Compositional Clustering: Applications to Multi-Label Object Recognition and Speaker Identification

Zeqian Li[a], Xinlu He[a], Jacob Whitehill[a,*]

[a] *Worcester Polytechnic Institute, 100 Institute Road, 01609, Worcester, USA*

Abstract

We consider a novel clustering task in which clusters can have compositional relationships, e.g., one cluster contains images of rectangles, one contains images of circles, and a third (compositional) cluster contains images with both objects. In contrast to hierarchical clustering in which a parent cluster represents the *intersection* of properties of the child clusters, our problem is about finding compositional clusters that represent the *union* of the properties of the constituent clusters. This task is motivated by recently developed few-shot learning and embedding models [1, 19] can distinguish the label *sets*, not just the individual labels, assigned to the examples. We propose three new algorithms – Compositional Affinity Propagation (CAP), Compositional k-means (CKM), and Greedy Compositional Reassignment (GCR) – that can partition examples into coherent groups and infer the compositional structure among them. We show promising results, compared to popular algorithms such as Gaussian mixtures, Fuzzy c-means, and Agglomerative Clustering, on the OmniGlot and LibriSpeech datasets. Our work has applications to open-world multi-label object recognition and speaker identification & diarization with simultaneous speech from multiple speakers.

Keywords: Clustering Algorithms, Compositional Learning, Few-Shot Learning, Embedding Models, Speaker Diarization, Affinity Propagation

*Corresponding author

Email addresses: zli14@wpi.edu (Zeqian Li), xhe4@wpi.edu (Xinlu He), jrwhitehill@wpi.edu (Jacob Whitehill)

1. Introduction

We consider a new kind of clustering problem in which clusters have compositional structure, in the sense that each example in one cluster may exhibit the *union* of the properties found in another set of clusters. The goal is not just to partition the data into distinct and coherent groups, but also to infer the compositional relationships among the groups. This scenario arises in speaker diarization (i.e., infer who is speaking when from an audio wave) in the presence of simultaneous speech from multiple speakers [6, 36], which occurs frequently in real-world speech settings: The audio at each time t is generated as a *composition* of the voices of all the people speaking at time t, and the goal is to cluster the audio samples, over all timesteps, into sets of speakers. Hence, if there are 2 people who sometimes speak by themselves and sometimes speak simultaneously, then the clusters would correspond to the speaker sets $\{1\}$, $\{2\}$, and $\{1, 2\}$ – the third cluster is not a third independent speaker, but rather the composition of the first two speakers. An analogous scenario arises in open-world (i.e., test classes are disjoint from training classes) multi-label object recognition when clustering images such that each image may contain multiple objects from a fixed set (e.g., the shapes in Figure 1). In some scenarios, the *composition* function that specifies how examples are generated from other examples might be as simple as superposition by element-wise maximum or addition. However, a more powerful form of composition – and the main motivation for our work – is enabled by *compositional embedding models*, which are a new technique for few-shot learning.

Compositional embedding models: Standard (non-compositional) embedding models for few-shot learning such as FaceNet [28] and x-vector [29] have an embedding function f^{emb} (typically a neural network) that maps each example (e.g., image, audio clip) into an embedding space so that examples with the same label are mapped close together, and examples with different labels are mapped far apart. Compositional embeddings [1, 19] go a step further and are trained to separate not just individual labels, but entire *sets* of labels. As an example of how this is performed using the approach by [19], suppose an image collection contains some images of rectangles, some of circles, and some of both (see Figure 1). Then the embedding function f^{emb} would induce three clusters in the embedding space corresponding to $\{\text{rectangle}\}, \{\text{circle}\}$ and $\{\text{rectangle}, \text{circle}\}$. In addition to f^{emb}, compositional embedding models have a *composition* function g that

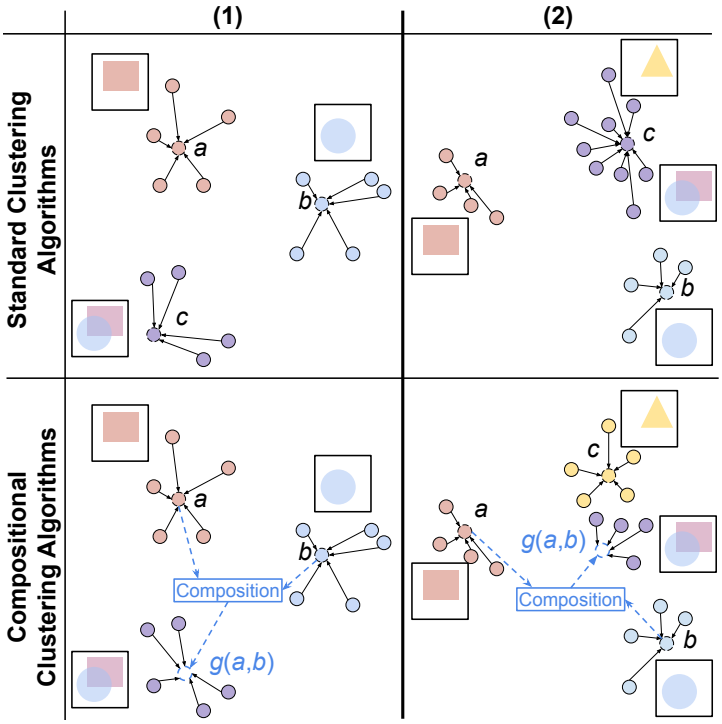

Figure 1: *Conceptual overview of our paper*: Scenario (1) shows clusters of images (containing rectangles, circles, or both) and their assigned exemplars (for exemplar-based methods) or centroids (for centroid-based methods) a, b, c, etc. Each arrow represents the assignment of an example to its cluster exemplar/centroid. Standard clustering algorithms such as k-means or Affinity Propagation detect 3 clusters that are independent of each other. Compositional clustering algorithms like CAP, CKM, and GCR can infer that each example in the bottom/purple cluster is *composed* (via g) of examples from clusters a & b. Scenario (2) illustrates how modeling compositionality can enable CAP and CKM to find purer clusters by not lumping the two sets of images (some with triangles, and some with rectangles & circles) together.

takes the embedding vectors x_a, x_b of two examples and computes a set relationship between them. For instance, $g(x_a, x_b)$ might return another vector in the same embedding space corresponding to where an example containing the *union* of the labels in the two inputs would lie – see Figure 1 (lower left). In particular, the training objective is for $g(x_a, x_b) \approx x_{ab}$, where x_{ab} is the embedding of an example containing *both* classes a and b. By applying it recursively (e.g., $g(x_a, g(x_b, x_c)) \approx x_{abc}$), the same function g can be used to estimate the embeddings of larger sets of examples as well. At test time, f^{emb} and g are used together (along with a support set of few-shot examples) to infer the *set* of labels in an example. Other recent works have explored a similar idea of training the embedding network for set operations such as union, difference, and containment [30, 37], or to use the embedding space to synthesize feature vectors with specific properties [13].

Compositional clustering methods: In this paper we present and evaluate three novel algorithms for tackling the "compositional clustering" problem: (1) **Compositional k-means (CKM)**, which is a centroid-based clustering method; (2) **Compositional Affinity Propagation (CAP)**, which is an exemplar-based method; and (3) **Greedy Compositional Reassignment (GCR)**, which can be used in tandem with any standard clustering algorithm. All three of these methods have the ability to assign each example to either a "singleton" cluster corresponding to a single class (e.g., a single speaker, or a single object) *or* to a "compositional" cluster corresponding to the union of multiple classes (e.g., a set of speakers, or a set of objects). CKM and CAP have the additional ability to harness the compositional structure of the data to partition them more accurately than is possible with standard clustering algorithms.

As a conceptual illustration, see Figure 1. In scenario (1) (left half of the figure), there are three sets of images – some contain circles, some contain rectangles, and some contain both. Standard clustering algorithms such as Affinity Propagation and k-means can separate the data correctly into three clusters. However, a compositional clustering algorithm such as CKM, CAP, or GCR can also infer that the cluster shown in purple in the bottom-left is actually a *compositional cluster* in which each example contains *both* objects from the first two clusters. Scenario (2) in the figure shows how modeling the compositionality can yield a more accurate partition: whereas standard clustering algorithms will lump together the images containing triangles with those containing

a composition of rectangles and circles, CAP and CKM can identify this relationship automatically and thereby obtain purer clusters.

General Workflow: Here is how a compositional clustering algorithm can be used for open-world object recognition, speaker diarization, and similar tasks: The first step is to (1) train a compositional embedding model [1, 19] with both an embedding function f^{emb} (e.g., with triplet loss, ArcFace loss [8], etc.) as well as a composition function g that computes the location in the embedding space corresponding to the *set union* of the classes represented in its two input embeddings. Note that g can be trained recursively [20] to enable the computation of set unions of *arbitrary* size; moreover, it needs to be trained only once and can then be reused. Next, (2) compute the embeddings of all the speaker utterances (or images) in the dataset; we denote the set of these embeddings as $\mathcal{X} = \{x_1, \ldots, x_n\}$. (3) Pass $\mathcal{X}$, as well as the composition network g, as input to the compositional clustering algorithm (CAP, CKM, or GCR). The clustering algorithm then (4) infers the cluster label – which could be either a *singleton* (a single speaker in isolation, or a single object appearing by itself) or a *set* (multiple speakers in simultaneous speech, or multiple objects co-occuring in an image) – of each example.

Contributions: (1) We consider the computational problem of clustering data with compositional structure, particularly as afforded by compositional embedding models, in the setting where (a) the test classes are disjoint from training classes, (b) each example can belong to multiple classes, and (c) no information about the test classes (neither a support set, nor a semantic description) is given. To our knowledge, this particular task has not been tackled previously. We also define a new accuracy metric, the Compositional Rand Index, for this problem. (2) We present three novel clustering algorithms – CAP, CKM, and GCR – that can partition data and infer their compositional structure automatically. (3) We illustrate how these new methods can infer the clusters, as well as their compositional relationships, more accurately compared to standard clustering algorithms (Affinity Propagation, k-means, Gaussian mixtures, etc.) in two challenging application areas: speaker recognition from speech with multiple overlapping speakers, and multi-label object recognition in open-world scenarios.[1]

[1]The data and code are available at https://github.com/jwhitehill/CompositionalClustering.

2. Related Work

2.1. Multi-Label Few-Shot and Zero-Shot Learning

The past 5 years have seen significant growth in the fields of multi-label few-shot and zero-shot learning (e.g., [18, 23, 7, 16]). Much of this work relies on the existence of a knowledge graph such as WordNet [22], a word embedding space such as GloVe [26], or external attribute vectors, to represent relationships among classes and thereby enable the model to generalize to data from unseen classes at test time. In contrast, the compositional embedding models of [1] and [19], and thus our work as well, make no such assumption – each class can be completely independent of each other. To our best knowledge, no prior work has investigated how to *cluster* examples automatically when the test classes are disjoint from training classes, when no support sets are provided, and when no semantic information about the test classes is provided. (Note that, when few-shot examples are provided for the test classes, then the "clustering" problem becomes trivial – the examples can be grouped based just on their estimated label vectors.)

2.2. Clustering

To our best knowledge, no previous clustering algorithm can both cluster a dataset and infer the compositionality among clusters. (A recent paper [24] examines how to cluster data that is "compositional" in the sense that they lie on a simplex and thus the features within every example "compose" to 1, but this is very different from our scenario.) Below we discuss the most similar work.

