

DEEP CLUSTERING AND INTERPOLATION VIA THE FEDERATED SELF-ORGANIZING MAP

Anonymous authors

Paper under double-blind review

ABSTRACT

We introduce FedSOM, a clustering and interpolation module based on the Self-organizing Map (SOM), which can be appended to any encoder and which can be trained in a federated way either in tandem with the encoder or post training on the resulting representations. The result is a discrete moduli space of representations that provides for cluster or sample-level interpolation, hierarchical clustering, and can be leveraged as a function to cluster new vectors at test time. This moduli space can either be created from data alone or by glueing pre-existing clusters along regions of commonality, although we do not explore the latter in this work. Interpolation is accomplished by considering the n -dimensional tensor underlying the SOM as a weighted undirected graph, where the weights are computed as a function of the dispersion of the two clusters corresponding to the nodes bounding the given edge. Any two clusters or samples may then be interpolated by computing the lowest-cost path between their associated graph nodes via Dijkstra’s algorithm. The method is validated on MNIST-like and parsed-binary malware datasets.

1 INTRODUCTION

Self-supervised learning has been successfully applied to clustering and interpolation tasks in domains such as vision, text, and speech, in which datasets are large and homogeneous. However, recent advancements such as VIME [27] have facilitated the application of self-supervised learning to clustering and interpolation on tabular data. The typical approach to creating representations for downstream tasks is to minimize a contrastive loss, in which positive pairs are constructed by conditionally sampling from column marginals. Non-contrastive methods such as VICReg [2] have also shown great promise on downstream image classification tasks. Approaches to deep clustering vary widely not only in terms of preferred data, but in terms of classical cluster approaches, i.e., centroid based, density based, and online vs batch. In [44] they explore clustering utterance data with BERT embeddings with a centroid-based approach. Works such as [8] and [29] rely on a softmax function to assign cluster labels, meaning that the number of clusters must be fixed at train time. The number of clusters are also fixed at train time in [17]. While the vast majority of deep clustering methods are batch trained, online methods such as [45] also exist, in which the number of clusters need not be set at train time. In [48] the authors describe a GMM-type approach in which the number of clusters is cast as a hidden variable in an EM-type approach. Similar to multi-modal learning on tabular data, multi-view learning has successfully been applied to both tabular and feature-homogeneous datasets as in [46] and [47], in which multiple views take the place of tabular perturbations. The latter seeks to discover correlations between multiple views of the same object. This differs from our work in that perturbations are created for the sake of creating tightly-grouped representations for downstream clustering tasks.

Self-organizing map structures have long been leveraged both for clustering and interpolation. Training methodologies vary widely, but the main idea is that SOM tensor nodes correspond to cluster labels. In [9], the authors construct SOM-VAE in which they combine the structure of a SOM with a differentiable learning objective to learn the underlying discrete structure. Learning such a discrete structure is an example of vector quantisation. See [23] in which a vector quantised-variational autoencoder is constructed for homogeneous (images, video, speech) rather than tabular data. Interpolation is one of the primary applications of self-organizing map structures and there are many such approaches. Leveraging weighted-averages between weight vectors attached to SOM tensor nodes is

054 a popular approach taken in [10], in which they leverage topological information contained in local
055 neighborhoods within the SOM to learn nonlinear relationships between samples in the input data.
056 A additional layer may also be added as in [11] that computes interpolations between two given
057 nodes. Interpolation may also be carried out by leveraging the entire SOM. This sort of approach
058 is appropriate when a SOM is trained for each sample in an image dataset [13]. Retraining is not
059 typically required for SOM-based interpolation. See [26] for further details. Interpolation is also
060 leveraged to aid in the learning of deep networks, as in [21] in which interpolation is leveraged to
061 learn intermediate representations of time series samples.

062 1.1 OUR CONTRIBUTION

- 064 • A novel self-organizing map structure that can be trained in a federated way
- 065 • A graph structure underlying our self-organizing map, which can be leveraged for cluster
066 interpolation in a way that does not depend on the manifold assumption
- 067 • A moduli space of cybersecurity data that allows analysts to visualize relationships, per-
068 form interpolation-based parser analysis, and perform similarity queries that are condi-
069 tioned on neighboring families

070 1.1.1 LIMITATIONS

071 The number of nodes in the SOM tensor τ is exponential in the dimension of the SOM, so training
072 becomes prohibitive for $\dim \tau \geq 4$ when the number of samples exceeds $1e5$. Because train-
073 ing is performed in an unsupervised way, the contrastive learning objective can lead to label-
074 inhomogeneous clusters and over-partitioning of coherent clusters. SOM training is performed
075 classically, meaning that the method does not benefit from being structured as a loss minimiza-
076 tion problem. This also represents a benefit, however, in that the method can be applied to existing
077 representations.

078 Finally, malware representations result from feeding parsed binaries through the encoder, which
079 means information available to FedSOM is limited by the parser itself. Future work should include
080 representations created from raw bytes.

