedly improve on some datasets with `contagion_num`>1 and `ransac_frac`<1 (implying contagion dynamics and RANSAC both help). Many parameters exhibit tradeoffs between component purity and dataset coverage.

sponds to the ϵ parameter from Mahapatra and Chandola (2017). We used 0.99 for Chopsticks and 0.95 for Spaceshapes; in general, we feel this is one of the most important parameters to tune, and should generally be reduced in the presence of noise or data scarcity.

- `contagion_num` - only add similar points to a manifold component when a threshold fraction of their neighbors have already been added. This is useful for robustness, and corresponds to the T parameter from Mahler (2020) (but expressed as a number rather than a fraction). We used 5 for Chopsticks and 3 for Spaceshapes. Values above 20% of `num_nearest_neighbors` will likely produce poor results, and we found the greatest increases in robustness just going from 1 (or no contagion dynamics) to 2.

MergeComponents (Algorithm 6)

- `min_size_init` - discard initial components smaller than this, which helps speed up the algorithm (by reducing the number of pairwise comparisons) and avoid incorrect merges through single-point components. We used 0.02% of the dataset size, or 20 points.
- `min_size_merged` - discard merged components smaller than this, which helps exclude spurious higher-dimensional interstitial points that appear at the boundaries where lower-dimensional components intersect. We used 2% of the dataset size, or 2000 points.
- `min_common_edge_frac(d)` - the minimum fraction of edges that two manifold components must share in common to merge, as a function of dimensionality d. We used $2^{-d-1} + 2^{-d-2}$; this is based on the idea that two neighboring (possibly distorted) hypercubes of dimension d should match on one of their sides; since they have 2^d sides, the fraction of matching edge points would be 2^{-d}. However, for robustness (as not all manifold segments will be hypercubes, and even then some edge points may not match), we average that fraction with the smaller fraction that would need for a $d+1$ dimensional hypercube, or 2^{-d-1}, for our resulting $2^{-d-1} + 2^{-d-2}$. In general, we found that this choice was not critical in the noiseless data case, as matches were common for separated components with the same true assignments and rare for others, but it did help in cases with many intersecting components.

ConstructHierarchy (Algorithm 7)

- `neighbor_lengthscale_mult` - the threshold for deciding whether a higher-dimensional component "encloses" a lower-dimensional component, expressed as a ratio of (1) the average distance from lower-dimensional component points to their nearest neighbors in the higher-dimensional component (inter-component distance), to (2) the average distance of points in the higher-dimensional component to their nearest neighbors in that same component (intra-component distance). We used 10, which we found was robust for our benchmarks, though it may need to be increased if ground-truth components are higher-dimensional than those in our benchmarks.

References

Christopher P Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in beta-vae. *arXiv preprint arXiv:1804.03599*, 2018.

Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

Hyunjik Kim and Andriy Mnih. Disentangling by factorising. In *International Conference on Machine Learning*, 2018.

Anna V Little, Jason Lee, Yoon-Mo Jung, and Mauro Maggioni. Estimation of intrinsic dimensionality of samples from noisy low-dimensional manifolds in high dimensions with multiscale svd. In *2009 IEEE/SP 15th Workshop on Statistical Signal Processing*, 2009.

Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Rätsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. *arXiv preprint arXiv:1811.12359*, 2018.

Suchismit Mahapatra and Varun Chandola. S-isomap++: multi manifold learning from streaming data. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 716–725. IEEE, 2017.

Barbara I Mahler. Contagion dynamics for manifold learning. *arXiv preprint arXiv:2012.00091*, 2020.

Stephen M Omohundro. *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

Benjamin Paassen, Bassam Mokbel, and Barbara Hammer. A toolbox for adaptive sequence dissimilarity measures for intelligent tutoring systems. In *EDM*, page 632, 2015.

Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 1989.

