ABSTRACTING FEATURES USING CLUSTER-EMBEDDING DECOMPOSITION

MATS RESEARCH PROPOSAL

Keira Wiechecki[1]

[1]Center for Genomics & Systems Biology, New York University, kaw504@nyu.edu

February 6, 2024

ABSTRACT

I propose a new approach to understanding the feature space of a model based on denoising. I further propose that the feature space can be decomposed into two subspaces with intuitive interpretations: a cluster space and an embedding space. I term this KE decomposition. This approach suggests a model's latent ontology can be derived using a paired compressor and a partitioner. I present a novel deep clustering architecture based on this principle. Preliminary analyses reveal several questions warranting further inquiry.

1 Threat Model

We don't currently know how to specify a robust goal for a model. Because goal space is large, we should expect goal misgeneralization as the default outcome. We can't currently look inside a model and make any conclusions about its goals. Mechanistic interpretability (mechinterp) is rapidly improving but is still nowhere near being scalable to frontier models.

To a large extent, deep learning models seem to learn universal features irrespective of architecture. This implies the existence of natural abstractions, which would be a very convenient shortcut for auditing a model. Unfortunately, the abstractions a model learns seem highly sensitive to choice of training data, and are thus not necessarily universal in the limit. This means that we cannot expect the abstractions learned by a weak model to remain robust in a more powerful model.

2 Theory of Change

2.1 My macro-scale strategy

I see the fundamental problems for alleviating this threat as

1. formalize the notion of a model's latent concept space
2. distill a model's latent concept space
3. enforce decomposability of latent concept space
4. enforce robustness of existing concepts as the dimensionality of the latent concept space increases

My justification for this line of research is

1. Almost every alignment proposal is limited by the ability to point at a concept.
2. According to singular learning theory (SLT), any sufficiently parameterized model trained on the same data will converge on the same latent representation of the data.
3. Almost every alignment proposal becomes much more tractable if we can reason about features in isolation.

2.2 My working theory of ontology

I refer to any mapping of data to abstractions as an *ontology*.

Rapid progress in scaling LLMs has made it increasingly clear that data rather than compute is the limiting factor on capabilities. This was not at all obvious in hindsight and if we survive the coming decades it will profoundly reshape every aspect of philosophy. From an alignment perspective, the universality of deep learning independent of architecture is strong evidence for the natural abstraction hypothesis. This is very good news! We have evidence that for any given data there is an "objective" ground truth.

2.2.1 Features and concepts

Mechinterp has been very successful at decomposing models into linear *features*. Features have proven to be a powerful way of abstracting over activation space. In other words, mechinterp aims to construct *an ontology of activation space*.

However, this is not the only model ontology we care about. Many of the concepts humans care about are discrete or stochastic. For humans, it's unintuitive to think of every image as containing a sliding "car vector". Humans classify images as containing a car. Note that there is a subtle difference between these two kinds of abstraction. The former is abstracting over *the pixels in an image* to obtain a characteristic measurement of how "car" it is. The latter is abstracting over *the images in a set* to obtain a likelihood (if we assume differentiability) that the image is in the "contains car" category. I will refer to the former as *features* and the latter as *concepts*. If we have some data $X \in \mathbb{R}^{m \times n}$, where n is the number of samples and m is the number of measurements from each sample, features are abstractions over m and concepts are abstractions over n.

2.3 Clustering and latent classification

I propose that this "ontology over concepts" is best expressed in terms of clustering. Neural networks are continuous, but can learn discrete tasks. SLT suggests a possible mechanism for this behavior, but a systematic way of characterizing discretization in the wild remains an open problem. Though SLT provides a rich thermodynamic description of learning, an alternative metaphor may be more intuitive. Consider a latent classifier contained in the model. Subsequent computations can be conditional on the classification

Additional information on the algorithm are in Appendix **??**.

2.4 SAEs and abstraction

Mechinterp is best charaterized as understanding the abstractions used by a model. Sparse autoencoders (SAEs) have become a powerful tool for identifying monosemantic features. Though SAE features are significantly more interpretable than raw activations, interpreting them is still a daunting task. A high quality library for a single layer can contain over 25,000 features. It would be a substantial interpretability advance if we could impose structure on these features.

Existing research on SAEs suggests that features assemble into clusters representing higher order features. This suggests the possibility of a hierarchical organization of features. Characterizing features requires carefully balancing feature sparcity and accurately reconstructing the model activations. We would like to be able to selectively zoom in on features.