081 2 RELATED WORK

082 In this work we define interpolation between two points as identifying a finite sequence of samples
083 that are successively similar and represent a transformation from the first sample to the second. We
084 do not refer to *interpolative* points as in [23,25] in which a point is said to be interpolative if it
085 lies within the convex hull of the given dataset. Most prior art leveraging self organizing maps for
086 interpolation involves the construction a sequence of self organizing maps that functions as a discrete
087 set of intermediate steps between two given samples. This differs dramatically from our approach
088 in which interpolation takes place within a fixed SOM. The purpose of leveraging a self-organizing
089 map is to create a discrete topological space to represent a typically continuous latent space, or to
090 learning relationships between classes, as in [5]. As in the case of [14], the SOM clustering during
091 training encourages convex clusters. As in [14], we considered training the encoder in tandem
092 with the clustering algorithm, but settled on training the SOM on trained representations for wider
093 method applicability. There is no effort devoted to learning illusory underlying manifolds as in
094 [28] and instead focus on learning proximity-based cluster assignments via the SOM. In [7], they
095 construct smooth interpolations in latent space to construct interpolations via GANs in the space of
096 architectural design. Our notion of interpolation is akin to the *Category Morphing* process described
097 in [15]. Self-organizing maps have been used previously for interpolation via stepping successively
098 between neighboring nodes. This method was leveraged in [9] for interpolating between steps in
099 a time series. Interpolation between SOM nodes is performed in [9] by computing a weighted
100 average between weight vectors, where the weight is a function of the given input vectors. This
101 is essentially a generative method in that representation vectors which do not correspond to any
102 samples in the original dataset are leveraged for interpolation. In contrast, all interpolations in this
103 work are performed using only samples that exist in the data.

104 The application of these methods to cybersecurity data is non-existent. Our self-organizing map
105 approach requires the user to specify only an upper bound on the number of clusters by specifying
106

108 the dimensions of the underlying tensor. Small perturbations in the encoder will change the inferred
109 metric on representation space, and therefore alter the geodesic interpolative path connecting two
110 test points. We avoid this sensitivity by creating a discrete moduli space of the input data optimized
111 for clustering and interpolation. Current SOM-based interpolation approaches involve either the dis-
112 cretization of each sample by training a SOM and considering the sequence of such, or by computing
113 a weighted average of weight vectors attached to neighboring nodes.

114 115 3 PRELIMINARIES 116

117 For the purpose of creating representations on which FedSOM is trained, we train both UMAP
118 and neural network encoders, the latter by minimizing some form of self-supervised loss. We take
119 the term *interpolation* to refer to a transformational sequence of samples that connects two given
120 samples, rather than referring to the well-known mode of overfitting, as described in [3].
121

122 3.1 SELF-ORGANIZING MAPS 123

124 A self-organizing map is a collection of learnable weight vectors indexed by the coordinates of
125 a tensor, along with an update rule that encourages the weight vectors to closely match the input
126 vectors. This update rule includes a neighborhood function that determines the extent to which
127 weight vectors attached to neighbor nodes are updated. See 1 for a visual representation. The key
128 SOM hyperparameter is σ , the variance of the Gaussian kernel which determines the reach of the
129 neighborhood function. The result is a map of the input data such that weight vector similarity
130 recedes with distance within the indexing tensor.
131

132 3.2 DATA 133

134 Interpolation and clustering of cybersecurity data is the focus of this work, but the method itself can
135 be applied to a variety of sparse tabular datasets. In order to validate this claim, we train our models
136 not only on parsed binaries and network data, but on MNIST-like datasets. This latter family was
137 desirable for its sparsity and its well-defined structural and ontological groupings.
138

139 3.2.1 CYBERSECURITY

140 Parsed portable executable datasets based on a parser are not amenable to smooth interpolation
141 within representation space. It is highly unlikely that a given region in latent space could give rise to
142 feature vectors that could have been produced by parsing executable binaries. For this reason, inter-
143 polation is performed by considering only representations corresponding to samples in the corpus.
144

145 In the case of Ember and Sorel20M, we considered only feature vectors corresponding to malicious
146 binaries. This was done to focus our experiments, but future work should include clustering and
147 interpolation with clean files included. In this way one may be able to discover and understand
148 parasitic file infectors from the point of view of the leveraged parser.
149

150 4 METHOD 151

152 FedSOM is a federated self-organizing map with an underlying weighted graph structure that fa-
153 cilitates clustering and interpolation between any two clusters or any two samples. Interpolation
154 between clusters of parsed binaries can be leveraged to understand the parser by inspecting feature
155 distribution changes through the interpolation path.

156 We consider two approaches to embedding creation for the sake of FedSOM clustering and interpo-
157 lation. The first is an encoder with a self-supervised learning objective. This approach benefits from
158 the scalability and flexibility of batch training. The second is UMAP, which offers the advantage of
159 considering all data at once and allows for a globally optimized topological embedding for datasets
160 that satisfy the manifold hypothesis. See ¹ for code and data.
161

¹<https://github.com/fed-som/fedsom>, <https://zenodo.org/records/11205063>

162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215

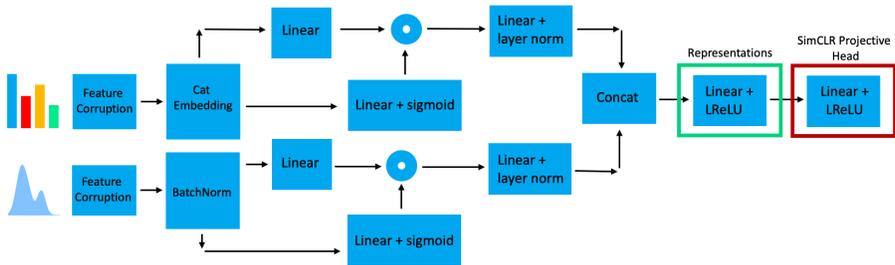


Figure 1: Categorical columns are first separated from continuous columns either via a heuristic. The categorical columns are separately passed through embedding layers. The remainder of the network is residual with a projection head for use in the contrastive learning objective.