Mixture models, such as the Mixture of Gaussians fit using Expectation-Maximization, the Dirichlet mixture process [5], and the fuzzy k-means clustering algorithm [3], extend the standard k-means algorithm by "softly" assigning each data point to a probability distribution over the mixture components instead of giving a "hard" assignment like in k-means. Importantly, these approaches assume that each data point is generated by a *single* cluster, and the probability distribution expresses the uncertainty over which cluster it is. They can capture compositionality only in a limited sense by assuming that examples that lie between two (or more) cluster centroids belong to both (or all) of these clusters. These methods cannot distinguish between an example that is unconfidently assigned to a single cluster (thus resulting in high entropy over the mixture components for that example), from an example that is confidently assigned to multiple clusters.

Moreover, they will fail if the compositional cluster (e.g., the purple cluster in the left half of Figure 1) does not lie near the mean of its constituent singleton clusters (the red and blue clusters in the figure).

Hierarchical clustering algorithms create a tree (dendrogram) such that the n leaf nodes correspond 1-to-1 to the examples in the dataset, and each internal node i represents a cluster whose members consist of all the leaf nodes descending from i. Internal nodes closer to the root correspond to higher-level abstractions of the data. Hierarchical clustering algorithms can work either top-down by splitting clusters or bottom-up by merging clusters, until some clustering criterion is reached. One popular variant is *Agglomerative Clustering* using the Ward Jr [32] criterion, which seeks to minimize the variance within each cluster. In all cases, hierarchical clustering algorithms assign each example to a sequence of clusters of increasing generality, starting from the internal node just above the leaf all the way up to the root node, such that each parent cluster captures the *intersection* of the characteristics of the child clusters. In contrast, our proposed method can assign each example to contain the *union* of the properties in multiple clusters; this is tantamount to a dendrogram where each example is connected by an edge to multiple parent nodes, thus yielding a directed acyclic graph rather than a tree.

Multi-view clustering algorithms (e.g., Bickel and Scheffer [4]) partition the feature space into multiple subsets, each corresponding to a different "view" of the data (see [34, 11] for recent surveys). For instance, each example might be a video and thus have both auditory and visual features associated with it. Since multiple views often contain complementary information, harnessing all of them can often improve clustering accuracy. Moreover, the structure of the data from one view can provide implicit supervision when clustering using the other views. However, existing multi-view clustering methods do not have the ability to model compositionality. Franklin and Frank [9] recently proposed a method for "compositional clustering in task structure learning", but their method is more akin to multi-view clustering, and the compositionality pertains to how they tackled a control problem (separately addressing the reward and transition functions), not the clustering problem itself.

Exemplar-based clustering algorithms differ from **centroid-based** algorithms in how clusters are represented: In the former, each cluster is represented by a specific

example in the dataset; in contrast, the latter (e.g., k-means) may compute a function of the examples (e.g., the mean) to represent the cluster. One of the mostly widely used exemplar-based clustering algorithms is Affinity Propagation [10].

3. Approach I: Compositional Affinity Propagation (CAP)

Our first novel algorithm is Compositional Affinity Propagation, which is an exemplar-based clustering method and based on standard Affinity Propagation (AP) algorithm that is widely used for speaker diarization to group clusters of utterances into distinct speakers [35, 20]. CAP is based on an undirected probabilistic graphical model whose likelihood is approximately optimized using discrete optimization. Before presenting CAP, we first review standard AP [10].

3.1. Review of Affinity Propagation

Let $\mathcal{X} = \{x_1, \ldots, x_n\} \subset \mathbb{R}^p$ be a dataset, and let $\mathcal{C} = \{1, \ldots, n\} \doteq [n]$ be the set of indices of the (embedded) examples in $\mathcal{X}$. Next, let $c_1, \ldots, c_n \in \mathcal{C}$ be the cluster assignments: Each c_i denotes the *exemplar* representing the cluster to which example i belongs; if example i itself is the exemplar for some other example $j \neq i$, then we require $c_i = i$. For instance, if $\mathcal{X}$ contains $n = 3$ examples, the first two of which belong to the same cluster and the third of which belongs to its own cluster, then we might have $c_1 = 2, c_2 = 2, c_3 = 3$ (or possibly $c_1 = 1, c_2 = 1, c_3 = 3$). Let $S : \mathcal{C} \times \mathcal{C} \to [-\infty, 0]$ map from a pair of example indices to the negative (squared) distance between the examples, i.e., $S(i, j) = -\|x_i - x_j\|^2$ for $i \neq j$; and let S map to a constant value for $i = j$, i.e., $S(i, i) = \gamma$, where γ is the "preference" (a hyperparameter) that x_i is an exemplar, where larger negative values discourage those examples from becoming exemplars. From these definitions, we can formulate the following constrained optimization problem:

$$\underset{c_1,\ldots,c_n \in \mathcal{C}}{\arg\max} \sum_{i=1}^{n} S(i, c_i) \qquad \text{s.t.} \quad (\exists i : c_i = k) \implies c_k = k$$

The objective is the sum of distances between each point and its assigned exemplar, and the constraints enforce consistency that examples used by others as exemplars also designate themselves as exemplars. The optimization has to weigh the cost γ of creating a new cluster against assigning examples to existing exemplars that are farther away.

Illustration: Given an appropriate choice for the γ, Affinity Propagation would yield the results shown in the top half of Figure 1. In particular, in scenario (1), the cluster shown in purple would be identified as an independent cluster with examplar c, and in scenario (2), the cluster shown in purple would contain the images with triangles as well as those composed of rectangles and circles.

Inference: Frey and Dueck [10] showed a procedure to find approximately optimal solutions by defining a *factor graph* to represent the variables and constraints, where S is interpreted as containing log-likelihoods, and then applying loopy belief propagation. This results in a new optimization problem where the goal is to find maximum a posteriori (MAP) solutions to $\arg\max_{c_1,\ldots,c_n \in \mathcal{C}} P(c_1, \ldots, c_n \mid S)$. where probability distribution P is understood to encode the constraints. Specifically, the factor graph contains variable nodes to represent $c_1, \ldots, c_n$ and factor nodes to represent both the log-likelihoods $S(1, \cdot), \ldots, S(n, \cdot)$ and a set of *constraints* $\delta_1, \ldots, \delta_n$. Each δ_k encodes whether c_k is compatible with the other $c_{k' \neq k}$:

$$\delta_k(c_1, \ldots, c_n) = \begin{cases} -\infty & \text{if } \exists i : (c_i = k) \wedge (c_k \neq k) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Given the factor graph, a sequence of "messages" (functions $\alpha, \rho : [n] \times [n] \times \mathcal{C} \to [-\infty, 0]$) is passed back and forth between the variable and factor nodes. Each variable i sends a message $\rho_{i \to k}(c_i)$ to constraint k, and each constraint k sends a message $\alpha_{i \leftarrow k}(c_i)$ to variable i, about the likelihood of each possible value of c_i. The values of α and ρ are determined by the max-product algorithm for loopy belief propagation [33] applied to the factor graph (see the Appendices). To find an approximate MAP estimate for all the c_i, we alternate between computing the α's and the ρ's. Finally, after any number of iterations, we compute $c_i^{\text{MAP}} = \arg\max_{c_i} [\sum_k \alpha_{i \leftarrow k}(c_i) + S(i, c_i)]$. Frey and Dueck [10] also presented an efficient ($O(n^2)$) method to calculate all the messages for each iteration.

3.2. Procedure: Compositional Affinity Propagation

Here we describe our proposed Compositional Affinity Propagation algorithm. At a high level, CAP innovates on classic AP by allowing each cluster to be represented by not just a single example ("singleton" cluster), but rather an entire *set* of examples ("compositional" cluster). Importantly, the examples in this set need not be semantically

similar or lie close to each other in the feature space; rather, the *union* of the characteristics of the examples in this set should be present in *each* of the examples belonging to the compositional cluster. In terms of the inference procedure, CAP is somewhat more complex than standard AP due to the need, as part of the max-product algorithm, to compute the maximum values of many subsets efficiently (FindAllMaxes).

Let $\mathcal{X} = \{x_1, \ldots, x_n\} \subset \mathbb{R}^p$ be a dataset. Let $\mathcal{C} \subset 2^{[n]} \setminus \emptyset$ be the set of compositions of examples in $\mathcal{X}$ under consideration, where we assume $\mathcal{C}$ contains all the singletons, i.e., $\{i\} \in \mathcal{C}, i = 1, \ldots, n$. Let $d = \max_{c \in \mathcal{C}} |c|$, i.e., the size of the largest composition under consideration. To identify which compositions contain (or do not contain) each example k, define functions $\phi, \overline{\phi} : [n] \to 2^{\mathcal{C}}$ such that $\phi(k) = \{c \in \mathcal{C} : c \ni k\}$ and $\overline{\phi}(k) = \mathcal{C} \setminus \phi(k)$.

Let f be defined as in standard Affinity Propagation. We further assume there is a function $g : 2^{\mathcal{X}} \setminus \emptyset \to \mathbb{R}^p$ that consumes a set of examples and produces another vector representing their composition; for singleton sets, we let g be the identity function, i.e., $g(\{x\}) = x$. For instance, g could be the element-wise maximum to perform pixel-wise superposition of the images; for word embeddings, it could be element-wise addition [2]; or it could be a trained neural network within a compositional embedding model. We define $S : [n] \times \mathcal{C} \to [-\infty, 0]$ to measure the distance between each example and each composition: $S(i, c) = -\|x_i - g(\{x_k : k \in c\})\|$ for $c \neq \{i\}$, and $S(i, \{i\}) = \gamma$ is a hyperparameter for each example.

Finally, define $c_1, \ldots, c_n \in \mathcal{C}$ as the assignment of which example belongs to which cluster. If $c_i = \{k\}$ (i.e., a singleton), then example i belongs to a *singleton* cluster with exemplar x_k. If $|c_i| \geq 2$, then example i belongs to the cluster with a *compositional exemplar* $g(\{x_k : k \in c_i\})$, i.e., the composition of all the examples in c_i. Note that, in general, compositional exemplars are not members of $\mathcal{X}$. In CAP, we require that, whenever some example i designates its exemplar either to *be* example k ($c_i = \{k\}$), or to *include* example k ($c_i \ni k$), then example k must designate itself as an exemplar ($c_k = \{k\}$). Example: if $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$ and we allow compositions of size at most 2, then $\mathcal{C} = \{\{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$; if x_1 and x_2 each constitutes its own cluster and the last two examples are both assigned to the composition of the first two clusters, then we would have $c_1 = \{1\}, c_2 = \{2\}, c_3 = c_4 = \{1, 2\}$.

Our new constrained optimization problem is thus:

$$\arg\max_{c_1,\ldots,c_n \in \mathcal{C}} \sum_{i=1}^{n} S(i, c_i) \quad \text{s.t.} \quad (\exists i : c_i \ni k) \implies c_k = \{k\}$$

Importantly, the optimization objective incurs *no additional cost* when an example is assigned to a compositional exemplar c_k as long as all of the examples $k' \in c_k$ have themselves already been designated as exemplars.

Illustration: Given an appropriate choice for γ, CAP would yield the results show in the bottom half of Figure 1. In scenario (1), the cluster shown in purple would be identified as a cluster with a *compositional exemplar*. In scenario (2), the compositional structure identified by the algorithm could help it to separate the images containing triangles from those containing both a rectangle and a circle.

3.2.1. Inference

As with standard Affinity Propagation, we find a MAP estimate for each c_i by defining a factor graph and computing and passing messages between the variables and the factors. We adjust the definition of δ_k to be:

$$\delta_k(c_1, \ldots, c_n) = \begin{cases} -\infty & \text{if } \exists i : (c_i \ni k) \wedge (c_k \neq \{k\}) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

In the Appendices, we derive a procedure to compute α and ρ efficiently; see Algorithm 1.