3 Plan

3.1 Overview

3.1.1 Research Questions

I attempt to determine whether an autoencoder combined with a self-supervised classifier produces more interpretable features than a naïve SAE. I perform clustering with multiple architecture variants to assess stability of the latent classifications. I try to asses whether a toy model trained on the same data but with different objectives learns the same abstractions.

3.1.2 A toy model of feature splitting

I trained a MNIST classifier with a 3 neuron bottleneck. I hypothesized that since the model had more labels than neurons in the bottleneck, features would be represented in superposition. I refer to this as the "outer model". Details are in Appendix **??**

3.1.3 Feature splitting

I train three "inner models" to split the bottleneck activations into sparse features. All of them have an embedding size of 27. The PSAE is identical to the SAE except for containing a partitioner submodel. Details are in Appendix **??**

3.1.4 Clustering

I introduce two novel architectures that combine feature splitting with clustering. See Appendix **??** for details. Both have identical clustering submodules with a maximum k of 12. They only differ is their encoders and (in the case of DeePWAK) decoders.

3.1.5 Cluster Enrichment

I performed a https://en.wikipedia.org/wiki/Hypergeometric_distribution#Hypergeometric_testhypergeometric test for pairwise enrichment of all classifications in each model.

3.2 Legend

[✓] **Completed**

[×] **Not started**

[+] **Started; needs work**

[−] **Remove or scale down**

3.3 Model Designs

3.3.1 Data

[✓] **MNIST**

[−] **Microscopy data** Requires too much context to explain. It's interesting to see how DeePWAK finds structure in a low quality data set but it's still a low quality data set.

3.3.2 Outer models

As a toy model of superposition, I trained a MNIST classifier with a 3 neuron information bottleneck. I will train a second model where the classifier component after the bottleneck is replaced with a decoder. In both cases the architecture is identical before the bottleneck, but the goal is different. I expect that both models will encode similar information in the bottleneck layer.

[✓] **Encoder**

[✓] **Classifier**

[×] **Decoder**

3.3.3 Inner models

[✓] **SAE** A minimal SAE implementation is available at https://github.com/kewiechecki/SAE

[✓] **PSAE** see above

[✓] **DeePWAK** see above

[−] **DeePWAKBlock** This can be relegated to a separate paper. It doesn't add much for the model I'm working with.

3.3.4 Inner model methods

[✓] **Encoding** This is just the

[✓] **Decoding**

[✓] **Clustering**

[✓] **Cluster centroids**

3.3.5 Training

[×] **Checkpoints** Save models at different stages of training

3.4 Visualizations

[+] **Loss**

[×] **Visualize cluster centroids throughout training**

3.4.1 Heatmaps

[×] **How to read** §4.1

[✓] **Embeddings** Fig. 2

[✓] **Clusters** Fig. 3

3.4.2 Hypergeometric test

[×] **How to read dot plot**

[✓] **Dot plot** Fig. 4

[×] **Remove outline scale from dot plot** This is confusing and doesn't add useful information.

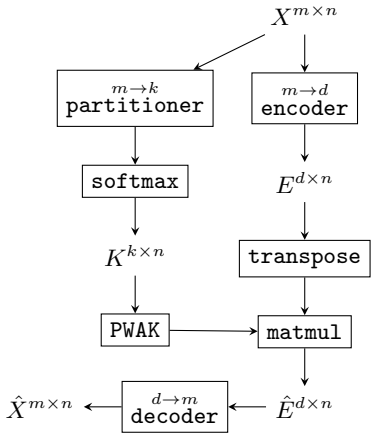
3.5 Distribution

[+] **Git repo** Code is divided between https://github.com/kewiechecki/DeePWAK and https://github.com/kewiechecki/SAE. These need to be unified.

3.5.1 Documentation

[+] **Model documentation**

[+] **Training documentation**



(a)

Figure 1: DeePWAK inference. See Appendix A for notation details.

[×] **Figure documentation**

[×] **Julia modules**

3.6 Publication

[✓] **LaTeXtemplate**

3.6.1 Illustrative figures

[×] **Outer models**

[✓] **DeePWAK flowchart** Fig. 1

[×] **PWAK example**

[+] **Diffusion example** Take out DEWAKSS example. Substitute example from actual data.

[+] **Figure legends**

3.6.2 Algorithms

[−] **noise2self**

[−] **PWAK**

3.7 Supplementary Publications

[×] **Theoretical foundations** Material cut from §2. Probably post to LessWrong.