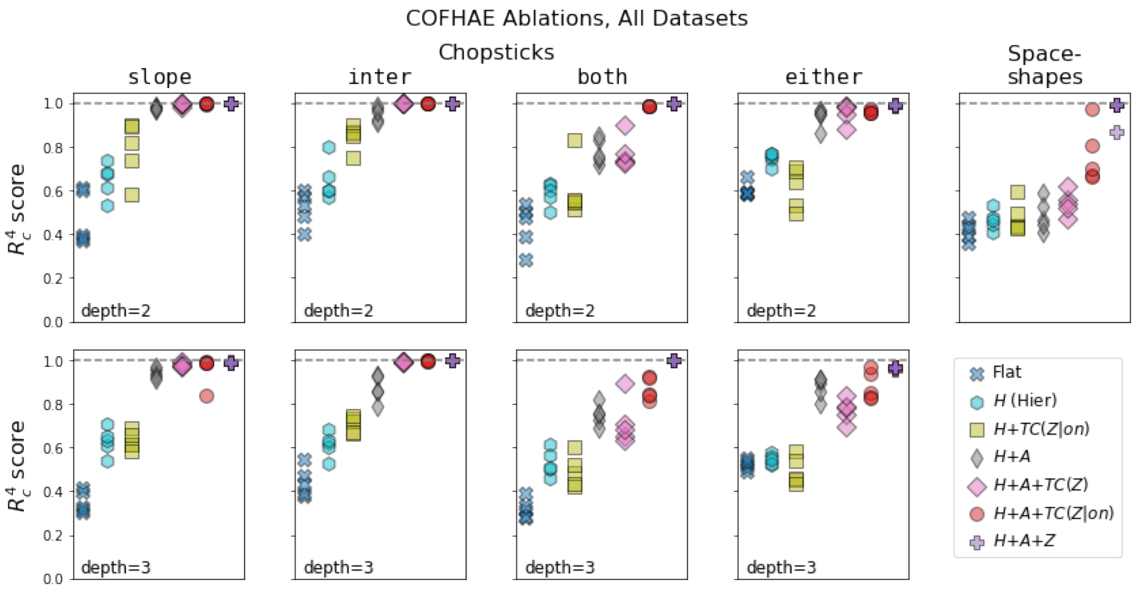


Figure A.6. A fuller version of main paper Fig. 5 showing COFHAE ablations on all datasets. Hierarchical disentanglement tends to be low for flat AEs (Flat), better with ground-truth hierarchy H (Hier H), and even better after adding supervision for ground-truth assignments A (H+A). Adding a FactorVAE-style marginal TC penalty (H+A+$TC(Z)$) sometimes helps disentanglement, but making that TC penalty conditional (H+A+$TC(Z|on)$, i.e. COFHAE) tends to help more, bringing it close to the near-optimal disentanglement of a hierarchical model whose latent representation is fully supervised (H+A+Z). Partial exceptions include the hardest three datasets (Spaceshapes and depth-3 compound Chopsticks), where disentanglement is not consistently near 1; this may be due to non-identifiability or adversarial optimization difficulties.