Theorem 3.1. *Let n be the number of examples in a dataset $\mathcal{X}$, and let d be the largest element in the set $\mathcal{C}$ containing all compositions under consideration. Then Algorithm 1 implements message passing (i.e., computation of sufficient statistics of α and ρ) for Compositional Affinity Propagation and operates in time $O(dn^{d+1})$ per iteration.*

Proof. See Appendices. □

Inferring the Number of Clusters: The hyperparameter γ in CAP is the penalty for creating a new cluster versus assigning data points to an existing one. It is similar to the concentration hyperparameter (often denoted α) in the Dirichlet mixture process [5].

3.3. CAP⊂: An Approximation to CAP

To improve the scalability of CAP, we can apply it to a randomly selected subset of examples $\widetilde{\mathcal{X}} \subset \mathcal{X}$ and infer the cluster assignments $\tilde{c}_1, \ldots, \tilde{c}_{|\widetilde{\mathcal{X}}|}$. Let $\mathcal{E} = \{\tilde{c}_i\}_{i=1}^{|\widetilde{\mathcal{X}}|} \subset \mathcal{C}$ be the set of *unique* exemplars (singleton or compositional) inferred for $\widetilde{\mathcal{X}}$. Then, for each

Algorithm 1 Compositional Affinity Propagation (CAP)

CAP$(S, \mathcal{C})$:
  $\phi(k) \leftarrow \{c \in \mathcal{C} : c \ni k\}, \quad \overline{\phi}(k) \leftarrow \mathcal{C} \setminus \phi(k) \quad \forall k$
  $q(i, c_i) \leftarrow 0 \quad \forall i, c_i$
  $a(i,k) \leftarrow 0, \quad \overline{a}(i,k) \leftarrow 0 \quad \forall i, k$
  while not converged **do**
    $b, \overline{b}, h \leftarrow$ ComputeRhoStats$(S, \mathcal{C}, \phi, \overline{\phi}, a, \overline{a}, q)$
    $a, \overline{a}, q \leftarrow$ ComputeAlphaStats$(\mathcal{C}, b, \overline{b}, h)$
  end while
  return $\arg\max_{c_i}(q(i, c_i) + S(i, c_i)) \quad \forall i$

ComputeRhoStats$(S, \mathcal{C}, \phi, \overline{\phi}, a, \overline{a}, q)$:
  for $i = 1, \ldots, n$ **do**
    $r, s \leftarrow$ FindAllMaxes$(S(i, \cdot) + q(i, \cdot), \mathcal{C}, \phi, \overline{\phi})$
    for $k = 1, \ldots, n$ **do**
      $b(i,k) \leftarrow \max(r(k) - a(i,k), s(k) - \overline{a}(i,k))$
      $\overline{b}(i,k) \leftarrow s(k) - \overline{a}(i,k)$
    end for
  end for
  for $k = 1, \ldots, n$ **do**
    $h(k) \leftarrow S(k, \{k\}) + q_k(\{k\}) - a(k,k)$
  end for
  return $b, \overline{b}, h$

ComputeAlphaStats$(\mathcal{C}, b, \overline{b}, h)$:
  for $k = 1, \ldots, n$ **do**
    $e(k) \leftarrow \sum_{i' \neq k} b(i', k)$
    $\overline{e}(k) \leftarrow \sum_{i' \neq k} \overline{b}(i', k)$
  end for
  for $i = 1, \ldots, n$ **do**
    for $k = 1, \ldots, n$ **do**
      if $i = k$ **then**
        $a(i,k) \leftarrow e(k)$
        $\overline{a}(i,k) \leftarrow \overline{e}(k)$
      else
        $a(i,k) \leftarrow h(k) + e(k) - b(i,k)$
        $\overline{a}(i,k) \leftarrow \max(\overline{b}(k,k) + \overline{e}(k) - \overline{b}(i,k),\ h(k) + e(k) - b(i,k))$
      end if
    end for
  end for
  for $i = 1, \ldots, n$ **do**
    $q^*(i) \leftarrow \sum_{k'} \overline{a}(i, k')$
    for $c_i \in \mathcal{C}$ **do**
      $q(i, c_i) \leftarrow q^*(i) + \sum_{k' \in c_i}(a(i,k') - \overline{a}(i,k'))$
    end for
  end for
  return $a, \overline{a}, q$

Algorithm 2 Finding Maxima of Many Subsets

FindAllMaxes$(q, \mathcal{C}, \phi, \overline{\phi})$:
  $r(k) \leftarrow \max q(\phi(k)) \quad \forall k$
  $s(k) \leftarrow -\infty \quad \forall k$
  for $j = 1, \ldots, d$ **do**
    for $\tau = \{t_1, \ldots, t_{j-1}\}$ s.t. $\exists t_j > t_{j-1} : \{t_1, \ldots, t_j\} \in \mathcal{C}$ **do**
      $\psi_\tau \leftarrow \{\{t_1, ..., t_{j-1}, t_j\}\}_{t_j > t_{j-1}} \cap \mathcal{C}$
      $c^1, c^2 \leftarrow \arg\max^{1,2}_{c \in \psi_\tau} q(c)$
      for $k = 1, \ldots, n$ **do**
        if $c^1 \in \overline{\phi}(k)$ **then**
          $s(k) \leftarrow \max(s(k), q(c^1))$
        else if $c^2 \in \overline{\phi}(k)$ **then**
          $s(k) \leftarrow \max(s(k), q(c^2))$
        end if
      end for
    end for
  end for
  return r, s

example x_i in the original dataset $\mathcal{X}$, we designate its exemplar to be the $\tilde{c}_i \in \mathcal{E}$ that is closest to it x_i according to f. Specifically, we assign $c_i = \arg\min_{\tilde{c}_i \in \mathcal{E}} \|x_i, g(\{x_j : j \in \tilde{c}_i\})\|$. We call this method CAP$\subset$.

4. Approach II: Compositional *k*-means

The second compositional clustering algorithm we propose is called Compositional k-means (CKM). In contrast to CAP, which uses discrete optimization to assign examples to exemplars, CKM uses gradient descent to minimize a sum of squared distances by adjusting the real-valued cluster centroids. Like CAP, the CKM method can potentially cluster the data in Figure 1 more accurately by harnessing the composition function g to infer which examples belong to singleton clusters versus compositional clusters. CKM is a centroid-based method rather than an exemplar-based clustering method. Hence, each cluster assignment variable c_i is a subset of $[k]$ (rather than of $[n]$, like with CAP).

4.1. Review of classic k-means

Given the number of clusters k as input, classic k-means seeks to assign each of the n examples to one of the k clusters (denoted $c_i \in [k]$ for each i), so as to minimize the sum of squared distances (SSD) $\text{SSD}(\{m_j\}_{j=1}^k, \{c_i\}_{i=1}^n) = \sum_{i=1}^n \|x_i - m_{c_i}\|^2$. Here, each

$m_j \in \mathbb{R}^p$ is a cluster centroid, and each $c_i \in [k]$ is a cluster index. To (locally) minimize the SSD, two steps are executed in alternation until convergence:

1. Assign each x_i to the cluster j whose centroid $m_j \in \mathbb{R}^p$ is closest to x_i; and
2. Compute each centroid m_j as the mean of the points assigned to cluster j.

In particular, the second step is the closed-form minimizer of the SSD w.r.t. the centroids m_j. Since each of these steps is guaranteed not to increase the SSD, and since a lower bound on SSD is always 0, the algorithm is guaranteed to converge to a local minimum.

4.2. Procedure: Compositional k-means

Let the number of singleton clusters k (e.g., the number of individual speakers in the audio, or the number of basic object classes in the image set) be known, and let $\mathcal{K} \subset 2^{[k]}$ be a set of possible compositions of the singleton clusters, where we require that $\mathcal{K}$ contains all the singletons, i.e., $\{i\} \in \mathcal{K}, i = 1, \ldots, k$. Assume composition function g is differentiable. CKM seeks to assign each x_i to *either* one of k singleton clusters (a single person speaking in isolation, or a single object by itself) *or* to a compositional cluster (the composition of multiple speakers in an audio, or multiple objects in an image) so as to minimize the following sum of squared distances:

$$\mathrm{SSD}(\{m_{\{j\}}\}_{j=1}^{k}, \{c_i\}_{i=1}^{n}) = \sum_{i=1}^{n} \|x_i - m_{c_i}\|^2 \tag{3}$$

where each compositional centroid $m_\eta = g(\{m_{\{j\}}\}_{j\in\eta})$ (for $\eta \in \mathcal{K}$ and $|\eta| > 1$) is computed using the composition function g. (Note the small difference in notation compared to the SSD in standard k-means in the subscript of m so as to emphasize that a cluster centroid may represent the composition of other clusters.)

Like the classic k-means, the SSD is a function of the *singleton* cluster centroids (i.e., $m_{\{1\}}, \ldots, m_{\{k\}}$). Unlike classic k-means, the CKM method can assign each example to either a singleton or a compositional cluster. By adjusting the singleton centroids, the locations of the *compositional* centroids – and thus the SSD value itself – are also affected due to their dependence via g.

At a high level, CKM works as follows: After initializing the singleton cluster centroids randomly and computing the compositional centroids using g, a two-step alternating procedure is executed whereby (a) each example x_i is assigned to the closest centroid

(either singleton or compositional), and (b) the *singleton* centroids $m_{\{1\}}, \ldots, m_{\{k\}}$ are adjusted using gradient descent (with learning rate ϵ) to reduce the SSD in Equation 3. Since we assume g is a differentiable function (typically implemented as a neural network), the gradient of the SSD w.r.t. each singleton centroid (keeping the weights of g fixed) can be computed easily. During the optimization (see Algorithm 3), CKM dynamically infers which clusters are singletons and which are compositional, and also estimates the centroids of the singleton clusters so as to trade off between fitting the singletons and the compositional clusters well. Note that (like with classic k-means) the initialization in step 1 can affect which local minimum is reached, and thus it is often useful to try multiple random seeds and to choose the best seed based on the lowest SSD.

Algorithm 3 Compositional k-means (CKM)

CKM$(\mathcal{X}, \mathcal{K}, \epsilon)$:
  Set each $m_{\{j\}}, j \in [k]$ to a randomly drawn (without replacement) example in $\mathcal{X}$.
  Compute compositional centroids: $m_\eta \leftarrow g(\{m_{\{j\}}\}_{j \in \eta}) \quad \forall \eta \in \mathcal{K} : |\eta| > 1$.
  while not converged **do**
    $c_i \leftarrow \arg\min_{\eta \in \mathcal{K}} \|x_i - m_\eta\|^2 \quad \forall i$.
    $m_{\{1\}}, \ldots, m_{\{k\}} \leftarrow \text{SGD}\left(\sum_{i=1}^n \|x_i - m_{c_i}\|^2; \{m_{\{j\}}\}_{j=1}^k; \epsilon\right)$
  end while
  return $\{c_i\}_{i=1}^n$

Convergence: Unlike in classical k-means, the second step of the alternation (adjustment of cluster centroids) in CKM is conducted numerically rather than analytically. However, assuming the learning rate of gradient descent is sufficiently small, it will not increase the SSD. Since the first step of the alternation can also never increase the SSD, and since the SSD is bounded below, the algorithm will converge to a local minimum.

Comparison to CAP: CKM requires that g be differentiable, and it uses gradient-based optimization using neural network packages such as TensorFlow, PyTorch, etc. In contrast, CAP, as it only involves computing maxes and sums, can be implemented in simple Python or C code, and it does not require a differentiable g. Each step of the while-loop takes runtime $O(nk^d)$, where d is the size of the largest composition in $\mathcal{K}$. Since the number of singleton clusters k is typically much smaller than the number of examples n, CKM can run much faster than CAP.

Inferring the Number of Clusters: While CKM takes the number of singleton

clusters k as input, it infers the number of compositional clusters automatically based on the data – if no examples are assigned to a particular composition, then that compositional cluster does not exist. Moreover, the value k itself can be estimated by techniques such as the Gap statistic [31] that is commonly used for standard k-means clustering.