[+] **DeePWAK** Split out all microscopy and multihead stuff to https://drive.google.com/file/d/1geDArkvUQVOp79dF9knNmO3iP-1BYGOD/view?usp=drive_link.

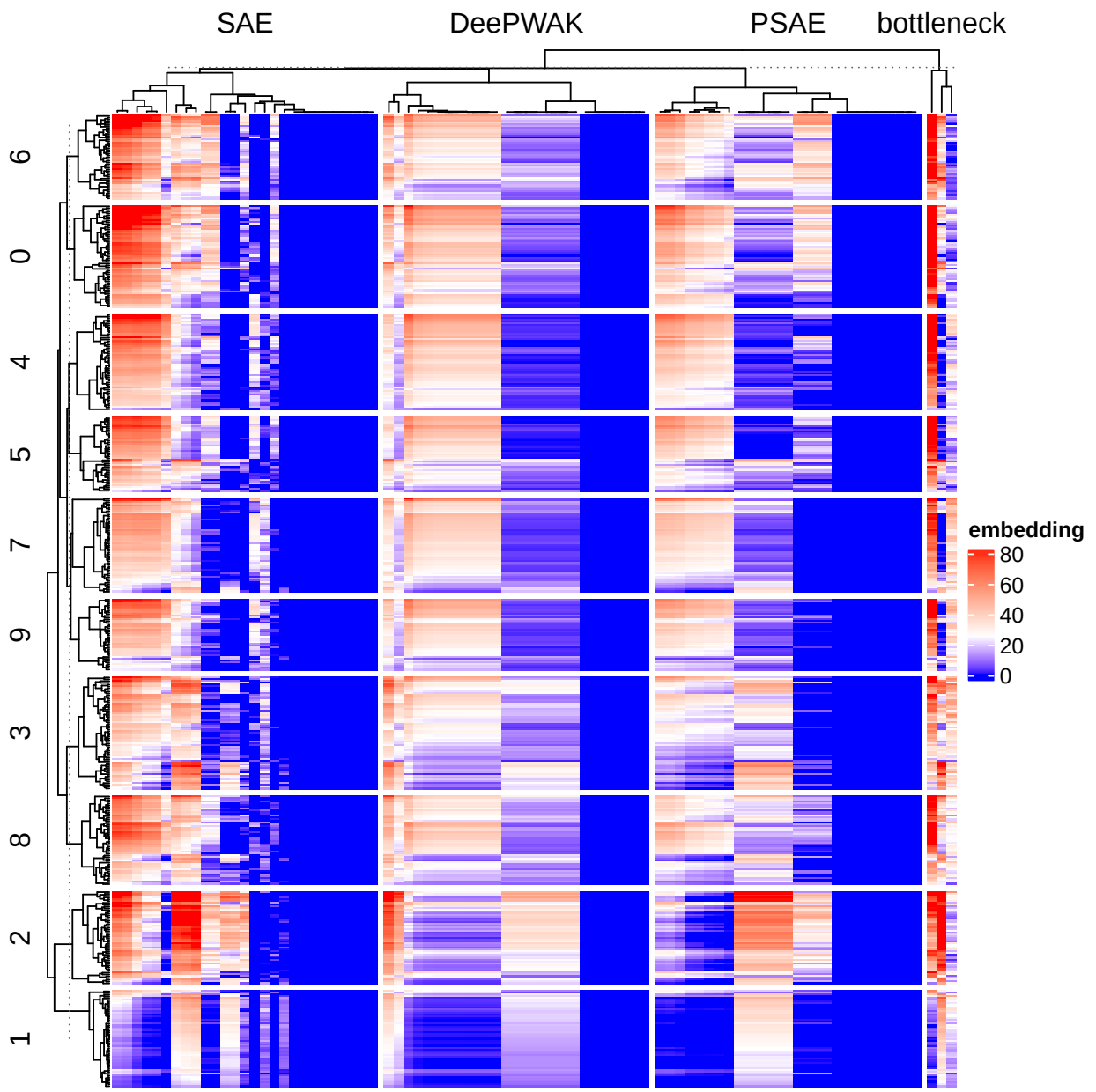


Figure 2: Feature splitting of bottleneck activations by different methods. There is a clear progression of sparcity from a naïve SAE, PSAE, and DeePWAK. Rows are split by training set label. Interestingly, adding a partitioner submodel seems to result in the same feature being copied by multiple embeddings.