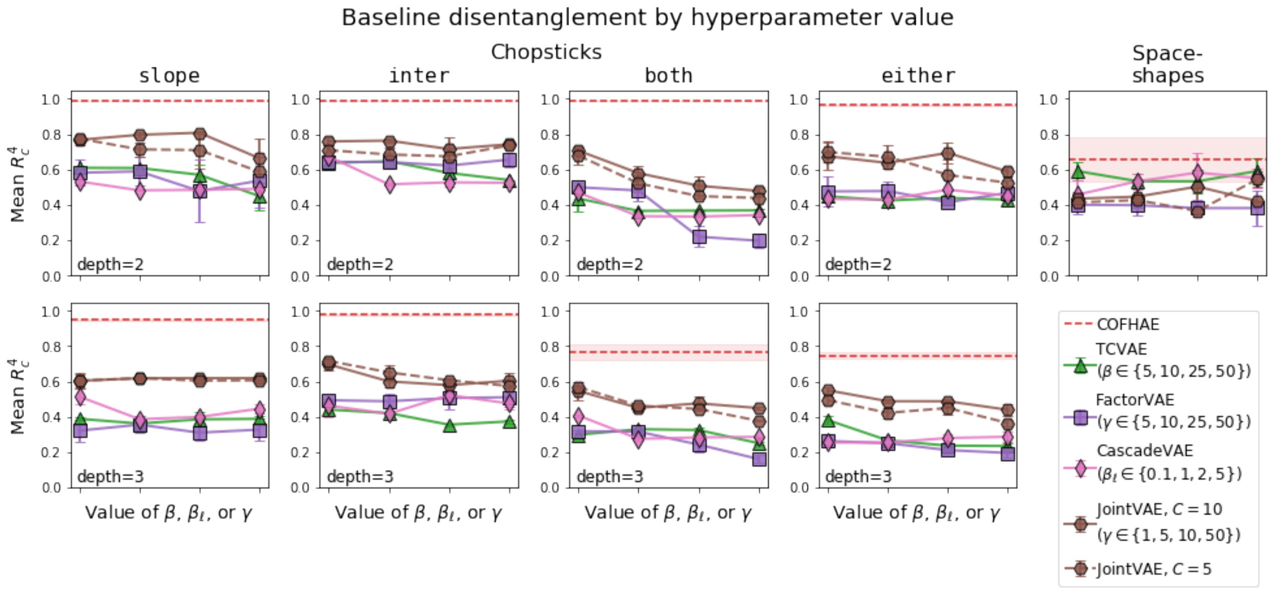
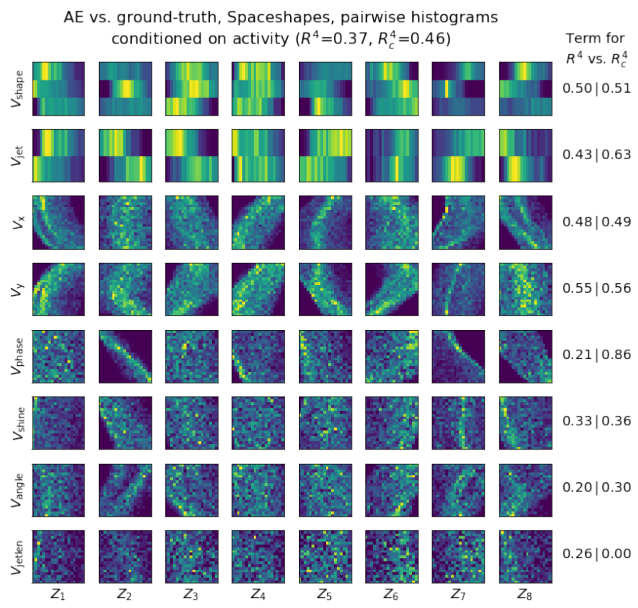
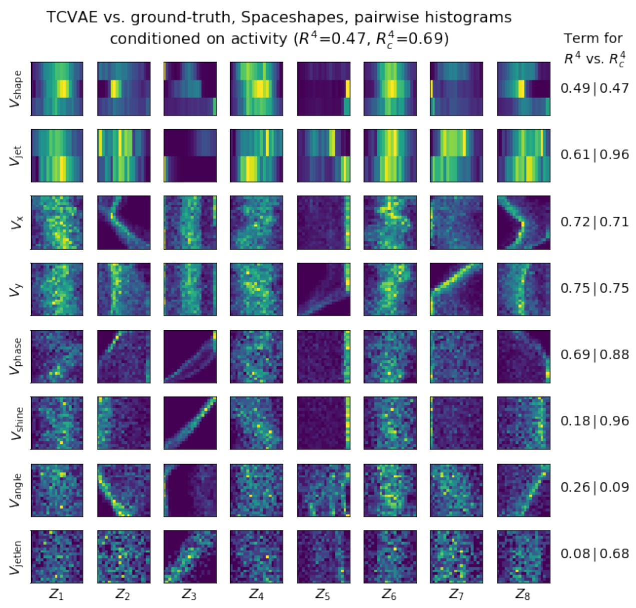


