Learning a Self-Expressive Network for Subspace Clustering

Shangzhi Zhang†, Chong You‡, René Vidal§ and Chun-Guang Li†
† School of Artificial Intelligence, Beijing University of Posts and Telecommunications
‡ Department of EECS, University of California, Berkeley, CA
§ Mathematical Institute for Data Science, Johns Hopkins University, Baltimore, MD

Abstract

State-of-the-art subspace clustering methods are based on the self-expressive model, which represents each data point as a linear combination of other data points. However, such methods are designed for a finite sample dataset and lack the ability to generalize to out-of-sample data. Moreover, since the number of self-expressive coefficients grows quadratically with the number of data points, their ability to handle large-scale datasets is often limited. In this paper, we propose a novel framework for subspace clustering, termed Self-Expressive Network (SENet), which employs a properly designed neural network to learn a self-expressive representation of the data. We show that our SENet can not only learn the self-expressive coefficients with desired properties on the training data, but also handle out-of-sample data. Besides, we show that SENet can also be leveraged to perform subspace clustering on large-scale datasets. Extensive experiments conducted on synthetic data and real world benchmark data validate the effectiveness of the proposed method. In particular, SENet yields highly competitive performance on MNIST, Fashion MNIST and Extended MNIST and state-of-the-art performance on CIFAR-10. The code is available at: https://github.com/zhangsz1998/Self-Expressive-Network

1. Introduction

With technological advances in data acquisition, storage and processing, there is a surge in the availability of large-scale databases in computer vision. While the development of modern machine learning techniques, such as deep learning, has led to great success in analyzing big data, such methods require a large amount of annotated data which is often costly to obtain. Extracting patterns and clusters from unlabeled big data has become an important open problem.

We consider the problem of clustering large-scale unlabeled data under the assumption that each cluster is approximated by a low-dimensional subspace of the high-dimensional ambient space, a.k.a. subspace clustering [62, 63]. This problem has wide applications in image clustering [23, 17], motion segmentation [12, 9], hybrid system identification [61, 5], cancer subtype clustering [44, 32], hyperspectral image segmentation [86] and so on.

Self-expressive model [16] is one of the most popular and successful methods for subspace clustering. Given a data matrix $X = [\mathbf{x}_1, \cdots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$ whose columns are drawn from a union of $n$ subspaces, the self-expressive model expresses each data point $\mathbf{x}_j \in \mathbb{R}^D$ as a linear combination of other data points, i.e.,

$$\mathbf{x}_j = \sum_{i \neq j} c_{ij} \mathbf{x}_i,$$

where $\{c_{ij}\}_{i \neq j}$ are self-expressive coefficients. A remarkable property of the self-expressive model is that solutions to (1) that minimize certain regularization function on the coefficients have the subspace-preserving property, i.e., nonzero coefficients $c_{ij}$ occur only between $\mathbf{x}_i$ and $\mathbf{x}_j$ lying in the same subspace \[16, 17, 37, 41, 56, 68, 83, 79, 76, 40\]. Consequently, correct clustering can be obtained by defining an affinity between any pair of data points $\mathbf{x}_i$ and $\mathbf{x}_j$ as, e.g., $|c_{ij}| + |c_{ji}|$, and applying spectral clustering to the affinity. Recent developments further extend the applicability of self-expressive models to the case where the data are corrupted by noise \[66, 57, 67\] and outliers \[56, 82\], are imbalanced over classes \[77\], or possess missing entries \[59\].

Despite its great empirical performance and broad theoretical guarantees for correctness, the self-expressive model suffers from the limitation that it requires solving for a self-expressive matrix of size $N \times N$, which is computationally prohibitive for large-scale data. Although scalable subspace clustering methods based on subsampling \[51\], sketching \[58\] or learning a compact dictionary \[3, 54\] already exist, they do not have broad theoretical guarantees for correctness and sacrifice accuracy for scalability. In addition, the self-expressive coefficients computed for a set of data cannot be used to produce self-expressive coefficients for previously unseen data, posing challenges for learning in an online setting and for out-of-sample data.

In this work, we introduce the self-expressive network
(SENet) to learn a self-expressive model for subspace clustering, which can be leveraged to handle out-of-sample data and large-scale data. Our method is based on learning a function $f(x_i, x_j; \Theta) : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$, implemented as a neural network with parameters $\Theta$, that is designed to satisfy the self-expressive model

$$x_j = \sum_{i \neq j} f(x_i, x_j; \Theta) \cdot x_i. \quad (2)$$

In principle, the number of network parameters does not need to scale with the number of points in the dataset, hence SENet can effectively handle large scale data. Moreover, an SENet trained on a certain dataset can be used to produce self-expressive coefficients for another dataset drawn from the same data distribution, therefore the method can handle out-of-sample data effectively. We present a network architecture for $f(x_i, x_j; \Theta)$ as well as a training algorithm that allow us to learn self-expressive coefficients with desired subspace-preserving properties. Our experiments showcase the effectiveness of our method as summarized below:

1. We show that the self-expressive coefficients computed by a trained SENet closely approximate those computed by solving for them directly without the network. This illustrates the ability of SENet to approximate the desired self-expressive coefficients.

2. We show that a SENet trained on (part of) the training set of MNIST and Fashion MNIST can be used to produce self-expressive coefficients on the test set that give a good clustering performance. This illustrates the ability of SENet to handle out-of-sample data.

3. We show that SENet can be used to cluster datasets containing 70,000+ data points, such as MNIST, Fashion MNIST and Extended MNIST, very efficiently, achieving a performance that closely matches (for MNIST, Fashion MNIST and Extended MNIST) or surpasses (for CIFAR-10) the state of the art.