5. Approach III: Greedy Compositional Reassignment (GCR)

The third approach that we explored for compositional clustering is based on the idea of using any standard clustering algorithm to partition the data $\mathcal{X}$ into clusters, and then using the composition function g to find the optimal "reassignment" of the inferred clusters so that some of them are considered to be compositions of others. Suppose we first obtain (e.g., from Agglomerative Clustering) a set $\mathcal{E} = \{m_{\{1\}}, \ldots, m_{\{k\}}\}$ of k cluster centroids. Then we could iterate over every possible subset $\tilde{\mathcal{E}} \subseteq \mathcal{E}$; these represent the compositional clusters. For each $\tilde{\mathcal{E}}$, we conduct an inner-loop to iterate over every possible 1-to-1 map from $\tilde{\mathcal{E}}$ to the set of compositions (via g) of $\mathcal{E} \setminus \tilde{\mathcal{E}}$; these are the singleton/singleton clusters. We would finally select $\tilde{\mathcal{E}}$ and its map to $\mathcal{E} \setminus \tilde{\mathcal{E}}$ so as to minimize the sum of distances between the examples and their assigned cluster centroids (either singletons or compositional).

Unfortunately, due to the factorial time cost, this brute-force approach quickly becomes completely impractical (e.g., for $|\mathcal{E}| = 15$ and $d = 2$, there are 107770296705436 possibilities). However, the idea gave us inspiration for a tractable greedy heuristic that we call Greedy Compositional Reassignment (GCR). Like CKM, GCR is a centroid-based clustering method. It uses g and the distances between cluster centroids to determine the compositional relationships in a greedy manner and thereby avoid the factorial time cost.

5.1. Procedure: Greedy Compositional Reassignment

Assume that a standard clustering method (we use Agglomerative Clustering) has produced a clustering with k centroids $m_{\{1\}}, \ldots, m_{\{k\}}$ and cluster assignments $c_1, \ldots, c_n$, where each $c_i \in [k]$. Let $\mathcal{K}$ be the set of compositions under consideration.GCR first uses g to compute the location of the compositional centroid for every $\eta \in \mathcal{K}$. It then finds, for each putative singleton cluster $j \in [k]$, the distance d_j to the closest compositional centroid $b_j \in \mathcal{K}$; if d_j is below a threshold τ, then cluster j is concluded to actually be a composition of the two other clusters in b_j, and all the examples that were previously assigned to

cluster j are reassigned to the compositional cluster b_j. The process is repeated for each singleton cluster j according to the distances d_j sorted from smallest to largest until one of several possible termination conditions are reached (so as to maintain consistency, e.g., avoid cycles of compositionality); once this point is reached, all the remaining clusters that were not reassigned to be compositional are deemed to be singletons. The algorithm uses sets $\mathcal{S}$ and $\mathcal{T}$ to keep track of which clusters have been assigned as singletons and which are assigned as compositions, respectively. The final assignment of an example to a cluster index is denoted $c'_i \in \mathcal{K}$ for each example $i \in [n]$. See Algorithm 4 for details.

Algorithm 4 Greedy Compositional Reassignment (GCR)

GCR$(\mathcal{X}, \mathcal{K}, \tau)$:
  $\mathcal{S} \leftarrow \emptyset, \quad \mathcal{T} \leftarrow \emptyset$.
  Obtain preliminary clustering: $\{m_{\{j\}}\}_{j=1}^{k}, \{c_i\}_{i=1}^{n} \leftarrow$ `AgglomerativeClustering`$(\mathcal{X})$.
  Compute compositional centroids: $m_\eta \leftarrow g(\{m_{\{j\}}\}_{j \in \eta}) \quad \forall \eta \in \mathcal{K}$.
  $b_j \leftarrow \arg\min_\eta \|m_{\{j\}} - m_\eta\| \quad \forall j \in [k]$.
  $d_j \leftarrow \|m_{\{j\}} - m_{c_j}\| \quad \forall j \in [k]$.
  for $j \in [k]$ according to `argsort`$(\{d_j\})$ **do**
    if $d_j \geq \tau$ or $j \in \mathcal{S}$ or $b_j \cap \mathcal{T} \neq \emptyset$ **then**
      Assign remaining clusters $j' > j$ to singletons: $c'_i \leftarrow \{j'\} \quad \forall i : c_i = j', j' \geq j$.
      break
    end if
    Assign cluster j to composition: $c'_i \leftarrow b_j \quad \forall i : c_i = j$.
    Add the clusters in b_j to the set of singletons: $\mathcal{S} \leftarrow \mathcal{S} \cup b_j$.
    Add the current cluster (j) to the set of compositions: $\mathcal{T} \leftarrow \mathcal{T} \cup \{j\}$.
  end for
  return $\{c'_i\}_{i=1}^{n}$.

Comparison to CAP and CKM: Excluding the runtime cost of the initial clustering using Agglomerative Clustering, GCR method is much faster than CAP and CKM since it iterates over the k singleton clusters at most once, and each iteration is simple. Note that, whereas CKM and CAP can use the compositionality to partition the data into clusters more cleanly (see the right half of Figure 1), GCR cannot – it only has the ability to infer the compositional relationships among already-formed clusters (left half of Fig. 1).

Inferring the Number of Clusters: Since GCR first runs a standard clustering algorithm as a subroutine, then any technique that can estimate the number of clusters for that clustering algorithm (e.g., Gap statistic) can also be used for GCR.

6. Experiments

To evaluate the proposed algorithms, we conducted experiments, using standard datasets that are widely used for few-shot learning research, on both multi-object image recognition and multi-person speaker diarization with overlapping speech. We follow the workflow described in the Introduction, i.e., for each problem domain, we use few-shot learning to train an embedding function f^{emb} to separate examples by their classes, as well as a composition function g [1, 19] that can estimate the location in the embedding space of the union of multiple sets of classes. We train these models jointly and episodically, where the episodes contains examples from different set of classes. **Evaluation**: Since standard clustering metrics such as the Adjusted Rand Index (ARI) do not capture compositionality, we devise a new evaluation metric called the Compositional Rand Index.

6.1. Evaluation Metric: the Compositional Rand Index (CRI)

Suppose dataset $\mathcal{X} = \{x_1, \ldots, x_n\}$ contains l singleton clusters and some number (possibly 0) of compositional clusters. Then $\mathcal{Y} = 2^{[l]} \setminus \emptyset$ is the set of all possible ground-truth cluster labels, and $y_1, \ldots, y_n \in \mathcal{Y}$ are the cluster assignments. A sensible evaluation criterion of some inferred labels $c_1, \ldots, c_n \in \mathcal{C}$ w.r.t. ground-truth should capture the *number* of clusters, their *purity*, and their *compositional relationships*. It should *not* depend on the particular naming of cluster labels (the identifiability issue). With these goals in mind, we propose the Compositional Rand Index (CRI) to compute the probability, over all pairs $i \neq j$, that the inferred labels agree with the ground-truth about whether the cluster assignment of example i subsumes the cluster assignment of example j:

$$\text{CRI}(c_1, \ldots, c_n, y_1, \ldots, y_n) = \frac{1}{n(n-1)} \sum_{i \neq j} \mathbb{I}[\mathbb{I}[c_i \supseteq c_j] = \mathbb{I}[y_i \supseteq y_j]] \tag{4}$$

where $\mathbb{I}[\cdot]$ is a 0-1 indicator function. For datasets without compositionality (i.e., $\mathcal{Y} = [l]$), CRI is equivalent to the standard Rand Index [27].

6.2. Baseline Methods

We chose several baselines that seemed the most reasonable alternative approaches, even if they had no explicit ability to model compositionality.

Ignore compositionality: One approach is simply to ignore the compositional relationships among clusters and consider each cluster as completely independent; this

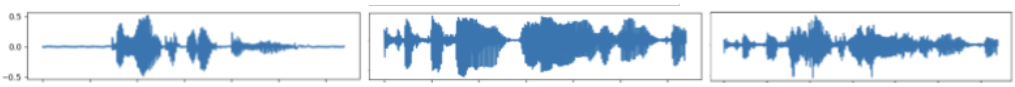

Figure 2: Some audio examples from LibriSpeech. From left to right are the waveforms from speaker 1, speaker 2, and overlapping speech from both speakers.

is illustrated in Figure 1 (top row). Any standard clustering method can thus be used. While it will pay a penalty under the CRI metric since it misses the compositional relationships, it can sometimes (Figure 1 upper-left, but not upper-right) still do a good job overall by correctly forming coherent clusters. With this motivation, we use standard **Affinity Propagation (AP)** as well as **Agglomerative Clustering (AC)** with the Ward criterion as two baselines.

Infer compositionality from "soft" label assignments: Mixture models such as the classic Mixture of Gaussians and fuzzy k-means [3] (it is more commonly called "fuzzy c-means" in the literature) assign to each example a vector of probabilities that express the likelihood that it belongs to each of the k clusters. By thresholding these probabilities with some threshold τ, one can obtain a *set* of cluster labels for each example. This method can work if the embedding space is structured so that examples whose cluster label set is $\{a, b\}$ lie near the midpoint between those examples whose label set is $\{a\}$ and those whose label set is $\{b\}$. Based on this approach, we tested both **Fuzzy c-Means (FCM)** and **Gaussian Mixture Models (GMM)** as baselines.

Oracle singleton clustering: To assess how well a *perfect* clustering method would work that can determine the cluster memberships exactly but not infer compositionality, we include an Oracle Singleton Clustering (OSC) baseline. Note it is not meaningful when measuring ARI (since it would be 100%); hence, we use it only for the CRI.

6.3. Experiment I: LibriSpeech

Real-world conversations and meetings often contain moments when multiple people are speaking simultaneously (due to interruptions, sub-group conversations, etc.). Hence, an important few-shot learning problem is to identify the *set* of people speaking at any given moment in time, where the classes (people) at test time usually differ from the classes at training time. We thus used the LibriSpeech [25] dataset to explore how well each clustering method can cluster speech samples into speaker sets and infer the

compositional relationships between sets. LibriSpeech is a corpus of approximately 1000 hours of English audiobook speech from 2484 speakers. While it contains only individual speakers, we can synthesize simultaneous speech by combining individual tracks, similarly to work by [12, 15, 21]. See Figure 2 for some examples of the audio waveforms.

Embedding model: We used LibriSpeech to train a compositional embedding model (f^{emb} and g) for speaker verification using an LSTM neural network on top of MFCC audio features (see Appendices for details). Importantly, none of the classes (speakers) that were used for optimizing these networks were used in the clustering experiments. [2]

Procedure: Our first experiment considers compositionality of degree at most $d = 2$. We created datasets $\mathcal{X}$ of size $n \in \{150, 750, 1500, 7500, 15000\}$; each dataset contained speech segments from 5 different speakers (picked from 100 speakers which were not seen during training). Some of the segments contained single speakers, and some contained combinations of two speakers. Hence, there were $\binom{5}{1} + \binom{5}{2} = 15$ different unique speaker sets in total. The test set contains 10 data trials for each n and the validation set has 10 trials when $n = 150$ (hyperparameters are picked based on $n = 150$ and used for all ns). For each n, we compared all three compositional clustering algorithms and all the baselines described above and then compared the resulting CRI (Section 6.1) and Adjusted Rand Index (ARI) scores. For CAP, we used the full inference procedure for $n = 150$, and we used CAP$\subset$ with a random subset of 150 examples for $n > 150$. All results of all clustering methods are averaged over 10 trials for each n.

To illustrate compositional clustering for $d = 3$ (i.e., up to 3 speakers speaking simultaneously), we performed a second experiment using the same composition function g as for $d = 2$ (i.e., it does not need to be retrained for different d). There are 25 classes in total (5 singletons, 10 2-sets, and 10 3-sets). We adopted the same hyperparameters for each method that were optimized in the previous experiment for $d = 2$. Due to the high computational cost, we varied n only up to 2500, and we did not try CAP.