4 Preliminary Results

4.1 How to read the figures

Heatmaps are ubiquitous in bioinformatics for visualizing very large data sets. In Fig. 2 and 3, rows are samples and columns are activations. Samples are split either by training label (Fig. 2) or predicted label (Fig. 3. Activations are split by model. For Fig. 2, these are the encoder activations. bottleneck gives the bottleneck activations for the outer model.

Even trained without a sparcity correction, `DeePWAK` finds a more sparse representation than the SAE or PSAE (Fig. 2, 3.

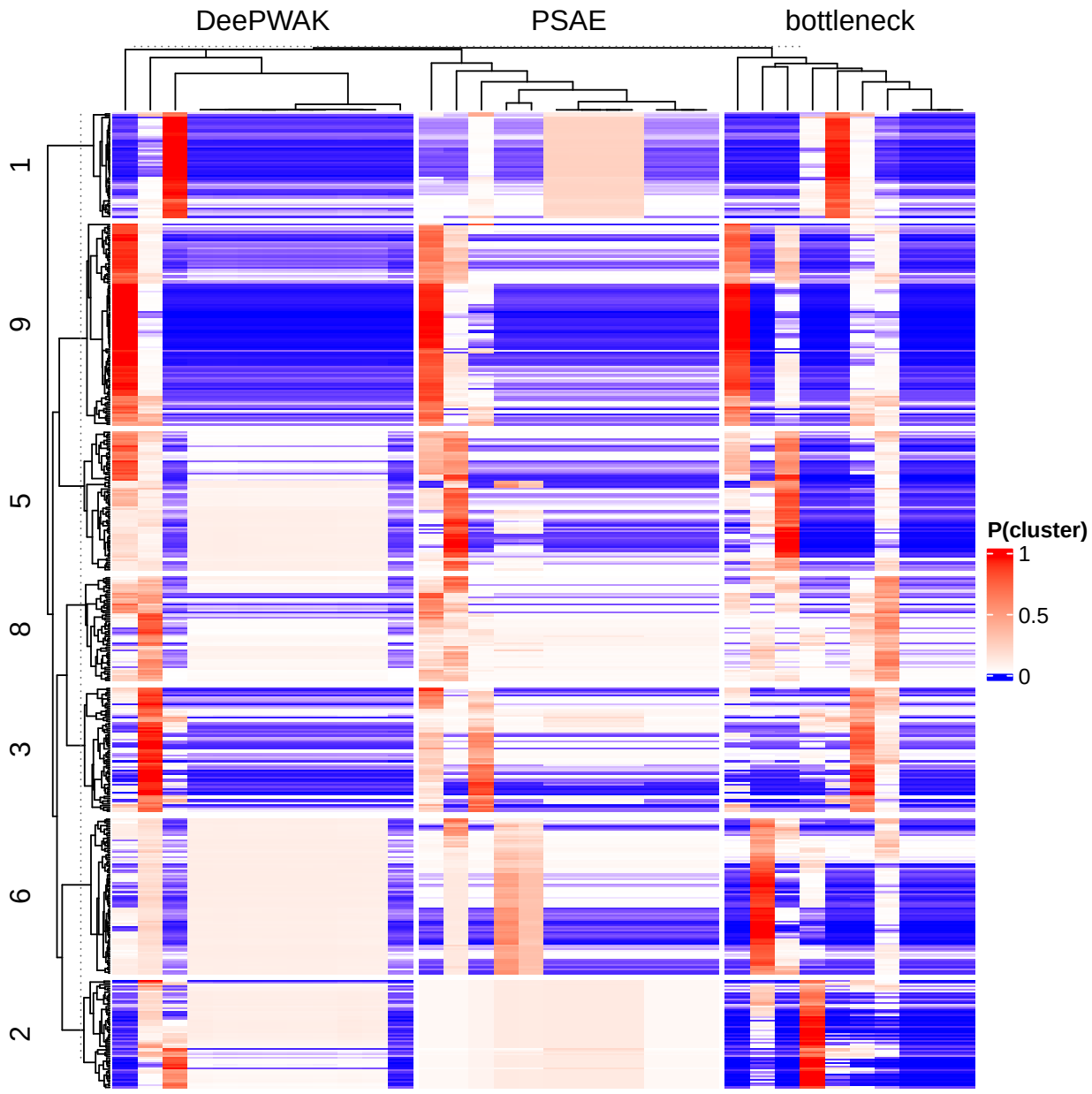


Figure 3: Both DeePWAK and PSEA learn sparse clusters. Rows are split by predicted label.