2. Related Work

Deep Clustering. Our work is fundamentally different from many existing studies on jointly training a deep neural network and learning self-expressive coefficients [50, 24, 49, 90, 74, 89, 88] for subspace clustering. In such methods, deep networks are used to extract features (so that they lie in linear subspaces) from input data (which may not lie in linear subspaces), and self-expressive model is applied in the feature space [21, 1]. In contrast, our work assumes that the input data already lie in linear subspaces, and focuses on computing the self-expressive coefficients. Our work also shares similarities with SpectralNet [53], which learns a neural network to produce a latent embedding by optimizing a spectral clustering objective on an affinity graph. Such a method does not have a low-dimensional modeling for data therefore is different from ours.1

Self-expressive Models. Many works have explored different choices of regularization on the self-expressive coefficients for subspace clustering. For instance, $\ell_1$ regularization is used in sparse subspace clustering [16, 17], for which the optimal solution is subspace-preserving when the subspaces are independent, disjoint, intersecting or even affine [17, 56, 66, 67, 83, 33, 80, 52]; nuclear norm and $\ell_2$ norm regularization are used in low-rank [37, 18] and least squares subspace clustering [41], respectively, for which the optimal solution is subspace-preserving when the subspaces are independent; mixing $\ell_1$ norm with either $\ell_2$ or nuclear norm regularization are used in [79] and [68], respectively, to improve connectivity of affinity graph while maintain broad theoretical guarantees for subspace-preserving property. In addition, there are works on noise modeling [30, 36, 22] and feature learning [38, 47, 48, 24, 90, 88, 84] for self-expressive models.

Scalable Subspace Clustering. Due to its importance in practical applications, large scale subspace clustering has drawn a lot of research attentions. An early work [51] presented a subsampling based approach in which a random subset of data is sampled and clustered, then the rest of the data are classified with sparse representation based classification [69]. Following this work, several methods adopt a two-step approach for computing self-expressive coefficients: 1) construct a dictionary, either generated in random [58] or learned/selected from data [54, 3, 77, 4, 2, 43], and 2) express each data point as a linear combinations of the atoms in the dictionary. In particular, motivated by the development of learned optimization solvers such as LISTA [20] and ISTA-Net [87] for solving sparse optimization problems, [34, 35] presented a framework where one jointly solves for the self-expressive coefficients and trains a neural network to approximate self-expressive coefficients with a dictionary in the first step, so that the computation of self-expressive coefficients in the second step can be carried out efficiently. In principle, the clustering performance of such a two-step approach increases with the size of the dictionary. However, the output dimension hence the scale of the optimization problem in [34, 35] increases at least quadratically with the size of the dictionary, therefore using a sufficiently large dictionary may be impossible.

Another group of methods achieve efficient computation by decomposing a large-scale optimization problem into a sequence of small scale problems, by either a greedy approach [81, 13], active support method [79], or dropout strategy [10]. These methods enjoy broad theoretical guar-

---

1When finalizing the submission, we became aware of a work-in-progress report [7] that presents a similar idea as ours. While [7] uses $\ell_2$ regularization and imposes symmetry on self-expressive coefficients, our model uses a general elastic net regularization and does not impose symmetry constraint, therefore has a better capability of obtaining subspace-preserving properties.
antees for correctness and have superior empirical performance. Nonetheless, they have quadratic time and memory requirement, therefore cannot handle very large scale data.

**Self-attention Models.** The self-attention mechanism used in Graph Attention Networks (GAT) [60], Transformer [27], Non-local Neural Networks [65], etc., shares similar idea with the self-expressive models. In these works, the (output) features of one data point are computed as a linear combination of (input) features of all data points. Similar to SENet, the coefficients in the linear combination are computed with a neural network. However, unlike the self-expressive models, which use the distance between the input features and output features to define a training loss in an unsupervised manner, the self-attention methods impose a supervised learning loss on the output features. This leads to a difference in the design of the network architecture, as we explain in the next section.

3. Self-Expressive Network

3.1. Model

Let \( X = [x_1, \cdots, x_N] \in \mathbb{R}^{D \times N} \) be a data matrix whose columns lie in a union of low-dimensional linear subspaces of \( \mathbb{R}^D \). Self-expressive methods for subspace clustering are based on solving for every \( j \in \{1, \cdots, N\} \) an optimization problem of the form

\[
\min_{\{c_{ij}\}, \gamma} \frac{\gamma}{2} \| x_j - \sum_{i \neq j} c_{ij} x_i \|_2^2 + \sum_{i \neq j} r(c_{ij}), \tag{3}
\]

where \( r(\cdot) : \mathbb{R} \to \mathbb{R}_+ \) is a regularization function and \( \gamma > 0 \) is a balancing parameter. The idea is that any column \( x_j \) can be expressed as a linear combination of other columns of \( X \) that are from the same subspace as \( x_j \). Such a linear combination is known as *subspace-preserving*, and it can be recovered by solving (3) with certain choices of regularization \( r(\cdot) \). Aggregating the solutions to (3) for all columns of \( X \) yields a self-expressive coefficient matrix \( C \in \mathbb{R}^{N \times N} \) with the \( i, j \)-th entry given by \( c_{ij} \). When \( C \) is subspace-preserving, spectral clustering [64] on an affinity given by, e.g., \( |C| + |C^\top| \), produces correct clustering of the data matrix \( X \).

We present a method that is based on solving the following optimization problem in lieu of (3):

\[
\min_{\Theta} \frac{\gamma}{2} \| x_j - \sum_{i \neq j} f(x_i, x_j; \Theta) x_i \|_2^2 + \sum_{i \neq j} r(f(x_i, x_j; \Theta)), \tag{4}
\]

where \( f(x_i, x_j; \Theta) : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R} \) is a function parameterized by \( \Theta \). There are two benefits of using the model in (4) over the model in (3).

First, the number of parameters in (3) (collectively for all \( j \in \{1, \cdots, N\} \)) is quadratic with the number of data points \( N \), which limits its applicability to large scale datasets since an \( N \times N \) matrix may not fit into memory. In contrast, the number of parameters in (4) needs not be related to the number of data points, and can be determined flexibly based on the availability of the memory. In principle, the model in (4) may be used to compute self-expressive coefficients for datasets of arbitrary size.

Second, self-expressive coefficients computed from (3) for a particular dataset cannot be used for another dataset that is drawn from the same distribution. This implies that the model in (3) cannot be used to handle out-of-sample data, for which self-expressive coefficients need to be computed from scratch. In contrast, a self-expressive function in (4) once learned on a particular dataset can be used to generate self-expressive coefficients for out-of-sample data. By our design of the network architecture for \( f(\cdot, \cdot; \Theta) \) as we discuss in Subsection 3.2, the calculation on out-of-sample data can be carried out very efficiently.