4.1 NEURAL NETWORK ENCODER

The encoder is a vanilla residual network that separately handles continuous and categorical values. The categorical variables are discovered by an initial sweep through the data, based on a set of heuristics. We cast a column as categorical if any one of the following holds: (1) contains only booleans, (2) contains only integers, (3) contains at least one string.

Categorical columns are encoded by an integer label encoder, and then embedded into a low-dimensional Euclidean space via an embedding layer \mathcal{E} . These embeddings are concatenated and passed through two separate fully-connected layers. The output of the first passes through a sigmoid and is combined with the latter via a Hadamard product. The result then passes through a third fully connected layer after which we apply layer norm.

Replacing the encoder and embedding layer with batchnorm, the numerical columns pass through an identical network. The numerical and categorical embeddings are then concatenated and passed through two fully-connected + leaky relu layers, the latter function as the SimCLR projection head.

All loss functions, with the exception of VICReg, are contrastive in nature. This means that for a given batch x , the loss is a function of the form $\mathcal{L}(x, \tilde{x})$, where \tilde{x} is a perturbed version of x and (x, \tilde{x}) is termed a *positive* pair. For the sake of constructing positive pairs, we leverage a perturbations module.

4.1.1 TABULAR PERTURBATIONS MODULE

Perturbation is performed by conditionally drawing from the marginal distribution of each column, which is a uniform over $[\min, \max]$ if the column is numerical and a uniform over unique observations if the column is categorical. The uniform distribution was chosen for the sake of maximizing sample speed.

We first perform a distribution learning step in which the sparsity of each column is measured via Welford’s algorithm applied counts of $x.is_null() | (x == 0)$. Columns with sparsity above a set threshold are discarded before being passed to the encoder. An example of such a column would be the upper left pixel of the MNIST dataset.

We perturb batches according to the following recipe:

- remove columns above the *upper* sparsity threshold ψ_u
- for each column c , draw a Bernoulli random variable $\rho_c \sim B(1, p)$ for p fixed across columns
- if $\rho_c = 1$, draw `batch_size` samples from the marginal distribution $P(c)$ of column c
- if the sparsity of c is above the *lower* sparsity threshold ψ_b , then draw `batch_size` random variables $\rho_r \sim B(1, q)$ and for each $\rho_r = 1$, sample from $U(\Omega_c)$ if c is categorical and from $U([\min(\Omega_c), \max(\Omega_c)])$ if c is numerical.

The purpose of the final step is to ensure contrastive positive pairs differ on sparse columns. The probability of perturbing a given element with table coordinates (r, c) is then pq if $\text{sparsity}(c) > \psi_b$ and p otherwise. The values ψ_u, ψ_b, p, q are chosen via hyperparameter tuning.

4.1.2 LOSS FUNCTIONS

Depending on the dataset, we utilize one of the following loss functions: SimCLR [4], VICReg [32], C3 [18], or our own loss function ContVICReg to create representations for the sake of applying FedSOM.

For each batch x , ContVICReg is defined as $\text{SimCLR}(x) + \sum_{C_j \in \mathcal{C}_z} \mathcal{L}_{\text{vic}}(C_j)$, where \mathcal{C}_z is the set of clusters obtained by training HDBSCAN on the pre-projection head representations z .

4.2 UMAP ENCODER

For the sake of testing FedSOM on embeddings created independently of any sort of SOM structure, we leverage UMAP on smaller datasets. We train separate UMAP encoders for categorical and numerical columns. A third UMAP encoder is then trained on the concatenation of the output from the first two. HDBSCAN is leveraged for optimization by maximizing normalized mutual information between the given cluster labels and the labels learned by HDBSCAN.

Datasets are preprocessed by Z -score normalization applied to the nonzero rows of each numerical column to preserve the sparsity of the original dataset.

4.3 SELF ORGANIZING MAP

The Self-Organizing Map update rule is given by $w_i^{s+1} = w_i^s + \theta(\beta, i)\alpha(v - w_i^s)$, where v is the sample, θ is the neighborhood kernel function, β is the index of the best matching unit (node) in the som grid, i is the index of the node being updated, s is the current iteration, α is the learning rate, w_i^s, w_i^{s+1} are the old and new weight vectors for node i , respectively. The neighborhood function is given by $\theta(\beta, i) = \exp(-\|\tau[i] - \tau[\beta]\|_2 / 2\sigma^2)$, where τ is the tensor of node coordinates. See Algorithm 2 for details.

The update rule for FedSOM is similar, but differs in that the training corpus consists of the weights of the several SOMs.

4.3.1 TENSOR DIMENSIONALITY

The ability of the SOM to partition a neighborhood of representation space into distinct clusters is a function of the number of nodes neighboring a given node in the tensor. The maximum number of distinct clusters neighboring a given cluster is equal to $3^{\text{dim}(\tau)} - 1$, making train time exponential in the number of tensor dimensions. We thus limited our experiments to dimensions two and three.