Hyperparameter optimization: CAP, CAP$\subset$, and AP have one hyperparameter, which is the cost γ of creating a new singleton cluster. GCR has two hyperparameters: the first is the number of clusters in the first step of clustering and the second is the threshold

[2] For the baseline clustering methods, we also tried training a simpler embedding model f^{emb} *without* jointly training g to check whether that gave better performance. However, we found that this actually resulted in worse performance for the baselines, and hence we abandoned it.

LibriSpeech Results (2 Spkrs, CRI%)

n	150	750	1500	7500	15000
GCR	94.6 (1.1)	95.8 (0.7)	96.0 (0.8)	**96.5 (0.5)**	**96.6 (0.5)**
CAP	94.8 (1.4)	93.8 (1.4)	93.7 (1.5)	93.4 (1.3)	92.6 (2.1)
CKM	**95.7 (0.8)**	**96.3 (0.6)**	**96.1 (0.6)**	96.2 (0.4)	96.1 (0.5)
AP	87.9 (0.6)	87.0 (0.2)	86.1 (0.1)	85.0 (0.1)	84.8 (0.0)
AC	88.4 (0.3)	86.4 (0.2)	85.6 (0.1)	84.7 (0.0)	84.6 (0.0)
FCM	88.0 (0.5)	88.3 (0.5)	88.1 (0.4)	88.3 (0.4)	88.4 (0.4)
GMM	87.8 (0.4)	88.9 (0.4)	88.7 (0.4)	88.8 (0.3)	88.6 (0.5)
OSC	*91.1 (0.0)*	*91.1 (0.0)*	*91.1 (0.0)*	*91.1 (0.0)*	*91.1 (0.0)*

LibriSpeech Results (2 Spkrs, ARI%)

n	150	750	1500	7500	15000
GCR	77.2 (4.0)	**82.1 (3.4)**	**82.3 (3.7)**	**85.1 (2.9)**	**85.8 (2.6)**
CAP	76.5 (5.5)	74.2 (4.9)	74.1 (5.4)	73.2 (4.8)	72.5 (6.2)
CKM	**78.5 (3.7)**	81.8 (2.8)	81.4 (2.9)	83.4 (2.2)	83.1 (2.2)
AP	72.5 (5.1)	54.9 (2.9)	39.6 (2.2)	15.3 (0.9)	9.2 (0.6)
AC	74.2 (3.7)	43.1 (2.5)	28.6 (1.8)	8.4 (0.6)	4.7 (0.3)
FCM	70.0 (4.3)	70.5 (3.8)	69.0 (3.8)	70.7 (4.0)	71.8 (3.7)
GMM	70.5 (4.0)	80.3 (3.6)	78.6 (3.6)	79.0 (3.2)	78.8 (3.5)

Table 1: Results of LibriSpeech experiments (2 speakers) in CRI%/ARI% along with s.e.

τ to stop compositional label assignment. CKM has four hyperparameters: the number of singleton clusters k; the number of random initializations; the maximum number of alternations; and the learning rate. AC uses a distance threshold hyperparameter that determines whether to merge two clusters. GMM has one hyperparameter to decide the number of components. FCM has the number of clusters c and the corresponding threshold $1/c$ on the vector of probabilities that determines when the model infers that an example belongs to a compositional cluster; it also has a temperature m that can make the estimated class probabilities more or less entropic. The hyperparameter sets were decided separately for each method, based on pilot exploration, to give each method a good chance of succeeding. The hyperparameter values were then optimized so as to maximize the average (over 10 trials) CRI. Experiments were conducted using the Python code in the Github repository; the sklearn implementations of `AgglomerativeClustering` and `AffinityPropagation`; and the `SciKit-Fuzzy` of FCM.

Results: The mean CRI% and ARI% (along with standard error) for $d = 2$ are shown in Table 1. For all values of n, all the compositional clustering algorithms (the first three lines of the table) worked better than all of the standard clustering methods in

LibriSpeech Results (3 Spkrs, CRI%)			
n	250	1250	2500
GCR	87.4 (0.5)	88.6 (1.1)	89.6(1.3)
CKM	**95.1 (0.4)**	**93.1 (0.7)**	**94.7 (0.4)**
AP	80.8 (0.2)	84.2 (0.4)	84.9 (0.2)
AC	84.8 (0.1)	84.0 (0.1)	83.7 (0.0)
FCM	84.0 (0.1)	83.8 (0.3)	84.4 (0.2)
GMM	84.3 (0.2)	84.5 (0.4)	85.0 (0.2)
OSC	*87.1 (0.0)*	*87.2 (0.0)*	*87.2 (0.0)*
LibriSpeech Results (3 Spkrs, ARI%)			
n	250	1250	2500
GCR	55.7 (2.0)	59.0 (4.6)	69.3 (3.0)
CKM	**72.6 (2.0)**	**64.1 (3.1)**	**71.6 (1.8)**
AP	32.8 (1.5)	57.9 (4.6)	62.7 (2.6)
AC	54.2 (2.7)	32.9 (2.1)	22.6 (0.7)
FCM	51.2 (2.0)	50.4 (3.6)	57.0 (2.4)
GMM	53.3 (3.3)	60.4 (4.9)	67.8 (3.1)

Table 2: Results of LibriSpeech experiments (3 speakers) in CRI%/ARI% along with s.e.

terms of CRI; the differences (as assessed with matched-pair t-tests between methods over the 10 test trials, at the 0.05 significance level) were all stat. sig. GCR and CKM usually gave the highest accuracy. CAP comes in second place for $n = 150$ but as n increases, its accuracy decreases; this is likely because, for the larger n values, CAP$\subset$ sees a relatively smaller fraction of the total dataset as n increases. All the proposed methods outperformed the Oracle Singleton Clustering baseline, suggesting that they can both form coherent clusters and correctly infer the compositional relationships between them. Among the traditional methods, either FCM or GMM usually performed best: while it does have some ability to infer compositionality via the probability vector assigned to each example, it does not use the embedding model's composition function g; hence, it must rely on compositional clusters lying close to their constituent singleton clusters in the embedding space, which does not always happen in practice.

In terms of ARI – which can measure the purity of inferred clusters but not the accuracy of the inferred compositional relationships – both GCR and CKM always outperformed the best standard clustering method (though the differences were not always stat. sig.). This suggests that the g function enabled the compositional methods to obtain purer clusters. CAP outperformed all the standard clustering methods for $n = 150$ but not for

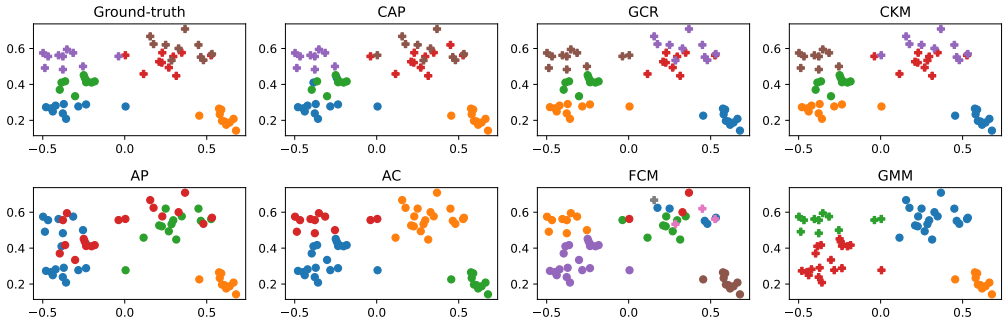

Figure 3: Some LibriSpeech clusters (for $d = 2$), according to different methods. Circles belong to singleton clusters; plusses belong to compositional clusters. Best viewed in color.

larger n (when CAP$\subset$ was used). Note that some ARI scores, particularly for the AC and AP methods, were very low for larger n; this is likely because the hyperparameters for all methods were optimized for CRI, not ARI, and because ARI adjusts for the accuracy obtained by just guessing. Manual inspection of the results suggests that some methods (e.g., AC for $n = 15000$) incorrectly deduced a very large number of clusters, which was heavily penalized by the ARI metric.

For $d = 3$ (Table 2), the trends were mostly similar to $d = 2$. The accuracy differences with the best-performing standard clustering method were stat. sig. for both GCR and CKM for CRI for all n; for ARI, the significance tests were mixed.

Inferred Clusters: Figure 3 shows some clustering results of the 4 different methods; in each plot (generated using PCA applied to $\mathcal{X}$), circles and plusses represent examples from singleton and compositional clusters, respectively. (To avoid clutter, we show just 3 singleton clusters and their compositions, and the inferred relationships of which clusters are composed to yield other clusters are not shown.) All three compositional clustering methods are largely successful in inferring both the clusters and their compositionality. AP and AC can approximately infer the clusters but sometimes lump groups of examples together that actually come from distinct clusters. FCM and GMM do manage to infer some compositionality correctly, but not as well as the compositional methods.

6.4. Experiment II: OmniGlot

Here we considered a multi-object image recognition problem using the OmniGlot [17] dataset. OmniGlot contains images of handwritten symbols from many languages. It

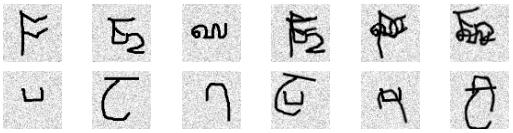

Figure 4: Some representative examples from OmniGlot images: the first three columns show examples from singleton clusters, whereas the latter show images from compositional clusters.

has 1623 different handwritten characters from 50 different alphabets. We can synthesize images with multiple symbols by element-wise superposition; Figure 4 shows 4 groups of examples used in the experiment, where each group contains images with cluster labels $\{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}$. Because of the intertangling of the different symbols in each compositional image, the recognition problem is quite challenging. We created 10 trials for each n in the test set and 10 trials when $n = 150$ in the validation set.

Embedding model: In our experiment, all characters are augmented with random scaling and shifting, and random Gaussian noise is added to the background. We pre-trained a compositional embedding model f^{emb} using a ResNet18 [14] network; the composition function g is the same as for LibriSpeech (see Appendices).

Procedures are analogous to the experiments on LibriSpeech (see Section 6.3).

Results are in Table 3. CKM gave the highest accuracy for almost all values of n. With CKM (both on LibriSpeech and OmniGlot), we found that trying different random initializations of the singleton cluster centroids, and then choosing the final clustering based on the sum of squared distances after training, was important to get good performance. Nonetheless, CKM's accuracy was not just due to randomly "guessing" which of the clusters were singleton clusters – the number of random seeds in our experiments (we used 100) was far smaller than the total number of possible choices of 5 singleton clusters out of 15 total clusters ($\binom{15}{5} = 3003$), suggesting that CKM uses g and the numerical SSD-minimization procedure to deduce compositional structure. After CKM, GCR was usually second best, followed by CAP (which was sometimes slightly worse than the best standard clustering methods, in terms of ARI). Among the traditional clustering methods, FCM usually performed best. The CRI accuracy improvement compared to the best standard clustering method was stat. sig. for all the compositional clustering algorithms. For ARI, the results of the t-tests were mixed.