**Choice of Regularization** \( r(\cdot) \). It is known that sparsity regularization in self-expressive models enforces subspace-preserving properties under broadest conditions [16, 17, 83, 67, 59, 33, 80]. For example, the work [56, 67] showed that with \( \ell_1 \) regularization on the coefficients, the model in (3) produces subspace-preserving solutions even when the subspaces intersect, provided that the subspaces are sufficiently separated and points in each subspace are well-distributed. On the other hand, sparsity regularization produces solutions that have too many false negatives, i.e., the self-expressive coefficient \( c_{ij} \) can often be zero even when \( x_i \) and \( x_j \) are from the same subspace. This may lead to a poorly connected affinity graph that results in over-segmentation. Hence, the work [79] advocated using elastic regularization, which is given by a weighted sum of \( \ell_1 \) and \( \ell_2^2 \) regularization with a balancing parameter \( \lambda \in [0, 1] \):

\[
r(\cdot) = \lambda |\cdot| + \frac{1 - \lambda}{2} |\cdot|^2. \tag{5}
\]

This regularizer provably produces subspace-preserving solutions under similar conditions as for the \( \ell_1 \) regularizer, and at the same time produces a denser coefficient matrix, hence an improved clustering performance. Therefore, we adopt elastic net regularization for our model in (4).

3.2. Network Instantiation

Inspired by recent advances in deep learning, we implement the self-expressive function \( f(\cdot, \cdot; \Theta) \) in our model (4) via a deep neural network with training parameters \( \Theta \). We refer to the network as *Self-Expressive Network (SENet)*.

Specifically, we propose the following network formulation for SENet:

\[
f(x_i, x_j; \Theta) = \alpha T_0(u_j^\top v_i), \tag{6}
\]
where
\[
\begin{align*}
  u_j &:= u(x_j; \Theta_u) \in \mathbb{R}^p, \\
v_i &:= v(x_i; \Theta_v) \in \mathbb{R}^p.
\end{align*}
\]

In above, \(u(\cdot; \Theta_u)\) and \(v(\cdot; \Theta_v)\), referred to as query and key networks, are two multilayer perceptrons (MLPs) that perform mappings \(\mathbb{R}^D \rightarrow \mathbb{R}^p\) with learnable parameters \(\Theta_u\) and \(\Theta_v\), respectively, where \(p\) is a model parameter. \(\mathcal{T}_b(\cdot)\) is a learnable soft thresholding operator defined as
\[
\mathcal{T}_b(t) := \text{sgn}(t) \max(0, |t| - b),
\]
where \(b\) is a learnable parameter and \(\alpha > 0\) is a fixed numerical constant. For clarity, we use \(\Theta := \{\Theta_u, \Theta_v, b\}\) to denote all the trainable parameters in SENet, and illustrate the architecture of the neural network \(f(x_i, x_j; \Theta)\) in Fig. 1.

By the design of network architecture in (6), the self-expressive coefficient for a pair of data points \((x_j, x_i)\) is computed by learning a pair of representations \(u_j\) and \(v_i\), respectively, and taking the inner product of \(u_j\) and \(v_i\) before applying a soft-thresholding. We empirically find (see Sec. 4.1) that such a network can produce self-expressive coefficients that well approximate the solution to (3), which justifies its ability in obtaining the desired subspace-preserving and denser connection properties.²

An important benefit of the design in (6) is that the computation of the self-expressive coefficient matrix for a given data matrix \(X\) can be made very efficient. In particular, instead of evaluating \(f(x_i, x_j; \Theta)\) for all possible (i.e., \(N^2\) number of) pairs of \((x_i, x_j)\), one may evaluate \(u(\cdot; \Theta_u)\) and \(v(\cdot; \Theta_v)\) separately for all columns of \(X\), which can be parallelized. After that, the coefficient matrix can be obtained by computing inner product between pairs of \((u_j, v_i)\), which again can be parallelized and computed very efficiently. Such a property also allows us to train the network efficiently as we explain in the next subsection.

Comparison to Self-attention Models. The network architecture in (6) bears a close resemblance to the self-attention model in Transformer [27], Non-Local Neural Networks [65], and Graph Attention Networks [60], etc., which also aim to compute self-expressive coefficients for a set of signal (e.g., sequence, image, video, nodes on graph) representations. However, we note that our choice of architecture in (6) has several favorable properties over the self-attention models.

- The functions \(u(\cdot)\) and \(v(\cdot)\) in self-attention models are linear maps, while we use MLPs for our SENet. This design is to increase the expressive power of SENet to gain universal approximation ability so that it can easily approach the optimal solution for the convex formulation in (3) and hence enjoy subspace-preserving property.
- The self-attention model usually adopts a normalization factor such that each self-expression is given by a convex combination. Such a requirement is, however, too restrictive for our purpose: for sample points that lie in a vertex of the convex hull of sample points in one of the subspaces, they cannot be expressed as a convex combination of other points. In such cases, self-attention models cannot produce subspace-preserving solutions.
- We adopt a soft-thresholding operator at the output of SENet, borrowed from learned sparse optimization networks such as LISTA [20] and ISTA-Net [87], to enforce sparsity of the output. This is to account for the fact that the solution to the model in (4) with the elastic net regularization in (5) is expected to be sparse (due to the \(\ell_1\) norm inside).

### 3.3. Training

We train SENet in (6) via solving the following optimization problem:

\[
\min_{\Theta} \mathcal{L}(X; \Theta) := \sum_{j=1}^{N} \ell(x_j, X; \Theta),
\]

where \(\ell(x_j, X; \Theta)\) is the objective function in (4), i.e.,
\[
\ell(x_j, X; \Theta) := \gamma \frac{1}{2} \| x_j - \sum_{i \neq j} f(x_i, x_j; \Theta) x_i \|_2^2 + \sum_{i \neq j} r(f(x_i, x_j; \Theta)).
\]

Then, the network parameters \(\Theta\) can be learned by Stochastic Gradient Descent (SGD). We summarize the algorithm (assuming that batch size is 1 for simplicity) in Algorithm 1.