4.4 THE FEDERATED SOM

The Federated Self Organizing Map is a natural extension of the SOM, as it consists of nothing more than a set of disparate SOMs along with an additional *meta* SOM trained on the weight vectors of the individual SOMs. The original dataset D is partitioned by i.i.d. sampling to ensure that each SOM is trained on a subset drawn from the same distribution as the original dataset.

Training is performed in a federated way in the sense that separate SOM models are learned for each partition of the original data.

Algorithm 1: Federated Self-Organizing Map

Result: $\tau_j, W_j = \text{som}_j(D_j), \tau_m, W_m = \text{som}_m(\bigsqcup W_j)$ where j indexes the i.i.d. sampled partition of the dataset D , and som_m is the *meta* SOM operating on the weight vector sets W_j of the several SOMs.

Data:

- D : the dataset
- N_s : number of SOMs
- Remaining parameters as in Algorithm 1, uniform across $\{\text{som}_j\}$

Partition $D = \bigsqcup D_j$ into N_s subsets by i.i.d. sampling;

for j *in* $[1, \dots, N_s]$ **do**

 | train $\tau_j, W_j = \text{som}_j(D_j)$

end

train $\tau_m, W_m = \text{som}_m(\bigsqcup W_j)$

4.4.1 THE FEDSOM GRAPH

We construct a graph from the SOM tensor in order to create a structure within which any two clusters or any two samples may be interpolated. Weights are defined so as to facilitate interpolation between neighboring nodes corresponding to high-quality clusters, i.e., clusters with low dispersion and high pairwise separation.

The nodes of the graph correspond to the nodes of the tensor and two nodes are connected by an edge in the graph if and only if the nodes are neighbors in the tensor. Two nodes with coordinates $(n_1, \dots, n_{\dim(\tau)}), (m_1, \dots, m_{\dim(\tau)})$ are neighbors if and only if $\sum_j |n_j - m_j| = 1$.

The Calinski-Harabazs score is given by

$$\kappa = \frac{(N - k) \sum_j^k n_j \|\mathbf{c}_j - \mathbf{c}_{\text{total}}\|^2}{(k - 1) \sum_j^k \sum_{x_i \in C_j} \|x_i - \mathbf{c}_j\|^2}, \quad (1)$$

where N is the number of samples, k is the number of clusters, n_j is the number of data points in cluster j , \mathbf{c}_j is the centroid of cluster j , $\mathbf{c}_{\text{total}}$ is the centroid of the entire dataset, and C_j is the set of data points belonging to cluster j .

Weights are assigned to the edges based on the Calinski-Harabazs score computed from the embeddings attached to the two nodes bounding the given edge. The weight assigned to the edge $e_{uv} := e(u, v)$ is

$$\omega(e_{uv}) = \begin{cases} 1 - (\kappa / (\kappa + 1)) & \text{if } |\mathbf{c}_u|, |\mathbf{c}_v| > 2 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where κ is computed only from the samples whose cluster labels are given by the nodes u, v , and $\mathbf{c}_u, \mathbf{c}_v$ are the clusters corresponding to nodes u, v , respectively.

Edge weights are normalized so as to sit within $[0, 1)$, as the Calinski-Harabazs score is unbounded above. Because edge weight corresponds to cost in the utilized version of Dijkstra’s algorithm, we subtract the normalized score from 1 as large κ indicates high cluster separation and low intra-cluster dispersion.

4.5 INTERPOLATION

Samples $\mathbf{x}^{(i)}, \mathbf{x}^{(j)}$ are mapped via the encoder E to representation vectors $v^{(i)}, v^{(j)} = E(\mathbf{x}^{(i)}), E(\mathbf{x}^{(j)})$, and finally to nodes \mathbf{n}_i and \mathbf{n}_j in the SOM tensor via cluster assignment. The optimal interpolative path is discovered via Dijkstra’s algorithm. See Figure 2 for a visual representation.

The smoothness of the interpolation is a function of the σ parameter in the self-organizing map, as this parameter determines how finely the map partitions the set of encodings into clusters. The self-organizing map can be retrained with a smaller σ if smoother interpolations are desired.

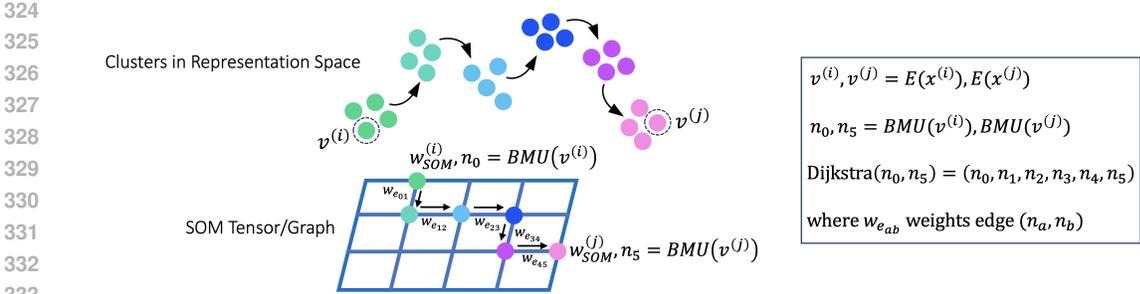


Figure 2: Interpolation between the samples $x^{(i)}, x^{(j)}$ within representation space \mathcal{X} is optimized within the SOM grid by running Dijkstra’s shortest path algorithm between the best matching unit weight vectors $w_{SOM}^{(i)}, w_{SOM}^{(j)}$ corresponding to the samples $v^{(i)}, v^{(j)}$, respectively.