OmniGlot Results (CRI%)

n	150	750	1500	7500	15000
GCR	**94.9 (0.4)**	95.9 (0.3)	96.0 (0.2)	96.3 (0.3)	96.3 (0.3)
CAP	93.3 (0.4)	92.8 (0.5)	92.6 (0.5)	92.7 (0.5)	92.8 (0.2)
CKM	94.3 (1.2)	**97.1 (0.4)**	**96.7 (0.5)**	**96.9 (0.4)**	**96.9 (0.5)**
AP	87.9 (0.1)	86.3 (0.1)	85.6 (0.1)	84.8 (0.0)	84.7 (0.0)
AC	87.9 (0.1)	85.7 (0.1)	85.2 (0.0)	84.7 (0.0)	84.6 (0.0)
FCM	88.1 (0.1)	88.0 (0.1)	87.9 (0.2)	88.0 (0.1)	87.9 (0.3)
GMM	85.7 (0.4)	86.4 (0.5)	86.0 (0.9)	87.9 (0.3)	87.1 (0.7)
OSC	*91.1 (0.0)*	*91.1 (0.0)*	*91.1 (0.0)*	*91.1 (0.0)*	*91.1 (0.0)*

OmniGlot Results (ARI%)

n	150	750	1500	7500	15000
GCR	76.0 (1.3)	81.0 (1.2)	81.4 (1.0)	84.6 (0.9)	84.9 (0.9)
CAP	63.1 (2.2)	64.9 (2.5)	64.3 (2.2)	64.7 (1.9)	66.1 (0.9)
CKM	**77.7 (4.1)**	**86.1 (1.3)**	**85.5 (1.5)**	**85.7 (1.3)**	**85.8 (1.6)**
AP	69.8 (1.3)	43.8 (1.7)	29.3 (1.3)	9.3 (0.4)	6.3 (0.1)
AC	65.4 (1.8)	29.1 (1.6)	18.1 (0.9)	5.8 (0.2)	4.4 (0.1)
FCM	69.7 (1.1)	67.1 (1.0)	66.6 (1.5)	67.1 (1.4)	67.7 (1.5)
GMM	55.1 (2.2)	63.1 (2.7)	65.5 (4.4)	75.0 (2.3)	70.0 (3.3)

Table 3: Results of OmniGlot experiments in CRI%/ARI% along with s.e.

7. Conclusions

We presented three new algorithms (CAP, CKM,and GCR) that can both cluster data and infer the compositional relationships between clusters. These algorithms can facilitate data visualization and exploratory data analyses on datasets where the classes have not been previously seen. Our experiments on the LibriSpeech and OmniGlot datasets suggest that modeling compositionality explicitly is useful and enables the proposed methods to identify coherent and distinctive clusters, and also to infer the compositional relationships between them. The proposed methods deliver substantially higher accuracy than can be achieved with standard methods (e.g., GMM, FCM), even when the latter have the ability to assign examples "softly" to multiple clusters. Among CKM, GCR, and CAP, we found that CKM and GCR gave higher accuracy and also scale better with dataset size n.

Limitations of proposed methods: In practice, training embedding function f^{emb} jointly with composition function g can be challenging, especially when the compositional degree $d \geq 3$. On the other hand, as few-shot learning is an active research field, more powerful embedding approaches could arise that make this challenge less severe.

Future work: We anticipate that, as multi-label few-shot learning research continues to grow, there will be increasing interest for methods to cluster data from unseen classes. Research on more accurate compositional embedding models, especially to the extent that the composition function g can be increased in accuracy, will likely lead to accuracy improvements in compositional clustering methods. One possible downstream application of our work is a simplified pipeline for speaker diarization: instead of separate algorithms to detect overlapping speech, separate speech segments into long vs. short turns, and then cluster the utterances [6], it may be possible to apply a compositional clustering algorithm that can diarize the set of all speech utterances in just one pass.

Acknowledgement: This research was supported by the NSF National AI Institute for Student-AI Teaming (iSAT) under grant DRL #2019805. The opinions expressed are those of the authors and do not represent views of the NSF. The research was also supported by NSF awards #2046505 and #1822768.

References

[1] Alfassy, A., Karlinsky, L., Aides, A., Shtok, J., Harary, S., Feris, R., Giryes, R., Bronstein, A.M., 2019. Laso: Label-set operations networks for multi-label few-shot learning, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 6548–6557.

[2] Allen, C., Hospedales, T., 2019. Analogies explained: Towards understanding word embeddings, in: International Conference on Machine Learning, PMLR. pp. 223–231.

[3] Bezdek, J.C., Ehrlich, R., Full, W., 1984. Fcm: The fuzzy c-means clustering algorithm. Computers & geosciences 10, 191–203.

[4] Bickel, S., Scheffer, T., 2004. Multi-view clustering., in: ICDM, Citeseer. pp. 19–26.

[5] Blei, D.M., Jordan, M.I., 2006. Variational inference for dirichlet process mixtures. Bayesian analysis 1, 121–143.

[6] Bullock, L., Bredin, H., Garcia-Perera, L.P., 2020. Overlap-aware diarization: Resegmentation using neural end-to-end overlapped speech detection, in: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE. pp. 7114–7118.

[7] Chen, T., Lin, L., Chen, R., Hui, X., Wu, H., 2020. Knowledge-guided multi-label few-shot learning for general image recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 44, 1371–1384.

[8] Deng, J., Guo, J., Xue, N., Zafeiriou, S., 2019. Arcface: Additive angular margin loss for deep face recognition, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 4690–4699.

[9] Franklin, N.T., Frank, M.J., 2018. Compositional clustering in task structure learning. PLoS computational biology 14, e1006116.

[10] Frey, B.J., Dueck, D., 2007. Clustering by passing messages between data points. science 315, 972–976.

[11] Fu, L., Lin, P., Vasilakos, A.V., Wang, S., 2020. An overview of recent multi-view clustering. Neurocomputing 402, 148–161.

[12] Fujita, Y., Kanda, N., Horiguchi, S., Nagamatsu, K., Watanabe, S., 2019. End-to-end neural speaker diarization with permutation-free objectives. arXiv preprint arXiv:1909.05952 .

[13] Hariharan, B., Girshick, R., 2017. Low-shot visual recognition by shrinking and hallucinating features, in: Proceedings of the IEEE international conference on computer vision, pp. 3018–3027.

[14] He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.

[15] Hershey, J.R., Chen, Z., Le Roux, J., Watanabe, S., 2016. Deep clustering: Discriminative embeddings for segmentation and separation, in: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE. pp. 31–35.
[16] Huynh, D., Elhamifar, E., 2021. Interaction compass: Multi-label zero-shot learning of human-object interactions via spatial relations, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 8472–8483.
[17] Lake, B.M., Salakhutdinov, R., Tenenbaum, J.B., 2015. Human-level concept learning through probabilistic program induction. Science 350, 1332–1338.
[18] Lee, C.W., Fang, W., Yeh, C.K., Wang, Y.C.F., 2018. Multi-label zero-shot learning with structured knowledge graphs, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1576–1585.
[19] Li, Z., Mozer, M., Whitehill, J., 2021. Compositional embeddings for multi-label one-shot learning, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pp. 296–304.
[20] Li, Z., Whitehill, J., 2021. Compositional embedding models for speaker identification and diarization with simultaneous speech from 2+ speakers, in: ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE. pp. 7163–7167.
[21] Menne, T., Sklyar, I., Schlüter, R., Ney, H., 2019. Analysis of deep clustering as preprocessing for automatic speech recognition of sparsely overlapping speech. arXiv preprint arXiv:1905.03500 .
[22] Miller, G.A., 1995. Wordnet: a lexical database for english. Communications of the ACM 38, 39–41.
[23] Narayan, S., Gupta, A., Khan, S., Khan, F.S., Shao, L., Shah, M., 2021. Discriminative region-based multi-label zero-shot learning, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 8731–8740.
[24] Pal, S., Heumann, C., 2022. Clustering compositional data using dirichlet mixture model. Plos one 17, e0268438.
[25] Panayotov, V., Chen, G., Povey, D., Khudanpur, S., 2015. Librispeech: an asr corpus based on public domain audio books, in: 2015 IEEE international conference on acoustics, speech and signal processing (ICASSP), IEEE. pp. 5206–5210.
[26] Pennington, J., Socher, R., Manning, C.D., 2014. Glove: Global vectors for word representation, in: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp. 1532–1543.
[27] Rand, W.M., 1971. Objective criteria for the evaluation of clustering methods. Journal of the American Statistical association 66, 846–850.
[28] Schroff, F., Kalenichenko, D., Philbin, J., 2015. Facenet: A unified embedding for face recognition and clustering, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 815–823.
[29] Snyder, D., Garcia-Romero, D., Sell, G., Povey, D., Khudanpur, S., 2018. X-vectors: Robust dnn embeddings for speaker recognition, in: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE. pp. 5329–5333.
[30] Song, G., Tan, X., Zhao, J., Yang, M., 2021. Deep robust multilevel semantic hashing for multi-label cross-modal retrieval. Pattern Recognition 120, 108084.
[31] Tibshirani, R., Walther, G., Hastie, T., 2001. Estimating the number of clusters in a data set via the gap statistic. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 63, 411–423.
[32] Ward Jr, J.H., 1963. Hierarchical grouping to optimize an objective function. Journal of the American statistical association 58, 236–244.
[33] Weiss, Y., Freeman, W.T., 2001. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. IEEE Transactions on Information Theory 47, 736–744.
[34] Yang, Y., Wang, H., 2018. Multi-view clustering: A survey. Big Data Mining and Analytics 1, 83–107.
[35] Yin, R., Bredin, H., Barras, C., 2018. Neural speech turn segmentation and affinity propagation for speaker diarization, in: Annual Conference of the International Speech Communication Association.
[36] Zelenak, M., Segura, C., Luque, J., Hernando, J., 2012. Simultaneous speech detection with spatial features for speaker diarization. IEEE Transactions on Audio, Speech, and Language Processing 20, 436–446.
[37] Zhou, F., Huang, S., Liu, B., Yang, D., 2021. Multi-label image classification via category prototype compositional learning. IEEE Transactions on Circuits and Systems for Video Technology 32, 4513–4525.

Appendix A. Loopy Belief Propagation for Standard Affinity Propagation

When applying the max-product algorithm to the factor graph in standard Affinity Propagation, a sequence of "messages" (functions $\alpha, \rho : [n] \times [n] \times \mathcal{C} \to \mathbb{R}_{\leq 0} \cup \{-\infty\}$) is passed back and forth between the variable and factor nodes. Each variable i sends a message $\rho_{i \to k}(c_i)$ to constraint k, and each constraint k sends a message $\alpha_{i \leftarrow k}(c_i)$ to variable i, about the likelihood of each possible value of c_i. The max-sum algorithm (and the related max-product algorithm for factor graphs) dictates that $\rho_{i \to k}(c_i)$ equals the sum of messages over c_i's neighbors *except* δ_k (i.e., $\{\delta_{k'}\}_{k' \neq k}$):

$$\rho_{i \to k}(c_i) = S(i, c_i) + \sum_{k' \neq k} \alpha_{i \leftarrow k'}(c_i) \tag{A.1}$$

Also, for MAP estimation, $\alpha_{i \leftarrow k}(c_i)$ equals the maximum possible sum of the messages from all of δ_k's neighbors *except* i (i.e., $\{c_{i'}\}_{i' \neq i}$), plus the value of δ_k itself:

$$\alpha_{i \leftarrow k}(c_i) = \max_{\{c_{i'}\}_{i' \neq i}} \left[\delta_k(c_1, \ldots, c_n) + \sum_{i' \neq i} \rho_{i' \to k}(c_{i'}) \right] \tag{A.2}$$

Appendix B. Derivation of Algorithm 1 (CAP) and Proof of Theorem 1

Here we derive Algorithm 1 from the definitions of α and ρ to optimize the Compositional Affinity Propagation model. We also prove the time cost in Theorem 1. Recall the definitions of α, ρ, and δ:

$$\rho_{i \to k}(c_i) = S(i, c_i) + \sum_{k' \neq k} \alpha_{i \leftarrow k'}(c_i) \tag{B.1}$$

$$\alpha_{i \leftarrow k}(c_i) = \max_{\{c_{i'}\}_{i' \neq i}} \left[\delta_k(c_1, \ldots, c_n) + \sum_{i' \neq i} \rho_{i' \to k}(c_{i'}) \right] \tag{B.2}$$

$$\delta_k(c_1, \ldots, c_n) = \begin{cases} -\infty & \text{if } \exists i : (c_i \ni k) \wedge (c_k \neq \{k\}) \\ 0 & \text{otherwise} \end{cases} \tag{B.3}$$

Each message $\alpha_{i \leftarrow k}(c_i)$ computes, for a given value of c_i for variable i, an (unnormalized) log-likelihood of the *best possible configuration* of the assignments of all the *other* variables $\{c_{i' \neq i}\}$, given that constraint k is satisfied (i.e., δ_k is finite). There are four cases in which this occurs; they mirror those in standard Affinity Propagation but differ slightly.