Since the loss \(\ell(x_j, X; \Theta)\) depends on the entire data \(X\) (for any fixed \(x_j\)), the memory requirement for Algorithm 1 scales linearly with the number of data points. This restricts the ability of the algorithm to handle very large scale data. Next, we present a two-pass algorithm that is equivalent to Algorithm 1 but with constant memory complexity.

**Two-pass SGD Algorithm.** To derive our algorithm, we

²An analysis of its approximation power is left as future work.
Algorithm 1 A Naive SGD Algorithm for Training SENet

1: **Input:** Dataset $X \in \mathbb{R}^{D \times N}$, model parameters $\gamma > 0$ and $\lambda \in [0, 1]$, number of iterations $T$, learning rate $\eta$
2: **Initialization:** Random initialize SENet parameters $\Theta$
3: **for** each $t \in \{1, \ldots, T\}$ **do**
4: Sample a data point $x_j$ from $X$
5: **# Forward propagation to compute the loss**
6: Compute $u_j = u(x_j, \Theta_v)$
7: Load data $X$ and compute $V \triangleq [v_1, \ldots, v_N]$, where $v_i \equiv v(x_i, \Theta_v)$
8: Compute $f(X, x_j; \Theta) = \alpha \mathcal{T}_0(V^\top u_j)$
9: Compute $\ell(x_j, X; \Theta)$ from $f(X, x_j; \Theta)$ by (11)
10: **# Backward propagation to compute the gradient**
11: Compute $d\Theta = \frac{\partial \ell(x_j, X; \Theta)}{\partial \Theta}$
12: **# Gradient descent to update $\Theta$**
13: Set $\Theta \leftarrow \Theta - \eta \cdot d\Theta$
14: **end for**
15: **Output:** SENet with trained weights.

compute the gradient in step 11 of Algorithm 1 as
\[
\frac{\partial \ell(x_j, X; \Theta)}{\partial \Theta} = \sum_{i \neq j} r'(f(x_i, x_j; \Theta)) - \langle x_i, q_j \rangle \frac{\partial f(x_i, x_j; \Theta)}{\partial \Theta},
\]
(12)
where
\[
q_j := \gamma (x_j - \sum_{i \neq j} f(x_i, x_j; \Theta)x_i),
\]
(13)
and $r'(\cdot)$ denotes the derivative\(^3\) of $r(\cdot)$. Observe that if the vector $q_j$ in (12) is given, then the right hand side of (12) is a weighted sum of gradient computed at each data point $x_i$ for $i = 1, \ldots, N$. Therefore, it can be accumulated in an online fashion with constant space requirement (see step 14 - 20, Algorithm 2). Moreover, although $q_j$ is unknown, it can be computed by performing a separate forward propagation (and no backward propagation is needed). In particular, $q_j$ can be computed by subtracting the summation $\sum_{i \neq j} f(x_i, x_j; \Theta)x_i$ from $x_j$, where the summation term can be accumulated in an online fashion with constant space requirement as well (see step 6 - 13, Algorithm 2). Overall, this leads to a two-pass algorithm for training SENet as described in Algorithm 2.

Since the memory requirement for Algorithm 2 does not scale with the number of data points, in principle it can handle arbitrarily large datasets.

### 4. Experiments

We conduct extensive experiments on both synthetic data and real world benchmark datasets to evaluate the performance of SENet.

**Network Architecture.** For both the query and key networks in (7) and (8), we use a three-layer MLP with ReLU and tanh(·) as the activation functions for hidden layers and the output layer, respectively. The number of hidden units in each layer of the MLPs are $\{1024, 1024, 1024\}$, and the output dimension $p$ is 1024. By using tanh(·) as the output layer activation, the inner product of the output vectors $u_j$ and $v_i$ is bounded by $p$, i.e., $u_j^\top v_i \in (-p, p)$. Therefore, we use a small scalar multiplier $\alpha = \frac{1}{p}$ as in (6) to scale down the output of SENet. We use the Adam [25] optimizer with an initial learning rate of $10^{-3}$ and use the cosine annealing learning rate decay [39] with gradient clipping.

**Metrics.** Given a self-expressive coefficient matrix $C$, we use the subspace recovery error (SRE), defined as the proportion of the $\ell_1$ norm of $C$ that comes from the wrong subspace, to measure the subspace-preserving property of $C$. In addition, we use the algebraic connectivity (CONN) [45], defined as the second smallest eigenvalue of the normalized graph Laplacian of each ground-truth class minimized over all classes, to measure the connectedness of the affinity graph. As discussed in Subsection 3.1, we desire that $C$ has low SRE and high CONN. We refer the reader to

---

\(^3\)As $r(t)$ is not differentiable at $t = 0$, we set $r'(0) = 0$ which is in the sub-differential of $r(t)$ at $t = 0$. 

---

Algorithm 2 A Two-pass Algorithm for Training SENet

1: **Input:** Dataset $X \in \mathbb{R}^{D \times N}$, model parameters $\gamma > 0$ and $\lambda \in [0, 1]$, number of iterations $T$, learning rate $\eta$
2: **Initialization:** Random initialize SENet parameters $\Theta$
3: **for** each $t \in \{1, \ldots, T\}$ **do**
4: Sample a data point $x_j$ from $X$
5: Compute $u_j = u(x_j, \Theta_v)$
6: **# First pass (forward only): compute $q_j$**
7: Initialize $\bar{x} = 0$
8: **for** each $i \in \{1, \ldots, j - 1, j + 1, \ldots, N\}$ **do**
9: Load data $x_i$ and compute $v_i = v(x_i, \Theta_v)$
10: Compute $f(x_i, x_j; \Theta) = \alpha \mathcal{T}_0(u_j^\top v_i)$
11: Set $\bar{x} \leftarrow \bar{x} + f(x_i, x_j; \Theta)x_i$
12: **end for**
13: Set $q_j = \gamma (x_j - \bar{x})$
14: **# Second pass: compute gradient $d\Theta$**
15: Initialize $d\Theta = 0$
16: **for** each $i \in \{1, \ldots, j - 1, j + 1, \ldots, N\}$ **do**
17: Load data $x_i$ and compute $v_i = v(x_i, \Theta_v)$
18: Compute $f(x_i, x_j; \Theta) = \alpha \mathcal{T}_0(u_j^\top v_i)$
19: Set $d\Theta \leftarrow d\Theta + \langle x_i, q_j \rangle \frac{\partial f(x_i, x_j; \Theta)}{\partial \Theta}$
20: **end for**
21: **# Gradient descent to update $\Theta$**
22: Set $\Theta \leftarrow \Theta - \eta \cdot d\Theta$
23: **end for**
24: **Output:** SENet with trained weights.
[81] for a detailed explanation of these two quantities.