Because the nodes in the SOM index weight vectors that do not necessarily correspond to actual embeddings, we must perform a nearest neighbors query to recover a representative embedding. For the data considered here, we can compute brute-force nearest neighbors. If operating at scale, this can be accomplished with a vector database algorithm such as FAISS or SCANN.

5 EXPERIMENTS

Training FedSOM on a given dataset is a multi-step process. Hyperparameter search is performed for embedding creation (NN and UMAP) and separately for SOM and FedSOM training via the tree-structured Parzen estimator in Optuna. Full hyperparameter ranges for every variable and configs containing optimal hyperparameters for both encoders, SOM, and FedSOM models are available in the code. We experiment on a variety of tabular datasets of widely varying type, with a focus on security datasets, including Ember and Sorel20M. Every FedSOM model consists of at least two constituent SOMs. We compare SOM and FedSOM unsupervised clustering performance on both UMAP and encoder-generated representations by computing normalized mutual information (NMI) and adjusted rand score (ARS). Each MNIST-like image contains a caption giving label as well as graph coordinates. Malicious binary file moduli spaces for three malware datasets show how various malware families relate to each other from the point of view of the leveraged parser.

Table 1: Comparison of Methods on MNIST and Fashion-MNIST Datasets

Method	NMI	
	MNIST	Fashion-MNIST
k-means	0.541 ± 0.001	0.545 ± 0.000
minisom	0.342 ± 0.012	0.475 ± 0.002
GB-SOM	0.519 ± 0.005	0.514 ± 0.004
VQ-VAE	0.409 ± 0.065	0.517 ± 0.002
no_grads	0.001 ± 0.000	0.018 ± 0.016
gradcopy	0.436 ± 0.004	0.444 ± 0.005
SOM-VAE	0.594 ± 0.004	0.590 ± 0.003
FedSOM	0.883	0.613

6 CONCLUSION

A self-organizing map trained in a federated way can effectively cluster both simple image data as well as tabular cybersecurity data. FedSOM trained on parsed malicious binaries can also serve as a moduli space of malware families in which family similarity is presented in terms of proximity within the organizing tensor. This moduli space can be leveraged for interpolation between individual samples and between entire clusters via the underlying weighted graph structure, which

Table 2: Performance of SOM and FedSOM on MNIST-like and Security Datasets

Dataset	UMAP		Encoder		Model
	NMI	ARS	NMI	ARS	
MNIST-like Datasets					
mnist	0.889	0.872	0.580	0.349	SOM
	0.883	0.824	0.574	0.398	FedSOM
fashionmnist	0.684	0.515	0.270	0.020	SOM
	0.613	0.483	0.194	0.066	FedSOM
chars74k	0.677	0.333	0.485	0.072	SOM
	0.631	0.303	0.332	0.092	FedSOM
cifar10	0.095	0.047	0.130	0.004	SOM
	0.113	0.058	0.073	0.040	FedSOM
emnist	0.736	0.537	0.447	0.062	SOM
	0.746	0.564	0.390	0.236	FedSOM
kuz	0.602	0.380	0.346	0.043	SOM
	0.610	0.402	0.322	0.141	FedSOM
notmnist	0.568	0.322	0.447	0.163	SOM
	0.596	0.475	0.361	0.143	FedSOM
quickdraw	0.602	0.380	0.346	0.043	SOM
	0.610	0.402	0.322	0.141	FedSOM
slmnist	0.568	0.322	0.447	0.163	SOM
	0.596	0.475	0.361	0.143	FedSOM
Security Datasets					
ember	0.356	0.112	0.662	0.548	SOM
	0.327	0.120	0.551	0.570	FedSOM
ccc	0.333	0.059	0.295	0.071	SOM
	0.243	0.014	0.236	0.076	FedSOM
pdfmalware	0.291	0.239	0.132	0.027	SOM
	0.209	0.120	0.093	0.076	FedSOM
sorel	0.261	0.046	0.373	0.029	SOM
	0.076	0.021	0.217	0.054	FedSOM
syscalls	0.321	0.042	0.286	0.020	SOM
	0.166	0.059	0.181	0.082	FedSOM
syscallsbinders	0.336	0.035	0.312	0.053	SOM
	0.233	0.069	0.190	0.073	FedSOM

Table 3: UMAP shows superior performance generating representations amenable to both the single SOM and FedSOM models on MNIST-like datasets. The encoder produces representations more amenable to SOM clustering for large security datasets. UMAP is still superior for smaller security datasets. Best NMI and ARS scores for each dataset are bold.