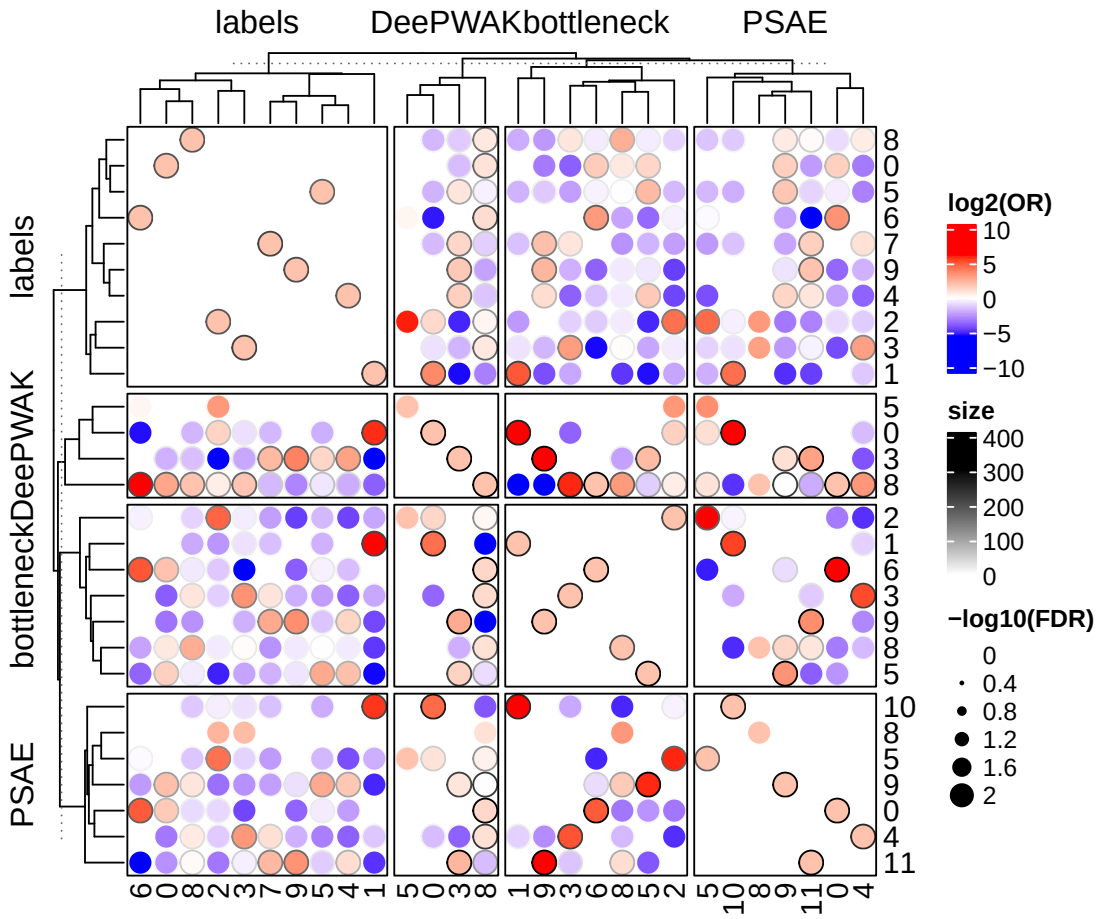


Figure 4: Hypergeometric test for enrichment of [rows] in [columns].

Both DeePWAK and PSAE seem to capture latent features from the bottleneck (Fig. 4. Strikingly, PSAE better predicts the model output than the model predicts the training labels. DeePWAK, on the other hand, appears to identify higher level features used by the model for classification. It reveals the model apparently grouping rounded (0,3,6,8) and pointed (4,5,7,9) digits. Even more curiously, these features appear *bisemantic*. Looking at DeePWAK 0, this cluster appears to regard 6 as "opposite" of 1. If we look back at Fig. 3, we can see the partitioner is much less confident classifying 6s than other digits. It's possible the model is using an internal logic of "I don't know what this is but it's definitely not a 1".

5 Plan

5.1 Can we split DeePWAK features?

`DeePWAK` seems to capture more general features than PSAE. It would be very useful if we could express high level features as compositions of lowe level features.

5.2 Why bisemanticity?

This is the biggest question I have. It shows up prominently in two very different data sets using different analyses. It suggests that monosemantic features may not be the most "natural" way to represent data. The previous analogy to PCA may be informative. If there is no privileged basis in feature space, it seems likely that features attempt to capture the vectors of highest variance.