To evaluate the clustering performance, we report clustering accuracy (ACC), normalized mutual information (NMI) and adjusted rand index (ARI) which are commonly used in the literature (see e.g., [84] for a definition).

### 4.1. Experiments on Synthetic Data

**Visualization of Self-expressive Coefficients.** We demonstrate the ability of SENet to produce self-expressive coefficients and generalize to out-of-sample data on synthetic data. For that purpose, we generate a synthetic dataset as in [81], where 5 subspaces of dimension 6 are sampled uniformly at random in the ambient space $\mathbb{R}^d$ (i.e., $n = 5$, $d = 6$ and $D = 15$), and 200 points are sampled uniformly at random on the unit sphere of each subspace. We randomly select 500 data points as training data $X_{\text{tr}}$ and the remaining 500 data points as testing data $X_{\text{ts}}$. We set the parameters $\gamma = 50.0$ and $\lambda = 0.9$ and use Algorithm 1 to train our SENet on $X_{\text{tr}}$ with maximum iteration $T_{\text{max}} = 500$. Then we take the trained SENet at the $t$-th iteration to evaluate and infer the matrices of self-expressive coefficients $C_{\text{tr}}^{(t)}$ and $C_{\text{ts}}^{(t)}$ on $X_{\text{tr}}$ and $X_{\text{ts}}$, respectively. A visualization of $|C_{\text{tr}}^{(t)}|$ and $|C_{\text{ts}}^{(t)}|$ is given as colored images in Fig. 2. We observe of that SENet is able to efficiently learn self-expressive coefficients that are approximately subspace-preserving after only a few hundred iterations and that the trained SENet is able to infer self-expressive coefficients for out-of-sample data with reasonably good quality. Note that spectral clustering could yield perfect result after training with 300 iterations.

![Visualization of self-expressive coefficients](image)

Figure 2. Visualization of self-expressive coefficients computed by SENet trained with \{100, 300, 500\} iterations on synthetic data where the percentage number in bracket is SRE.

**Comparing SENet to EnSC.** We demonstrate the ability of SENet to approximate the solution to (3) with $r(\cdot)$ being the elastic net regularization function in (5), which is a method known as EnSC [79]. For that purpose, we use the same parameters $\gamma = 50.0$ and $\lambda = 0.9$ for SENet and EnSC models, so that they solve the same optimization problems except that EnSC directly optimizes over the self-expressive coefficients while SENet optimizes over the parameters of a network that generates the coefficients.

We sample 5 subspaces of dimension 6 in the ambient space $\mathbb{R}^d$ (i.e., $n = 5$, $d = 6$ and $D = 9$), then sample $N_t$ data points from the unit sphere of each subspace with $N_t \in \{20, 100, 200, 1000, 2000\}$. We measure the difference between EnSC and SENet solutions by reporting the total loss ($\mathcal{L}$) in (10), as well as the reconstruction loss and regularization loss:

$$\mathcal{L}_{\text{rec}} = \sum_{j=1}^{N} \|x_j - \sum_{i \neq j} f(x_i, x_j; \Theta) x_i \|_2^2,$$

$$\mathcal{L}_{\text{reg}} = \sum_{j=1}^{N} \sum_{i \neq j} r(f(x_i, x_j; \Theta)).$$

We also report SRE, CONN and ACC. The results are shown in Table 1. We can see that the difference between the solution by SENet and EnSC is relatively small, indicating the strong approximation power of the SENet architecture. On the other hand, such a difference increases with $N_t$, showing that a larger (e.g., deeper and wider) network may be needed. By examining the values of SRE and CONN we can see that such a difference causes higher subspace-preserving error, but it helps improve the connectivity of the affinity graph.

To evaluate the generalization ability of the trained SENet, we prepare a set of test data that consists of $N_t$ data points per subspace sampled uniformly at random from the union of subspaces model that is used to generate the training data. Then, the trained SENet is used to directly infer the self-expressive coefficients on test data. The results are reported in the rows “SENet test” of Table 1. We can see that the trained SENet shows increasingly better ability to detect subspace structures when the number of data points per subspace is increased. Moreover, we can see that while $\mathcal{L}_{\text{reg}}$ is similar in scale to that given by SENet on the train data, the $\mathcal{L}_{\text{rec}}$ is significantly higher. This shows that the generalization ability of SENet is on detecting subspace structures, not on the reconstruction.

### 4.2. Experiments on Real World Datasets

We further evaluate the performance of SENet on four larger benchmark datasets: MNIST [29], Fashion-MNIST [71], CIFAR-10 [26] and Extended MNIST (EMNIST) [11].

**MNIST** contains 70,000 grey-scale images of handwritten digits “0” to “9”, which we denote as MNIST-full. The MNIST-full is divided into MNIST-train and MNIST-test, consisting of 60,000 and 10,000 images, respectively.
Methods

| $N_i$ | Methods | $\mathcal{L}$ | $\mathcal{L}_{rec}$ | $\mathcal{L}_{reg}$ | ACC (%) | SRE (%) | CONN |
|-------|---------|---------------|-------------------|------------------|---------|--------|------|
| 20    | EnSCI   | 139.132       | 0.107             | 132.442          | 72.0    | 49.611 | 0.178|
|       | SENet   | 139.132       | 0.109             | 132.416          | 71.0    | 49.720 | 0.178|
|       | SENet test | 1830.107     | 72.007            | 28.937           | 65.0    | 58.384 | 0.318|
| 100   | EnSCI   | 585.945       | 0.326             | 558.009          | 93.0    | 27.370 | 0.163|
|       | SENet   | 585.945       | 0.331             | 558.022          | 92.8    | 27.501 | 0.163|
|       | SENet test | 2955.424     | 89.325            | 702.509          | 79.0    | 56.897 | 0.387|
| 200   | EnSCI   | 1055.086      | 0.326             | 1040.009         | 96.0    | 20.195 | 0.159|
|       | SENet   | 1055.086      | 0.331             | 1040.009         | 96.0    | 20.195 | 0.159|
|       | SENet test | 17826.275    | 999.099           | 2486.779         | 84.1    | 56.256 | 0.398|
| 1000  | EnSCI   | 14835.076     | 2.015             | 1483.506         | 94.0    | 4.907  | 0.126|
|       | SENet   | 14835.076     | 2.020             | 1487.781         | 99.5    | 0.132  | 0.155|
|       | SENet test | 38073.012   | 887.325           | 7853.985         | 92.3    | 36.054 | 0.220|