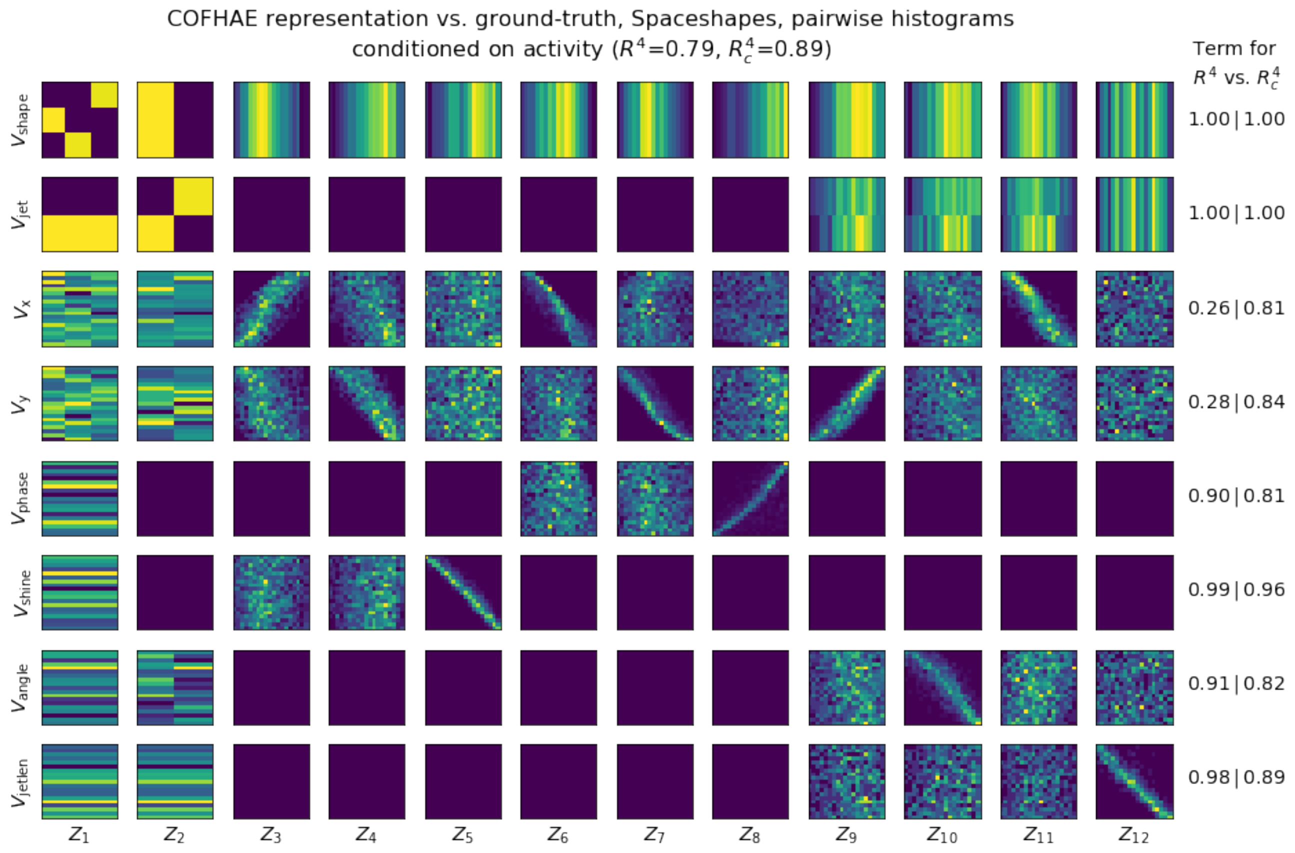
Figure A.7. Varying disentanglement penalty hyperparameters for baseline algorithms (TCVAE, FactorVAE, CascadeVAE, and JointVAE). Markers indicate mean R^4_c over 5 trials, with standard deviation errorbars. In contrast to COFHAE (mean performance in red, with standard deviation in pink), no setting produces near-optimal disentanglement, and disentanglement often *decreases* with increasing disentanglement penalty strength.



(a) AE pairwise histograms and R^4/R^4_c scores



(b) TCVAE pairwise histograms and R^4/R^4_c scores



(c) COFHAE pairwise histograms and R^4/R^4_c scores

Figure A.8. Pairwise histograms of ground-truth vs. learned variables for a flat autoencoder (top left), β-TCVAE (top right), and the best-performing run of COFHAE (bottom) on Spaceshapes. Histograms are conditioned on both variables being active, and dimension-wise components of the R^4_c score are shown on the right. β-TCVAE does a markedly better job disentangling certain components than the flat autoencoder, but in this case, COFHAE is able to fully disentangle the ground-truth by modeling the discrete hierarchical structure. See Fig. A.9 for a hierarchical latent traversal, or https://hreps.s3.amazonaws.com/viz/index.html?dataset=spaceshapes&model=cofhae for an interactive visualization.