For each case, the δ term in the RHS of Eqn. B.2 vanishes; the only remaining terms are the sum of the ρ's. Also, since each summand in Eqn. B.2 depends on just a single unique $c_{i'}$, the max of the sum becomes the sum of the max. Cases:

1. $i = k$, $c_i = \{k\}$: Since in this case example $i = k$ designates itself as an exemplar, then the constraint δ_k is immediately satisfied. Moreover, any of the other examples $i' \neq i$ is free to choose (or not choose) example k as an exemplar, and therefore we can take the maximum over *any* possible value for each $c_{i'}$. Hence,

$$\alpha_{i \leftarrow k}(c_i) \quad = \quad \max_{\{c_{i'}\}_{i' \neq k}} \left[0 + \sum_{i' \neq k} \rho_{i' \to k}(c_{i'}) \right] = \sum_{i' \neq k} \max_{c_{i'}} \rho_{i' \to k}(c_{i'})$$

2. $i = k, c_i \not\ni k$: Since example $i = k$ does not designate itself as an exemplar, then none of the other examples $i' \neq i$ may choose k as its exemplar. Hence, $\alpha_{i \leftarrow k}(c_i) = \sum_{i' \neq k} \max_{c_{i'} \not\ni k} \rho_{i' \to k}(c_{i'})$.
3. $i \neq k, c_i \ni k$: Since example i designates its exemplar either to be or to include example k, then α is finite only if $c_k = \{k\}$, and each remaining example $i' \notin \{i, k\}$ is free to designate any example as its exemplar. Hence, $\alpha_{i \leftarrow k}(c_i) = \rho_{k \to k}(\{k\}) + \sum_{i' \notin \{i,k\}} \max_{c_{i'}} \rho_{i' \to k}(c_{i'})$.
4. $i \neq k, c_k \not\ni k$: Since example i does not designate k as an exemplar, then example k can either be an exemplar or not, and we take the max over both possibilities:

$$\alpha_{i \leftarrow k}(c_i) \quad = \quad \max \left[\max_{c_k \not\ni k} \rho_{k \to k}(c_k) + \sum_{i' \notin \{i,k\}} \max_{c_{i'} \not\ni k} \rho_{i' \to k}(c_{i'}), \quad \rho_{k \to k}(\{k\}) + \sum_{i' \notin \{i,k\}} \max_{c_{i'}} \rho_{i' \to k}(c_i) \right]$$

Note that $\alpha_{i \leftarrow k}(c_i) = -\infty$ if $i = k$, $c_i \ni i$ and $c_i \neq \{i\}$. However, in practice we can avoid this case by instead setting $S(i, c_i) = -\infty$ whenever $c_i \ni i$ and $c_i \neq \{i\}$. Given the four cases above, we have the following definition of α:

$$\alpha_{i \leftarrow k}(c_i) \quad = \quad \max_{\{c_{i'}\}_{i' \neq i}} \left[\delta_k(c_1, \ldots, c_n) + \sum_{i' \neq i} \rho_{i' \to k}(c_{i'}) \right] \tag{B.4}$$

$$= \begin{cases} \sum_{i' \neq k} \max_{c_{i'}} \rho_{i' \to k}(c_{i'}) & i = k, c_i = \{k\} \\ \sum_{i' \neq k} \max_{c_{i'} \not\ni k} \rho_{i' \to k}(c_{i'}) & i = k, c_i \not\ni k \\ \rho_{k \to k}(k) + \sum_{i' \notin \{i,k\}} \max_{c_{i'}} \rho_{i' \to k}(c_{i'}) & i \neq k, c_i \ni k \\ \max \Big[\max_{c_k \not\ni k} \rho_{k \to k}(c_k) + \sum_{i' \notin \{i,k\}} \max_{c_{i'} \not\ni k} \rho_{i' \to k}(c_{i'}), & \\ \qquad \rho_{k \to k}(k) + \sum_{i' \notin \{i,k\}} \max_{c_{i'}} \rho_{i' \to k}(c_{i'}) \Big] & i \neq k, c_k \not\ni k \end{cases} \tag{B.5}$$

In the most naive implementation, evaluating α for *each* tuple (i, k, c_i) would take time $O(n^2)$ due to the summing over the max; the entire table of α values would thus take time $O(n^4 \times |\mathcal{C}|)$. However, there is massive redundancy that can be avoided: First, for each tuple (i, k), only two possible values of $\alpha_{i \leftarrow k}(c_i)$ exist: one for $c_i \ni k$ (i.e., $\alpha_{i \leftarrow k}(\phi(k))$) and one for $c_i \not\ni k$ (i.e., $\alpha_{i \leftarrow k}(\overline{\phi}(k))$). Hence, instead of computing $|\mathcal{C}|$ values for each tuple (i, k), we need to compute and store only 2 values. Second, the expressions $\sum_{i' \neq k} \max_{c_{i'}} \rho_{i' \to k}(c_{i'})$ and $\sum_{i' \neq k} \max_{c_{i'} \not\ni k} \rho_{i' \to k}(c_{i'})$ depend on k but not on i; hence, they can be reused for many tuples (i, k). Third:

$$\begin{aligned}
\sum_{i' \notin \{i,k\}} \max_{c_{i'}} \rho_{i' \to k}(c_{i'}) &= \sum_{i' \neq k} \max_{c_{i'}} \rho_{i' \to k}(c_{i'}) \; - \max_{c_i} \rho_{i \to k}(c_i) \\
\sum_{i' \notin \{i,k\}} \max_{c_{i'} \not\ni k} \rho_{i' \to k}(c_{i'}) &= \sum_{i' \neq k} \max_{c_{i'} \not\ni k} \rho_{i' \to k}(c_{i'}) \; - \max_{c_i \not\ni k} \rho_{i \to k}(c_i)
\end{aligned}$$

Hence, after computing each of the terms of the LHS above (just once for each k), we need only to "adjust" them for each i, in $O(1)$ time, by subtracting the corresponding term on the RHS. At the end of all the CAP iterations, we set $c_i^{\text{MAP}} = \arg\max_{c_i} [\sum_k \alpha_{i \leftarrow k}(c_i) + S(i, c_i)]$. Hence, as long as we can update α during each iteration of message passing, then we never need to know ρ explicitly.

For convenience, define the following functions:

$$\begin{aligned}
b(i,k) &= \max_{c_i} \rho_{i \to k}(c_i) \\
\overline{b}(i,k) &= \max_{c_i \not\ni k} \rho_{i \to k}(c_i) \\
e(k) &= \sum_{i' \neq k} \max_{c_{i'}} \rho_{i' \to k}(c_{i'}) = \sum_{i' \neq k} b(i', k) \\
\overline{e}(k) &= \sum_{i' \neq k} \max_{c_{i'} \not\ni k} \rho_{i' \to k}(c_{i'}) = \sum_{i' \neq k} \overline{b}(i', k) \\
h(k) &= \rho_{k \to k}(k) \\
a(i,k) &= \alpha_{i \leftarrow k}(\phi(k)) = \begin{cases} e(k) & i = k \\ h(k) + e(k) - b(i,k) & i \neq k \end{cases} \\
\overline{a}(i,k) &= \alpha_{i \leftarrow k}(\overline{\phi}(k)) = \begin{cases} \overline{e}(k) & i = k \\ \max(\overline{b}(k,k) + \overline{e}(k) - \overline{b}(i,k), h(k) + e(k) - b(i,k)) & i \neq k \end{cases}
\end{aligned}$$

Visual inspection of Equation B.5 confirms that the $a(i, k)$ and $\overline{a}(i, k)$ defined above recover all $2n^2$ degrees of freedom of α. Below we show how we can compute e,$\overline{e}$,b, $\overline{b}$,

and h in a *total* time of $O(dn^{d+1})$ per iteration. First, however, we need to derive the computation of some intermediate quantities.

Appendix B.1. Computing $q(i, c_i) = \sum_{k'} \alpha_{i \leftarrow k'}(c_i)\ \forall i, c_i$

Define $q(i, c_i) = \sum_{k'} \alpha_{i \leftarrow k'}(c_i)$. For each i, we can compute $q(i, c_i)$ for each c_i by splitting the sum over k' into two parts: those k' such that $\phi(k') \ni c_i$ and those k' such that $\overline{\phi}(k') \ni c_i$. We then substitute $\alpha_{i \leftarrow k'}(c_i) = \alpha_{i \leftarrow k'}(\phi(k'))$ for k' s.t. $\phi(k') \ni c_i$ (and similarly for $\overline{\phi}(k')$) to yield:

$$\begin{aligned} q(i, c_i) &= \sum_{k' : \phi(k') \ni c_i} \alpha_{i \leftarrow k'}(\phi(k')) + \sum_{k' : \overline{\phi}(k') \ni c_i} \alpha_{i \leftarrow k'}(\overline{\phi}(k')) \\ &= \sum_{k'} \alpha_{i \leftarrow k'}(\overline{\phi}(k')) + \sum_{k' : \phi(k') \ni c_i} (\alpha_{i \leftarrow k'}(\phi(k')) - \alpha_{i \leftarrow k'}(\overline{\phi}(k'))) \\ &= \sum_{k'} \alpha_{i \leftarrow k'}(\overline{\phi}(k')) + \sum_{k' \in c_i} (\alpha_{i \leftarrow k'}(\phi(k')) - \alpha_{i \leftarrow k'}(\overline{\phi}(k'))) \end{aligned}$$

We can define $q^*(i) = \sum_{k'} \alpha_{i \leftarrow k'}(\overline{\phi}(k'))$. Then we have $q(i, c_i) = q^*(i) + \sum_{k' \in c_i}(a(i, k') - \overline{a}(i, k'))$. The term $q^*(i)$ takes time $O(n)$ for each i but is reused for all c_i. The summation on the RHS contains at most d terms (for a maximum composition size of d). Hence, for each i, the total computation (over all c_i) is $O(n + |\mathcal{C}|d) = O(n + dn^d) = O(dn^d)$.

Appendix B.2. Efficiently Finding Maxima of Many Subsets

The next step we need is an efficient method to compute expressions of the forms (a) $\max_{c_i \in \phi(k)} q(i, c_i)$ and (b) $\max_{c_i \in \overline{\phi}(k)} q(i, c_i)$ for all k, in a total time of $O(n^{d+1})$.

Form (a): Since each such c_i must contain k, then there are only $d - 1$ remaining degrees of freedom for each $\phi(k)$; hence, $|\phi(k)| \leq n^{d-1}$ for each k, and directly computing the maximum of $q(i, \cdot)$ over every $\phi(k)$ takes a total time of $O(n^d)$ (summed over all k).