Table 1. Comparing SENet to EnSC on synthetic data

Fashion-MNIST contains 70,000 grey-scale images of various types of fashion products, denoted as Fashion-MNIST-full. Fashion products (e.g., coat, trouser, shirt, dress, bag, etc.) with different styles correspond to 10 categories. Similar to MNIST, Fashion-MNIST-full is divided into Fashion-MNIST-train and Fashion-MNIST-test, consisting of 60,000 and 10,000 images, respectively. EMNIST contains grey-scale images of handwritten digits and letters where 190,998 images of the 26 lower case letters are used for the clustering problem with 26 categories. For these three datasets, we compute a feature vector of dimension 3,472 using the scattering convolution network [6], which extracts translational invariant and deformation stable features, and then reduce the dimension to 500 using PCA. CIFAR-10 contains 60,000 color images in 10 classes, where each image is of size $32 \times 32$. For CIFAR-10, we use the feature representation extracted by MCR$^2$ [84], which learns a union-of-subspace representation from data with self-supervised learning. All feature vectors are normalized to have unit $\ell_2$ norm.\(^4\)

To produce a segmentation from the self-expressive coefficient matrix, we compute an affinity matrix by either a) constructing a 3-nearest neighbor graph from the columns of $C$ as in [77] (for MNIST, Fashion-MNIST and EMNIST), or b) using $|C| + |C|\top$ (for CIFAR-10). Then, spectral clustering is applied to the affinity matrix.\(^5\)

Generalization Performance of SENet. We evaluate the generalization ability of SENet to out-of-sample data using MNIST and Fashion-MNIST. Specifically, we select $N \in \{200, 500, 1000, 2000, 5000, 10000, 20000\}$ data points uniformly at random from MNIST-train and train SENet for 100,000 iterations with a batch size fixed to 100 (likewise for Fashion-MNIST-train). Then, we take MNIST-test as test data for which the trained SENet is used to generate self-expressive coefficients and apply spectral clustering on the induced affinity for producing a segmentation (likewise for Fashion-MNIST-test). For EnSC, we directly compute the self-expressive coefficients on MNIST-test and Fashion-MNIST-test.

\(^4\)For EMNIST, we also remove the mean after PCA as in [77].

\(^5\)For MNIST and FashionMNIST, we use the eigenvectors corresponding to the 15 smallest eigenvalues of graph Laplacian to perform $k$-means.

Experimental results are reported in Table 2. We can see that with the increasing amount of training data, SENet is able to approach or surpass the performance of EnSC, which is directly optimized on the test data. This confirms that the trained SENet enjoys a promising generalization ability to out-of-sample data in real world datasets.

Subspace Clustering on Large-Scale Datasets. We demonstrate that SENet can effectively handle large-scale datasets MNIST-full (70k), Fashion-MNIST-full (70k), CIFAR-10 (60k) and EMNIST (190k). For each dataset, we randomly select $N$ points to train SENet, then apply the trained SENet to generate self-expressive coefficients on the entire dataset. At the end, spectral clustering is applied to obtain the segmentation. In EnSC and SENet, we use $\gamma = 200.0$ and $\lambda = 0.9$ for MNIST, Fashion-MNIST and CIFAR-10, and $\gamma = 150.0$ and $\lambda = 1.0$ for EMNIST.

In Fig. 3, we report the training time and clustering accuracy with varying $N$. The experiments are conducted on a single NVIDIA GeForce 2080Ti GPU (for EMNIST) or 1080Ti GPU (for all other datasets). We also compare with EnSC, for which the active support solver in [79] is used to compute the self-expressive coefficients on the entire datasets. Since there is no available GPU acceleration packages for this solver, we run EnSC using an Intel(R) Xeon E5-2630 CPU. The results confirm that our SENet is able to achieve reasonably good performance while using only a small amount of data. This leads to a significantly reduced training time. For EMNIST, as EnSC needs more than 24 hours, we instead compare SENet to ESC [77] in which 300 exemplars are used. Note that SENet achieves comparable performance as ESC within an acceptable time, showing its potential to handle large-scale datasets.

We further compare the performance of SENet to other methods in the literature, including $k$-means [42], spectral clustering with normalized cuts (Spectral) [55], elastic net subspace clustering (EnSC) [79], sparse subspace clustering by orthogonal matching pursuit (SSC-OMP) [81], neural collaborative subspace clustering (NCSC) [89] and exemplar-based subspace clustering (ESC) [77]. We also compare SENet to several state-of-the-art deep image clustering algorithms, including deep embedded clustering (DEC) [72], joint unsupervised learning (JULE) [73],

| Methods | Train Data: # | MNIST-test | EnSC | SENet |
|---------|----------------|------------|------|------|
|         |                | ACC (%)    | SRE (%) | ACC (%) | SRE (%) |
| NA      | 79.15          | 4.455      | 60.55 | 21.712 |
| 500     | 82.60          | 8.846      | 63.65 | 24.430 |
| 1000    | 80.87          | 7.290      | 70.46 | 23.502 |
| 2000    | 95.45          | 5.131      | 58.71 | 22.197 |
| 5000    | 95.80          | 4.785      | 60.67 | 21.109 |
| 10000   | 96.66          | 4.121      | 62.92 | 20.385 |
| 20000   | 96.25          | 3.978      | 64.64 | 20.442 |

Table 2. Generalization performance of SENet on MNIST-test and Fashion-MNIST-test.
Figure 3. Clustering accuracy vs. training time with varying training data sizes. SENet-N denotes SENet trained on N data points.