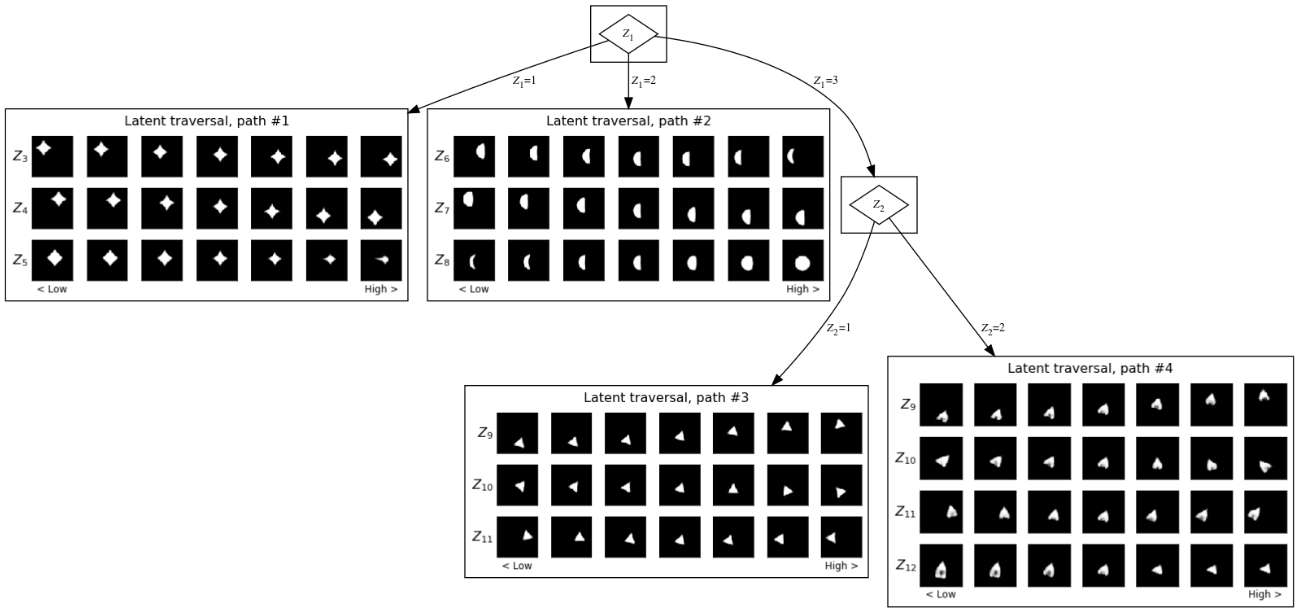


Figure A.9. Hierarchical latent traversal plot for the Spaceshapes COFHAE model shown in Fig. A.8c. Individual latent traversals show the effects of linearly sweeping each *active* dimension from its 1st to 99th percentile value (center column shows the same input with intermediate values for all active dimensions). Consistent with Fig. A.8c, the model is not perfectly disentangled, though primary correspondences are clear: star `shine` is modeled by Z_5, moon `phase` is modeled by Z_8, ship `angle` is modeled by Z_{10}, ship `jetlen` is modeled by Z_{12}, and (`x`, `y`) are modeled by (Z_3, Z_4), (Z_6, Z_7), and (Z_{11}, Z_9) respectively for each shape. See also an interactive visualization.