432
433
434
435
436
437
438
439
440
441
442
443
444
445
446

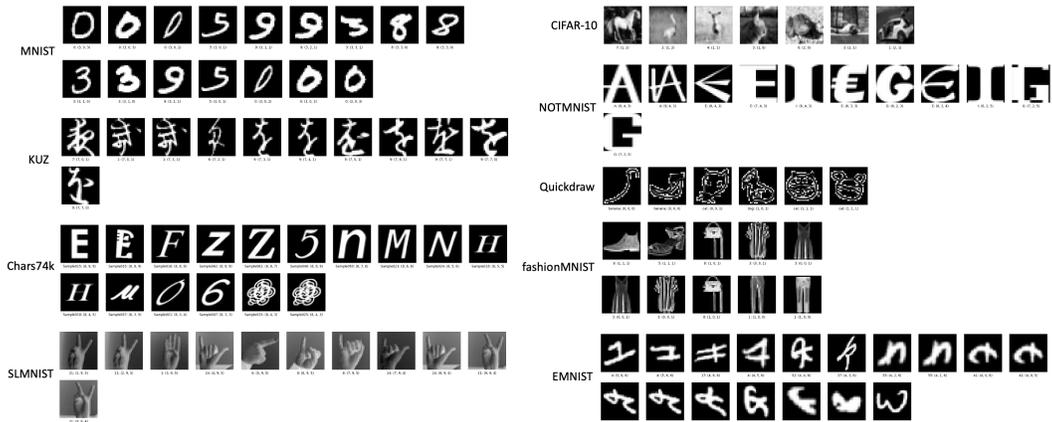


Figure 3: Interpolations occur through clusters in terms of actual datapoints rather than points generated from a latent space via a decoder.

447
448
449
450
451
452
453
454
455
456
457
458
459
460

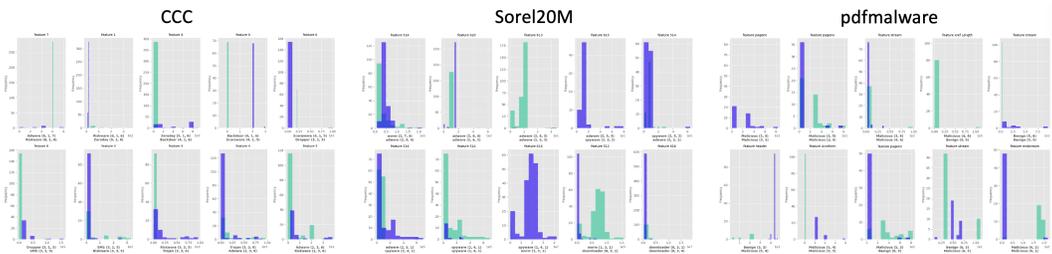


Figure 4: Interpolations through the malware datasets CCCS-CIC-AndMal2020 (CCC), Sorel20M (Ember parser), and CIC-Evasive-PDFMal2022 (PDFMalware). Sequences display the largest successive feature distribution changes along the interpolative path between clusters of various types. This provides analysts the ability to understand the evolution of various feature distributions as a given file class is interpolated into another.

461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480

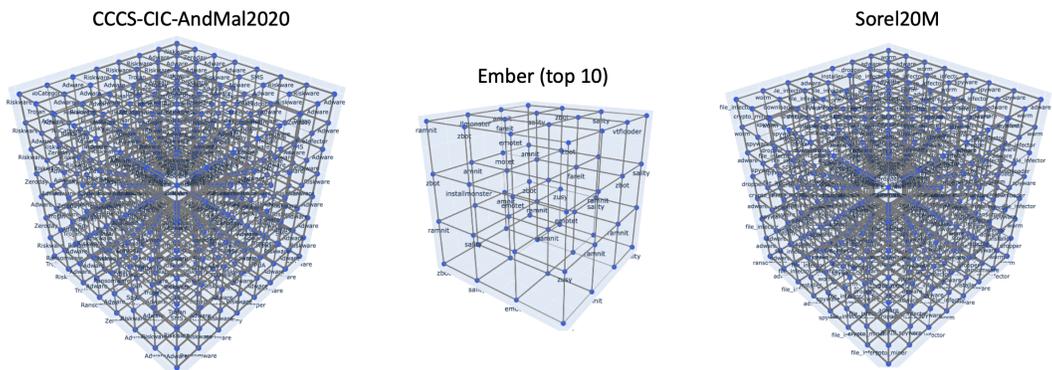


Figure 5: We present discrete moduli spaces for CCCS-CIC-AndMal2020 (CCC), Sorel20M (Ember parser), and Ember. Thicker edges indicate lower cost for the purpose of Dijkstra’s algorithm. The moduli space allows analysts to understand the relationships between various file and malware classes, as well as perform nearest neighbors search conditioned on nodes neighboring the node of inquiry.

486 allows for a deeper understanding of feature distribution changes as the interpolative path is tra-
487 versed between classes. The samples obtained via interpolation in this way are also faithful to the
488 data generating process, as all such samples belong to the given dataset.
489