Form (b): Define $\overline{\phi}^j(k) = \{c \in \overline{\phi}(k) : |c| = j\}$. Since $\max_{c_i \in \overline{\phi}(k)} q(i, c_i) = \max_{j \in [d]} \max_{c_i \in \overline{\phi}^j(k)} q(i, c_i)$, we can split the task into subtasks by j and then take the max over all of them. To compute the max over each $\overline{\phi}^j(k)$, we can iterate over all n^{j-1} tuples $(t_1, \ldots, t_{j-1}) \in [n]^{j-1}$; for each tuple, we can compute in $O(n)$ time the largest and second-largest value of $q(i, \cdot)$ over the set $(t - 1, \ldots, t_{j-1}, t_j)$ and then "adjust" the result in constant time to obtain the update for each k. In particular, for each such tuple τ, let $\psi_\tau = \{\{t_1, \ldots, t_j\} \in \mathcal{C} : t_1 < \ldots < t_j\}$. (For instance, if $j = 2$, $n = 4$, $\tau = (1, 2)$, and $\mathcal{C}$ contains all 3-tuples,

then $\psi_\tau = \{\{1,2,3\},\{1,2,4\}\}$.) In each iteration, let $c^1, c^2 \in \psi_\tau$ be the arguments corresponding to the largest and second-largest elements in $q(i, \psi_\tau)$; if $|\psi_\tau| = 1$, then define $c^2 = \emptyset$; if $|\psi_\tau| = 0$, then define both $c^1 = c^2 = \emptyset$. (Note that $\emptyset \notin \overline{\phi}(k)$ for any k.) For any k, it must be the case that the number of elements in the set $\psi_\tau \cap \overline{\phi}^j(k)$ is either 0 (if any $k \in \{t_1, \ldots, t_{j-1}\}$), $|\psi_\tau| - 1$ (if $k > t_{j-1}$, such that we must ignore exactly one element of ψ_τ for each k), or $|\psi_\tau|$ (if $k \notin \{t_1, \ldots, t_{j-1}\}$ and $k < t_{j-1}$). In the first case (intersection is empty), we make no update to $\max_{c_i \in \overline{\phi}^j(k)} q(i, c_i)$. In the second (intersection is of size $|\psi_\tau| - 1$), we update $\max_{c_i \in \overline{\phi}^j(k)} q(i, c_i)$ with $q(i, c^1)$ if $c^1 \not\ni k$ and with $q(i, c^2)$ otherwise. And in the third (intersection is of size $|\psi_\tau|$), we always update $\max_{c_i \in \overline{\phi}^j(k)} q(i, c_i)$ with $q(i, c^1)$. Since c^1, c^2 can be computed in time $|\psi_\tau| \leq n$ and then reused for *each* of the k (in constant-time) for the updates, and since there are at most n^{j-1} such tuples τ when scanning the entire $\mathcal{C}$, then this amounts to a total time of $O(dn^j)$ for each j. Summing over all $j = 1, \ldots, d$, this yields a running time of $O(dn^d)$. See Algorithm 2. The $\arg\max^{1,2}$ function returns the c^1, c^2 that give the largest and second-largest values of the specified function, where $c^2 = \emptyset$ if the input set is of size 1, and $c^1 = c^2 = \emptyset$ if the input set is empty.

Appendix B.3. Computing Maxes of Sums Except Row k

We can now show how expressions of the form $b(i', k) = \max_{c_{i'}} \rho_{i \to k}(c_{i'})$ and $\overline{b}(i', k) = \max_{c_{i'} \not\ni k} \rho_{i \to k}(c_{i'})$ can be computed efficiently. We first examine the former, which by definition is:

$$\max_{c_{i'}} \rho_{i' \to k}(c_{i'}) = \max_{c_{i'}} \left[S(i', c_{i'}) + \sum_{k' \neq k} \alpha_{i' \leftarrow k'}(c_{i'}) \right]$$

In other words, we need to find the $c_{i'}$ that maximizes $S(i', c_{i'})$ plus the sum (except the kth term) of the $\alpha_{i \leftarrow k'}(c_{i'})$ (see Figure B.5). As mentioned above, for each i', k, function $\alpha_{i' \leftarrow k}(\cdot)$ has only 2 degrees of freedom: one for $c_{i'} \in \phi(k)$ (the blue regions in Figure B.5) and one for $c_{i'} \in \overline{\phi}(k)$ (the clear regions); hence, there exist numbers u, v such that $\alpha_{i' \leftarrow k}(\phi(k)) = u$ and $\alpha_{i' \leftarrow k}(\overline{\phi}(k)) = v$. Assume we have already computed $q(i, c_{i'}) = \sum_{k'} \alpha_{i' \leftarrow k'}(c_{i'})\ \forall c_{i'}$ (this is the sum over *all* k) and also, for each k, the values $r(k) = \max_{c_{i'} \in \phi(k)} \sum_{k'} \alpha_{i' \leftarrow k'}(c_{i'})$ and $s(k) = \max_{c_{i'} \in \overline{\phi}(k)} \sum_{k'} \alpha_{i' \leftarrow k'}(c_{i'})$. Then, for any k, we can find, in $O(1)$ time, $\max_{c_{i'}} \sum_{k' \neq k} \alpha_{i' \leftarrow k'}(c_{i'})$ by "adjusting" $\max_{c_{i'}} \sum_{k'} \alpha_{i' \leftarrow k'}(c_{i'})$ as

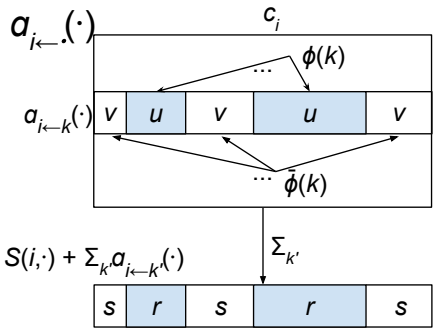

Figure B.5: For each i, k, to compute the max (over $c_{i'}$) of the sum of all rows $k' \neq k$, we can (1) compute the max of the sum of *all* rows within region $\phi(k)$ and (separately) within region $\overline{\phi}(k)$; (2) adjust each maximum by subtracting the value of row k in region $\phi(k)$ and the value of row k in region $\overline{\phi}(k)$, respectively; (3) take the larger result.

follows:

$$\max_{c_{i'}} \sum_{k' \neq k} \alpha_{i' \leftarrow k'}(c_{i'}) = \max(r(k) - u, s(k) - v)$$

The latter case ($\max_{c_{i'} \not\ni k} \rho_{i' \to k}(c_{i'})$) is even easier since we ignore all $c_{i'} \in \phi(k)$ entirely:

$$\max_{c_{i'} \not\ni k} \sum_{k' \neq k} \alpha_{i' \leftarrow k'}(c_{i'}) = s(k) - v$$

We have already defined $u = a(i', k)$ and $v = \overline{a}(i', k)$; hence, we have:

$$b(i', k) = \max(r(k) - a(i', k), s(k) - \overline{a}(i', k)), \qquad \overline{b}(i', k) = s(k) - \overline{a}(i', k)$$

Appendix B.4. Computing $h(k) = \rho_{k \to k}(k)$

As the last step, we can compute $h(k) = \rho_{k \to k}(k) = S(k, \{k\}) + \sum_{k' \neq k} \alpha_{k \leftarrow k'}(k) = S(k, \{k\}) + q_k(\{k\}) - a(k, k)$. This completes the derivation of Algorithm 1.

Appendix B.5. Time Cost Analysis

As explained in Section Appendix B.2, the FindAllMaxes takes time $O(dn^d)$ operations for each i. The function ComputeRhoStats calls FindAllMaxes n times (and also executes $O(n^2)$ further operations) for a cost of dn^{d+1}. The function ComputeAlphaStats takes $O(n^2)$ for the nested for-loops, and (as explained in Section Appendix B.1) a further $O(dn^d)$ for the computation of each $q(i, \cdot)$, amounting to $O(dn^{d+1})$ in total.

This completes the proof.

Appendix C. Brute-Force Reassignment

Here is how a brute-force reassignment could work: We first obtain a set of k singleton clusters with associated exemplar indices $\mathcal{E} \subset [n]$. Then we iterate over every possible subset $\tilde{\mathcal{E}} \subseteq \mathcal{E}$; these represent the compositional clusters. For each $\tilde{\mathcal{E}}$, we conduct an inner-loop to iterate over every possible 1-to-1 map from $\tilde{\mathcal{E}}$ to the set of compositions of $\mathcal{E} \setminus \tilde{\mathcal{E}}$; these represent the singleton clusters. If we consider compositions of at most d exemplars, then we have $\sum_{i=0}^{k} C(k,i) P\left(\sum_{d'=2}^{d} C(k-i,d'), i\right)$ total possible maps, where $C(k,i)$ and $P(k,i)$ are the numbers of combinations and permutations of i objects from a set of k, respectively. The P arises due to iterating over all 1-to-1 maps. Note that the number of possible maps grows factorially with k, and hence it quickly becomes intractable as k grows (e.g., for $k = 15$ and $d = 2$, the number of possibilities is 107770296705436).

Appendix D. LibriSpeech

LibriSpeech contains 1000+ hours of recorded English-language speech of people reading audiobooks. While the dataset contains speech from only individual speakers, we can synthesize speech by adding the waveforms of multiple speakers. Figure 2 shows of an example of how simultaneous speech data is synthesized from LibriSpeech data.

Compositional embedding model: Speaker embeddings were extracted from mel-frequency cepstrum coefficient (MFCC) features (32 coefficients, 0.025s window size, 0.01s step size) using an embedding function f^{emb} that contains a 2-layer LSTM with 256 hidden units. Composition function g is defined as $g(x_a, x_b) = W_1 x_a + W_1 x_b + W_2(x_a \odot x_b)$, where W_1, W_2 are learnable weights and x_a, x_b are speaker embeddings. f^{emb} and g were optimized jointly. During training, 15 audio samples from 5 unique speakers (5 labeled with 1 speaker and 10 with 2 speakers) are used to extract reference speaker embeddings using f^{emb}. 20 query speaker embeddings were extracted from the same 5 speakers using $f^{\text{emb}}\&g$, with audio or audio pairs. The distances between reference embeddings and query embeddings are computed and the model is optimized using triplet loss so that the distance between a reference-query pair share the same label is smaller then that of other pairs. After training, the model achieves overall accuracy of 86.9% on a validation set where each episode contains 20 queries as above.

After function f^{emb} and g are trained, we selected hyperparameters based on a separate validation set and then tested on test set. Both the validation set and test set contain 10 groups of data, and all clusters have the same number of samples in each group of data. (For example, in the setting of $l = 3, n = 120$, there are 6 clusters with labels $\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}$ and each one has 20 samples.) For all methods, hyperparameters are selected for $l = 3$ and for $l = 5$ (both n=150 and n=495) separately. For CAP/CAP$\subset$ and AP, there is only one hyperparameter, γ, which we varied over the set $\{1, 2, \ldots, 7\}$. For AC, there is a distance threshold hyperparameter, which we varied over $\{1, 2, 3, 4, 4\}$. These sets of values were chosen in pilot experimentation to give a fair chance to each algorithm; in particular, they were chosen so that the best result, during the validation process, did not fall on the boundary of these sets. During the message-passing process, we dampened the values returned by ComputeAlphaStats and ComputeRhoStats using a damping value of $\lambda = 0.65$: $\text{Val} = \text{OldVal} * \lambda + \text{NewVal} * (1 - \lambda)$. This value for λ was used for CAP, CAP$\subset$, and AP.

Appendix E. OmniGlot

OmniGlot contains images of handwritten symbols from a variety of languages. Figure 4 shows 4 groups of examples used in the experiment. Each group contains images with cluster labels $\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}$. **Compositional embedding model**: For the image embedding function f^{emb} we used ResNet18. Composition function g is defined the same as for LibriSpeech. The training procedure of f^{emb} and g are the same as for LibriSpeech. After training, the embedding model achieves overall accuracy of 75.0% on validation set. After function f^{emb} and g are trained, the hyperparameters are selected in the same way, and from the same sets, as in the LibriSpeech experiment. We used damping just like for LibriSpeech.