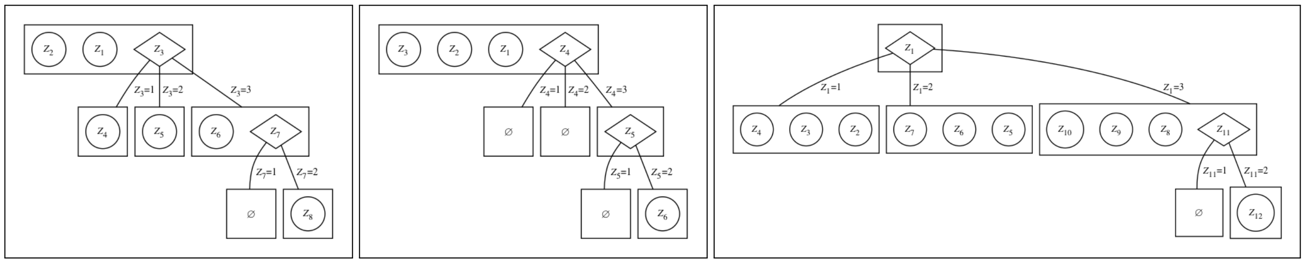


Figure A.10. Three different potential hierarchies for Spaceshapes which all have the same structure of variable groups and dimensionalities, but with different distributions of continuous variables across groups. The ambiguity in this case is that the continuous variable that modifies each shape (`phase`, `shine`, `angle`) could either be a child of the corresponding shape category, or be "merged up" and combined into a single top-level continuous variable that controls the shape in different ways based on the category. Alternatively, the location variables `x` and `y` could instead be "pushed down" from the top level and duplicated across each shape category. In each of these cases, the learned representation still arguably disentangles the ground-truth factors—in the sense that for any fixed categorical assignment, there is still 1:1 correspondence between all learned and ground-truth continuous factors. We deliberately design our R_c^4 and H-error metrics in §6 to be invariant to these transformations, leaving this specific disambiguation to future work.

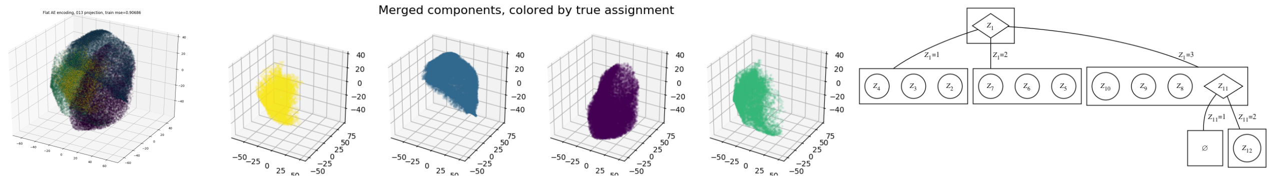


Figure A.11. MIMOSA-learned initial encoding (left), components (middle), and hierarchy (right) for Spaceshapes. Initial points are in 7 dimensions and projected to 3D for plotting. Three identified components are 3D and one is 4D. Analogue of Fig. 3 in the main text.

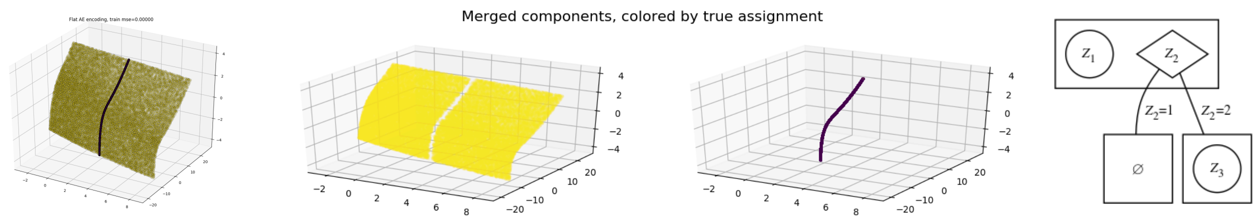


Figure A.12. MIMOSA-learned initial encoding (left), 2D and 1D components (middle), and hierarchy (right) for depth-2 Chopsticks varying the slope. Analogue of Fig. 3 in the main text.

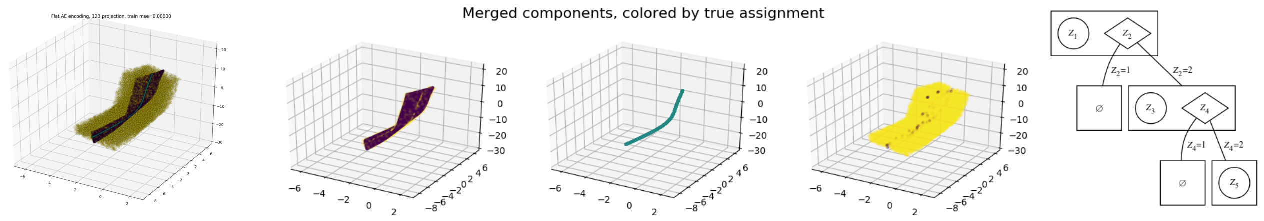


Figure A.13. MIMOSA-learned initial encoding (left), 2D, 1D, and 3D components (middle), and hierarchy (right) for depth-3 Chopsticks varying the slope. Initial points are in 4 dimensions and projected to 3D for plotting. Analogue of Fig. 3 in the main text.