490 REFERENCES

- 491 [1] Balestriero, R. & Pesenti, J. & LeCun, Y. (2021). Learning in High Dimension Always Amounts
492 to Extrapolation. arxiv preprint arXiv:2110.09485 [cs.LG]
- 493 [2] Bardes, A. & Ponce, J. & LeCun, Y. (2022). VICReg: Variance-Invariance-Covariance Regular-
494 ization for Self-Supervised Learning. *International Conference on Learning Representations*
495
- 496 [3] Berrada, L. & Zisserman, A. & Kumar, P. (2019). Training Neural Networks for and by Interpo-
497 lation. arxiv preprint arXiv:1906.05661 [cs.LG]
- 498 [4] Chen, T. & Kornblith, S. & Norouzi, M. & Hinton, G. (2020). A Simple Framework for Con-
499 trastive Learning of Visual Representations. *Proceedings of the 37th International Conference on*
500 *Machine Learning*, Vienna, Austria, PMLR 119, 2020.
- 501 [5] Dang, Z. & Deng, C. (2021). Doubly Contrastive Deep Clustering. arxiv preprint
502 arXiv:2103.05484 [cs.CV]
- 503 [6] Darlow, L.N. & Storkey, A. (2020). DHOG: Deep Hierarchical Object Grouping. arXiv preprint
504 arXiv:2003.08821 [cs.CV]
- 505 [7] Dokonal, W. & Hirschberg, U. & Wurzer, G. (2023). From Linear to Manifold Interpolation:
506 Exemplifying the paradigm shift through interpolation. *Proceedings of the 41st Conference on Ed-*
507 *ucation and Research in Computer Aided Architectural Design in Europe* 2:419–429
- 508 [8] Fogel, S. & Averbeck-Elor, H. & Cohen-Or, D. & Goldberger, J. (2019). Clustering-Driven
509 Embedding with Pairwise Constraints. *IEEE Computer Graphics and Applications*, 39:16–27
- 510 [9] Fortuin, V. & Huser, M. & Locatello, F. & Strathmann, H. & Ratsch, G. (2019). SOM-VAE: In-
511 terpretable Discrete Representation Learning on Time Series. *International Conference on Learning*
512 *Representation*
- 513 [10] Goppert, J. & Rosenstiel, W. (1993). Topology-Preserving Interpolation in Self-organizing
514 Maps. *Proceedings of NeuroNimes* 93 425-434, EC2, Nanterre, France.
- 515 [11] Goppert, J. & Rosentiel, W. (1997). The Continuous Interpolating Self-organizing Map. *Neural*
516 *Processing Letters* 5, pages 185-192 (1997).
- 517 [12] Huang, J. & Gong, S. (2021). Deep Clustering by Semantic Contrastive Learning. *British*
518 *Machine Vision Conference*
- 519 [13] Knopf, G. & Sangole, A. (2004). Interpolating Scattered Data using 2D Self-organizing Feature
520 Maps. *Graphical Models* 66, Issue 1, 50-69.
- 521 [14] Li, F. & Qiao, H. & Zhang, B. (2018). Discriminatively Boosted Image Clustering with Fully
522 Convolutional Auto-Encoders. *Pattern Recognition*, 83:161–173
- 523 [15] Miyato, T. & Koyama, M. (2018). cGANs with Projection Discriminator. *International Con-*
524 *ference on Learning Representations*
- 525 [16] Rezaei, M. & Dorigatti, E. (2022). Joint Debaised Representation Learning and Imbalanced
526 Data Clustering. arXiv preprint arXiv:2109.05232.
- 527 [17] Rezaei, M. & Dorigatti, E. & Rugamer, D. & Bischl, B. (2022) Joint Debaised Representa-
528 tion Learning and Imbalanced Data Clustering. *IEEE International Conference on Data Mining*
529 *Workshops (ICDMW)*
- 530 [18] Sadeghi, M. & Hojjati, H. (2022). C3: Cross-instance guided Contrastive Clustering. arXiv
531 preprint arXiv:2211.07136.
- 532 [19] Shah, S. & Koltun, V. (2018). Deep Continuous Clustering. arXiv preprint arXiv:1803.01449
533 [cs.LG]

- [20] Shen, Z. & Liu, Z. & Savvides, M. & Darrell, T. & Xing, E. (2020). Un-mix: Rethinking Image Mixtures for Unsupervised Visual Representation Learning. *AAAI Conference on Artificial Intelligence*
- [21] Shukla, S.N. & Marlin, B.M. (2019). Interpolation-Prediction Networks for Irregularly Sampled Time Series. *International Conference on Learning Representations*
- [22] Sundareswaran, R. & Herrera-Gerena, J. & Janessari, A. (2021). Cluster Analysis with Deep Embeddings and Contrastive Learning. arXiv preprint arXiv:2109.12714 [cs.LG]
- [23] van den Oord, A. & Vinyals, O. & Kavukcuoglu, K. (2017). Neural Discrete Representation Learning. *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA.
- [24] Xie, J. & Girshick, R. & Farhadi, A. (2016). Unsupervised Deep Embedding for Clustering Analysis. *Proceedings of the 33rd International Conference on Machine Learning 48*, pp. 478–487.
- [25] Yang, B. & Fu, X. & Sidiropoulos, N.D. & Hong, M. (2017). Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering. *Proceedings of Machine Learning Research* ISSN: 2640-3498
- [26] Yin, H. & Allinson, N. (1999). Interpolating self-organizing map (iSOM). *Electronics Letters* **35**(19):1649-1650
- [27] Yoon, J. & Zhang, Y. & Jordan, J. & van der Schaar, M. (2020). VIME: Extending the Success of Self- and Semi-supervised Learning to Tabular Domain. *Advances in Neural Information Processing Systems 33*
- [28] Zhang, D. & Sun, Y. & Eriksson, B. & Balzano, L. (2017). Deep Unsupervised Clustering Using Mixture of Autoencoders. arXiv preprint arXiv:1712.07788 [cs.LG]
- [29] Zhong, H. & Chen, C & Jin, Z. & Hua, X. (2020). Deep Robust Clustering by Contrastive Learning. arXiv preprint arXiv:2008.03030 [cs.CV]

A APPENDIX / SUPPLEMENTAL MATERIAL

A.1 NEURAL NETWORK ENCODER ARCHITECTURE

Let W_i^c , W_j^n , and W_k^j represent the fully connected layers for the categorical, numerical, and joined representations, respectively. The encoder is given by xs

$$\text{representation} = \text{LReLU}(W_1^j([z_{\text{num}}, z_{\text{cat}}])), \quad (3)$$

where we pass x_{num} through two separate layers $f^{(1)} = W_1^n \mathcal{B}$, $f^{(2)} = \sigma(W_2^n \mathcal{B})$ and then a final layer $f^{(3)} = LW_3^{\text{num}}$, where \mathcal{B} is batch norm and L is layer norm. The portion of the representation constructed solely from numerical columns is then given by $z_{\text{num}} = f^{(3)}(f^{(1)}(x_{\text{num}}) \odot f^{(2)}(x_{\text{num}}))$.