5.3 Experiments on GPT2

Adapting DeePWAK to LLMs will present challenges. In its current form it performs poorly when the number of latent clusters is much larger than the minibatch size. A possible workaround is to add a dictionary of representative "platonic forms" that are appended to each minibatch.

Is a nonlinear decoder necessary?

How decomposable are clusters?

What makes a cluster "interpretable"?

Can we get a recurrent ontology?

References

[1] Joshua Batson and Loic Royer. *Noise2Self: Blind Denoising by Self-Supervision*. 2019. arXiv: 1901.11365 [cs.CV].

[2] Andreas Tjärnberg et al. "Optimal tuning of weighted kNN- and diffusion-based methods for denoising single cell genomics data". In: *PLOS Computational Biology* 17.1 (Jan. 2021), pp. 1–22. DOI: 10.1371/journal.pcbi.1008569. URL: https://doi.org/10.1371/journal.pcbi.1008569.

A Notation

We use lowecase Latin characters to denote scalars, boldface lowecase characters to denote vectors, and capital Latin characters to denote matrices. Subscripts indicate indices. Because we will mostly be working with matrices in $\mathbb{R}$, we abbreviate $X : \mathbb{R}^{m\times n}$ as $X^{m\times n}$. We use a circumflex to indicate a reconstruction of an input by a predictor. Lowercase Greek characters indicate the parameters of a model. Capital Greek letters indicate parameter spaces. Function names are in monospace.

$\overset{n\to m}{\texttt{function}}$ indicates a layer with input dimension n, output dimension m, and activation function `function`.

B Additional Background

B.1 Denoising the data explains the data

For a large class of denoising functions, it is possible to find optimal parameters using only unlabeled noisy data[1]. `noise2self` gives a near-maximally-general optimization target for hyperparameter search.

Let $J \in \mathcal{J}$ be independent partitions of noisy data X. Let $\mathcal{F}(\theta)$ be a family of predictors of X_J with tunable parameters $\theta \in \Theta$ that depends on its complement X_{J^C}

$$\hat{X}_J = \mathcal{F}(\theta)(X_{J^C}) \tag{1}$$

In other words, $\mathcal{F}$ predicts each data point X_J from some subset of the data excluding X_J.

The optimal θ is given by

(2)

B.2 Diffusion with weighted affinity kernels

`noise2self` is particularly useful for finding optimal parameters for generating a graph[2]. (see Appendix **??**) The adjacency matrix G of any graph can be treated as a transition matrix (or weighted affinity kernel) by setting the diagonal to 0 and normalizing columns to sum to 1. We call this the `WAK` function (Algorithm **??**).

For each value in data X, an estimate is calculated based on its neighbors in the graph. This can be expressed as matrix multiplication.

$$\hat{X} := \texttt{WAK}(G)X^\top \tag{3}$$

B.3 Partitioned weighted affinity kernels

Though DEWAKSS uses a k-NN graph, any adjacency matrix will do. A clustering can be expressed as a graph where points within a cluster are completely connected and clusters are disconnected.

Let $K^{k\times n}$ be a matrix representing a clustering of n points into k clusters. Let each column be a 1-hot encoding of a cluster assignment for each point. We can obtain a partition matrix $P \in \mathbb{R}^{n\times n}$ by what we'll call the *partitioned weighted affinity kernel* (`PWAK`) function.

$$\texttt{PWAK}(K) := \texttt{WAK}(K^\top K) \tag{4}$$

This lets us define a loss function

$$\mathcal{L}_{\texttt{PWAK}}(K, X) := \mathbb{E}[\texttt{PWAK}(K)X^\top - X]^2 \tag{5}$$

`PWAK` can be extended to soft cluster assignment, making it possible to learn K via SGD. We will refer to a model of this sort as a `Partitioner` to emphasize that while it returns logits corresponding to classifications, there are no labels on the training data.

There is no accuracy measure separate from the decoder loss. The partitioner simply tries to find the best P for minimizing loss of the decoded output.

The only hyperparameters are the maximum number of clusters, the neural net architecture, and the training hyperparameters. Because $PX^\top$ is $\mathcal{J}$-invariant, this classifier will converge on a solution less than the maximum k. Intuitions from transformers may be helpful in visualizing why this works. Informally, P can be equated to position-independent attention with data points as tokens and the batch size as the context window. Attentive readers may make a connection between masking the diagonal and BERT.

We can now train a model to classify unlabeled data into an undefined number of clusters with no prior distribution in $\mathcal{O}(n^2)$ time!