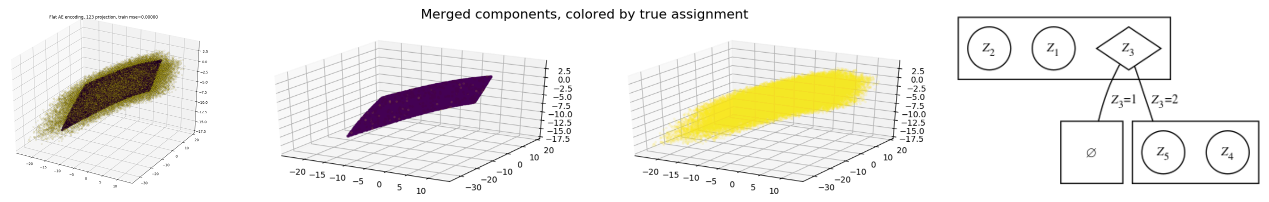


Figure A.14. MIMOSA-learned initial encoding (left), 2D and 4D components (middle), and hierarchy (right) for depth-2 Chopsticks varying `both` slope and intercept. Initial points are in 5 dimensions and projected to 3D for plotting. Analogue of Fig. 3 in the main text.

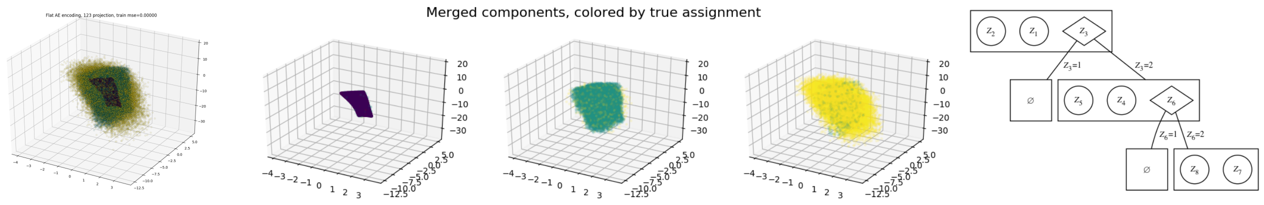


Figure A.15. MIMOSA-learned initial encoding (left), 2D, 4D, and 6D components (middle), and hierarchy (right) for depth-2 Chopsticks varying `both` slope and intercept. Initial points are in 7 dimensions and projected to 3D for plotting. Analogue of Fig. 3 in the main text.

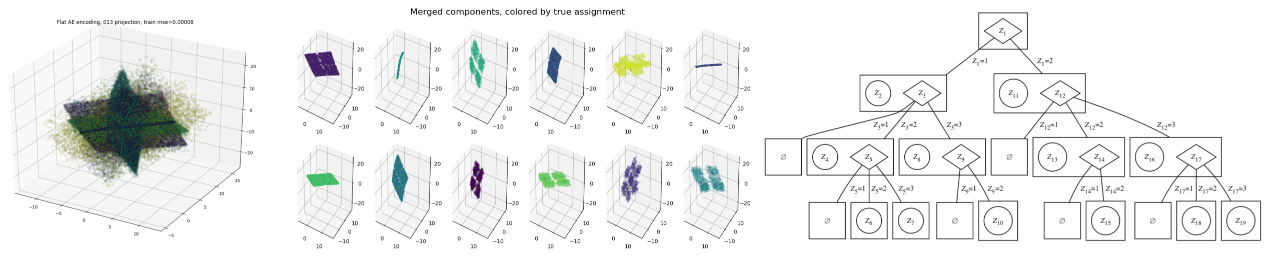


Figure A.16. MIMOSA-learned initial encoding (left), 1D-3D components (middle), and hierarchy (right) for depth-3 Chopsticks varying either slope or intercept. Note that the learned hierarchy is not quite correct (two nodes at the deepest level are missing). Initial points are in 5 dimensions and projected to 3D. Analogue of Fig. 3.