The categorical portion of the representation is given by $z_{\text{cat}} = g^{(3)}(g^{(1)}(x_{\text{cat}}) \odot g^{(2)}(x_{\text{cat}}))$, where $g^{(1)} = W_1^c \mathcal{E}$, $g^{(2)} = \sigma(W_2^c \mathcal{E})$, and $g^{(3)} = LW_3^c$.

We apply the dataset-dependent loss function to the embeddings produced by the projection head

$$\text{embedding} = \text{LReLU}(W_2^j(\text{representation})) \quad (4)$$

A.2 THE SELF-ORGANIZING MAP

Algorithm 2: Self-Organizing Map

Result: $\tau, W = \text{som}(D)$, where τ is the organizing tensor τ with integer coordinates and W the set of weight vectors, one for each node of τ

Data:

- D : the dataset
- s : the current iteration
- λ : number of epochs
- η : number of iterations per epoch
- v : target input data vector
- i : index of a node in the map
- w_i : weight vector attached to node i
- β : index of the best matching unit (BMU)
- $\theta(\beta, i, s)$: the kernel function
- $\alpha(s)$: learning rate

```

611 for  $s$  in  $[1, \dots, \lambda]$  do
612   Sample batch  $x \sim D$ ;
613   for  $t$  in  $[1, \dots, \eta]$  do
614     for  $v$  in  $x$  do
615       Find best matching unit  $i$  for  $v$ ;
616        $w_i^{t+1} := w_i^t + \theta(\beta, i) \alpha(v - w_i^t)$ ;
617     end
618   end
619 end

```

A.3 DIJKSTRA'S ALGORITHM

Algorithm 3: Dijkstra's Algorithm

Require: G (weighted graph), u (source node), v (destination node)

```

626 1: Initialize distances with infinity for all nodes except  $u$ 
627 2:  $\text{distances}[u] \leftarrow 0$ 
628 3: Initialize priority_queue with  $(0, u)$  {Priority queue with (distance, node)}
629 4: while priority_queue is not empty do
630 5:   (current_distance, current_node)  $\leftarrow$  extract min from priority_queue
631 6:   if current_node equals  $v$  then
632 7:     return  $\text{distances}[v]$  {Shortest path found}
633 8:   end if
634 9:   if current_distance >  $\text{distances}[\text{current\_node}]$  then
635 10:    continue
636 11:   end if
637 12:   for each neighbor of current_node do
638 13:      $\text{distance} \leftarrow \text{current\_distance} + \text{weight}(\text{current\_node}, \text{neighbor})$ 
639 14:     if  $\text{distance} < \text{distances}[\text{neighbor}]$  then
640 15:        $\text{distances}[\text{neighbor}] \leftarrow \text{distance}$ 
641 16:       insert  $(\text{distance}, \text{neighbor})$  into priority_queue
642 17:     end if
643 18:   end for
644 19: end while
645 20: return None {No path found}

```

Early experiments were run with $\omega(e_{uv}) = \kappa/(1 + \kappa)$ based on an assumption that interpolation between close and disperse clusters would result in smoother transitions than interpolation between

concentrated and separated clusters. However, pairwise disperse and close clusters were often both ontologically and structurally heterogenous, making interpolation less meaningful.

A.4 LOSSES

A.4.1 C3

The C3 loss (fill in reference) is given by

$$\mathcal{L}_{C3} = \frac{1}{2N} \sum_{i=1}^N (\tilde{\ell}_i^a + \tilde{\ell}_i^b)$$

where $\tilde{\ell}_i^a$ (sim $\tilde{\ell}_i^b$) are given by

$$\tilde{\ell}_i^a = \log \frac{\sum_{k \in \{a,b\}} \sum_{j=1}^N \mathbf{1}\{z_i^{a \top} z_j^k \geq \zeta\} \exp(z_i^{a \top} z_j^k)}{\sum_{k \in \{a,b\}} \sum_{j=1}^N w_{ij}^k \exp(z_i^{a \top} z_j^k)},$$

and

$$w_{ij}^k = \frac{\exp(\Gamma(1 - |z_i^{a \top} z_j^k|))}{\sum_{k \in \{a,b\}} \sum_{j=1}^N \exp(\Gamma(1 - |z_i^{a \top} z_j^k|))}$$

A.4.2 CONTRASTIVE LOSS

Of the many varieties of contrastive loss, we leverage the form described in (ref Hinton or SAINT), i.e.,

$$\mathcal{L}_{\text{cont}} = - \sum_{i=1}^m \log \frac{\exp(z_i \cdot z'_i / \tau)}{\sum_{k=1}^m \exp(z_i \cdot z'_k / \tau)}, \quad (5)$$

where z_i, z'_i are the representations of inputs x_i and its perturbation x'_i , respectively.