| Methods       | MNIST-full | Fashion-MNIST-full | CIFAR-10 | EMNIST |
|---------------|------------|---------------------|----------|--------|
|               | ACC        | NMI                 | ARI      |        |
| k-means [42]  | 0.541      | 0.507               | 0.362    |        |
| Spectral [55] | 0.728      | 0.856               | 0.667    |        |
| JULE [73]     | 0.964      | 0.913               | 0.927    |        |
| DEC [72]      | 0.863      | 0.834               |          |        |
| DAC [8]       | 0.978      | 0.935               | 0.949    |        |
| DEPICT [19]   | 0.965      | 0.917               |          |        |
| ClusterGAN [46]| 0.905    | 0.890               |          |        |
| DSCDAN [75]   | 0.978      | 0.941               |          |        |
| DCCM [70]     |            |                     |          |        |
| SSC-OMP [81]  | 0.928      | 0.842               | 0.849    |        |
| NCSC [89]     | 0.941      | 0.861               | 0.875    |        |
| EnSc [79]     | 0.980      | 0.945               | 0.957    |        |
| ESC [77]      | 0.971      | 0.925               | 0.936    |        |
| SENet         | 0.968      | 0.918               | 0.931    |        |

|                | ACC        | NMI                 | ARI      |
| MNIST-full     | 0.525      | 0.589               | 0.276    |
| Spectral [55]  | 0.455      | 0.574               | 0.256    |
| JULE [73]      | 0.272      | 0.192               | 0.138    |
| DEC [72]       | 0.301      | 0.257               | 0.161    |
| DAC [8]        | 0.522      | 0.396               | 0.306    |
| DEPICT [19]    |            |                     |          |
| ClusterGAN [46]|            |                     |          |
| DSCDAN [75]    |            |                     |          |
| DCCM [70]      |            |                     |          |
| SSC-OMP [81]   | 0.326      | 0.498               |          |
| NCSC [89]      | 0.863      | 0.834               |          |
| EnSc [79]      | 0.928      | 0.842               |          |
| ESC [77]       | 0.759      |                     |          |
| SENet         | 0.765      | 0.655               | 0.573    |

Table 3. Image clustering results on MNIST-full, Fashion-MNIST-full, CIFAR-10 and EMNIST. The best results are in bold font and the second best results are underlined. ‘T’ means the computation time exceeds 24 hours.

Deep adaptive image clustering (DAC) [8], deep embedded regularized clustering (DEPICT) [19], ClusterGAN [46], deep spectral clustering using dual autoencoder network (DSCDAN) [75] and deep comprehensive correlation mining (DCCM) [70].

Experimental results are reported in Table 3. We can see that our SENet is among the best performing methods on the four benchmarks. Specifically, SENet consistently outperforms previous subspace clustering methods on CIFAR-10, i.e., +15.2% on CIFAR-10 compared to EnSC in terms of accuracy. Although trained on sampled small datasets, our SENet could still achieve a comparable performance on MNIST-full with significantly reduced training time. Meanwhile, our SENet also achieves comparable performance when compared to state-of-the-art deep image clustering methods. In particular, our SENet outperforms all baseline methods on CIFAR-10 and achieves second highest accuracy on Fashion-MNIST and EMNIST.

5. Conclusion

We proposed a novel self-expressive network (SENet) for discovering low-dimensional subspace structures in high-dimensional data and presented two stochastic gradient descent (SGD) based training algorithms to effectively train SENet. Different from the conventional self-expressive model, which is defined on the given dataset only and cannot handle out-of-sample data, our proposed SENet is trained on the given dataset and can generalize to unseen new samples. We conducted extensive experiments on synthetic data and real world data and showed that the self-expressive coefficients learned by SENet are equally good or even better than the self-expressive coefficients learned by a convex self-expressive model. Moreover, we verified that the out-of-sample ability enables SENet to efficiently handle large-scale dataset. Beyond the clustering task, self-expressive models also have wide applications in classification [69], exemplar selection [15, 78], outlier/novelty detection [85, 82], and matrix completion [14, 31, 28] tasks as well, we believe that our SENet may also be extended for many of such tasks, and leave it to future work.

Acknowledgment

S. Zhang and C.-G Li are supported by the National Natural Science Foundation of China under Grant 61876022. C.-G. Li is the corresponding author. R. Vidal was partially supported by the Northrop Grumman Mission Systems Research in Applications for Learning Machines (REALM) initiative, NSF Grants 1704458, 2031985 and 1934979. C. You acknowledges support from the Tsinghua-Berkeley Shenzhen Institute Research Fund.
References

[1] Maryam Abdolali and Nicolas Gillis. Beyond linear subspace clustering: A comparative study of nonlinear manifold clustering algorithms. arXiv preprint arXiv:2103.10656, 2021. 2

[2] Maryam Abdolali, Nicolas Gillis, and Mohammad Rahmati. Scalable and robust sparse subspace clustering using randomized clustering and multilayer graphs. Signal Processing, 163:166–180, 2019. 2

[3] A. Adler, M. Elad, and Y. Hel-Or. Linear-time subspace clustering via bipartite graph modeling. IEEE Transactions on Neural Networks and Learning Systems, 26(10):2234–2246, 2015. 1, 2

[4] Akram Aldroubi, Ali Sekmen, Ahmet Bugra Koku, and Ahmet Faruk Cakmak. Similarity matrix framework for data from union of subspaces. Applied and Computational Harmonic Analysis, 2017. 2

[5] Laurent Bako and René Vidal. Algebraic identification of MIMO SARX models. In International Workshop on Hybrid Systems: Computation and Control, pages 43–57. Springer-Verlag, 2008. 1

[6] Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(8):1872–1886, 2013. 7

[7] Julian Busch, Evgeniy Faerman, Matthias Schubert, and Thomas Seidl. Learning self-expression metrics for scalable and inductive subspace clustering. arXiv preprint arXiv:2009.12875, 2020. 2

[8] Jianlong Chang, Lingfeng Wang, Gaofeng Meng, Shining Xiang, and Chunhong Pan. Deep adaptive image clustering. In IEEE International Conference on Computer Vision, 2017. 8