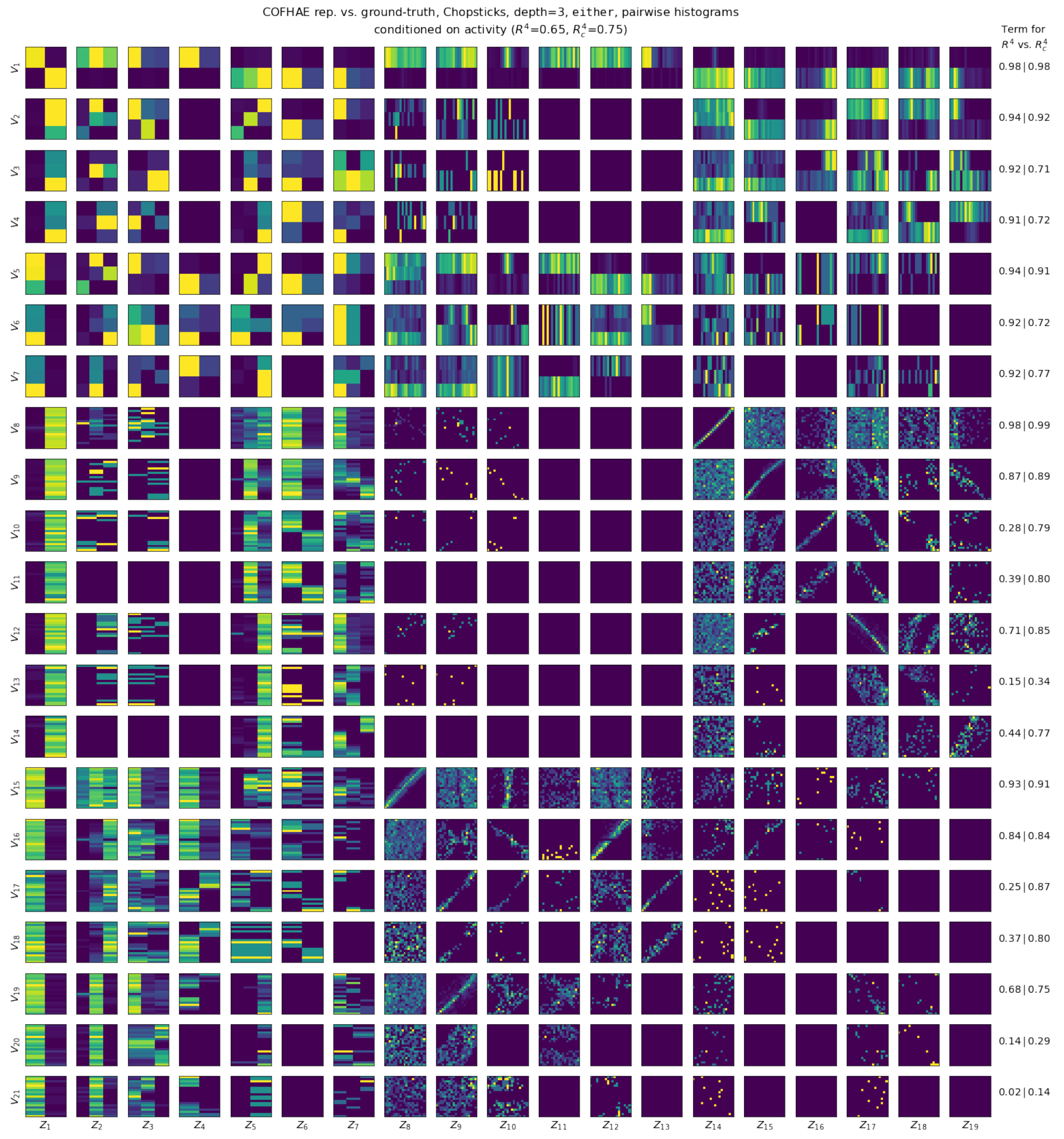


Figure A.17. Pairwise histograms of ground-truth vs. learned variables for COFHAE on the most complicated hierarchical benchmark (Chopsticks at a recursion depth of 3 varying `either` slope or intercept). Histograms are conditioned on both variables being active, and dimension-wise components of the R_c^4 score are shown on the right. Despite the depth of the hierarchy, COFHAE representations model it fairly well.