[9] G. Chen and G. Lerman. Spectral curvature clustering (SCC). International Journal of Computer Vision, 81(3):317–330, 2009. 1

[10] Ying Chen, Chun-Guang Li, and Chong You. Stochastic sparse subspace clustering. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4155–4164, 2020. 2

[11] Nadav Cohen, Or Sharir, Yoav Levine, Ronen Tamari, David Yakira, and Amnon Shashua. Analysis and design of convolutional networks via hierarchical tensor decompositions. arXiv preprint arXiv:1705.02302, 2017. 6

[12] Joao Paulo Costeira and Takeo Kanade. A multibody factorization method for independently moving objects. International Journal of Computer Vision, 29(3):159–179, 1998. 1

[13] Eva L. Dyer, Aswin C. Sankaranarayanan, and Richard G. Baraniuk. Greedy feature selection for subspace clustering. Journal of Machine Learning Research, 14(1):2487–2517, 2013. 2

[14] E. Elhamifar. High-rank matrix completion and clustering under self-expressive models. In Neural Information Processing Systems, 2016. 8

[15] E. Elhamifar, G. Sapiro, and R. Vidal. See all by looking at a few: Sparse modeling for finding representative objects. In IEEE Conference on Computer Vision and Pattern Recognition, 2012. 8

[16] E. Elhamifar and R. Vidal. Sparse subspace clustering. In IEEE Conference on Computer Vision and Pattern Recognition, pages 2790–2797, 2009. 1, 2, 3

[17] E. Elhamifar and R. Vidal. Sparse subspace clustering: Algorithm, theory, and applications. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(11):2765–2781, 2013. 1, 2, 3

[18] Paolo Favaro, René Vidal, and Avishek Ravichandran. A closed form solution to robust subspace estimation and clustering. In IEEE Conference on Computer Vision and Pattern Recognition, pages 1801–1807, 2011. 2

[19] Kamran Ghasedi Dizaji, Amirhossein Herandi, Cheng Deng, Weidong Cai, and Heng Huang. Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In IEEE International Conference on Computer Vision, pages 5736–5745, 2017. 8

[20] K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In International Conference on Machine Learning, pages 399–406. Omnipress, 2010. 2, 4

[21] Benjamin David Haeffele, Chong You, and Rene Vidal. A critique of self-expressive deep subspace clustering. In International Conference on Learning Representations, 2021. 2

[22] R. He, L. Wang, Z. Sun, Y. Zhang, and B. Li. Information theoretic subspace clustering. IEEE Transactions on Neural Networks and Learning Systems, 27(12):2643–2655, 2016. 2

[23] J. Ho, M.-H. Yang, J. Lim, K.-C. Lee, and D.-J. Kriegman. Clustering appearances of objects under varying illumination conditions. In Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition, pages 11–18, 2003. 1

[24] P. Ji, T. Zhang, H. Li, M. Salzmann, and I. Reid. Deep subspace clustering networks. In Neural Information Processing Systems (NIPS), 2017. 2

[25] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. International Conference on Learning Representations, 2014. 5

[26] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 6

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Neural Information Processing Systems, pages 5998–6008, 2017. 3, 4

[28] Connor Lane, Ron Boger, Chong You, Manolis Tsakiris, Benjamin Haeffele, and Rene Vidal. Classifying and comparing approaches to subspace clustering with missing data. In IEEE International Conference on Computer Vision Workshops, 2019. 8

[29] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998. 6
Appendices

In the appendix, we provide additional experimental results for SENet. In particular, we provide a set of ablation studies on CIFAR-10 to evaluate the effect of using the learnable soft thresholding activation, varying the number of hidden layers, the number of hidden neurons and the batch size in training. We provide the learning curves on both synthetic data and real world data. We also report the running time of SENet on datasets used in our experiments. Finally, we provide a comparison of Algorithm 1 and Algorithm 2 to demonstrate the scalability of SENet.

A. Additional Results on Synthetic Datasets

Learning Curves. We report the losses, ACC and SRE during training, evaluated on both the training and testing data, for the experiments discussed in the paragraph “Visualization of Self-expressive Coefficients” of Section 4.1. The results are shown in Fig. 4.

We see that on the training data, the loss $L$ and its two components $L_{\text{rec}}$ and $L_{\text{reg}}$ all monotonically decrease as training proceeds. Correspondingly, the ACC monotonically increases and SRE monotonically decreases. On the other hand, we see that $L_{\text{rec}}$ on test data monotonically increases, showing that the SENet does not learn to reconstruct for test data. Nonetheless, the SRE monotonically decreases on test data, showing that the SENet learns to produce subspace-preserving solutions which leads to an improved ACC. We leave a study of why this behavior occurs to future work.

B. Additional Results on Real Datasets

B.1. Effects of Network Design / Training Choices

We provide additional results for our experiments on clustering 60,000 images in the CIFAR-10 dataset (see Section 4.2). In all experiments below, we train a SENet on $N = 2,000$ random samples from CIFAR-10 and use it to generate self-expressive coefficients on the entire dataset for clustering.

SENet with vs. without Soft-Thresholding Layer. To demonstrate the necessity of the soft-thresholding layer in (6), we conduct experiments both with and without the soft-thresholding layer and report results in Table 4. We can see that the performance of SENet improves significantly with the learnable soft-thresholding layer.

| Methods                  | ACC    | NMI    | ARI |
|--------------------------|--------|--------|-----|
| With soft-thresholding   | 0.765  | 0.655  | 0.573|
| Without soft-thresholding| 0.610  | 0.588  | 0.424|

Table 4. Ablation study of soft-thresholding layer on CIFAR-10.

Effect of Network Depth. We vary the number of layers of the query and key networks in the range of $\{1, 2, 3, 4\}$ while fixing the network width to 1024. In Table 5, we report average and standard deviation for ACC, NMI, ARI and SRE over 5 trials. We can see that the performance of SENet increases when increasing the number of hidden layers from 1 to 3. However, further increasing the number of layers beyond three does not help to improve the performance. Note that even using a single hidden layer, SENet still works well.

| Depth | ACC    | NMI    | ARI    |
|-------|--------|--------|--------|
| 1     | 0.759 ± 0.002 | 0.634 ± 0.001 | 0.554 ± 0.002 |
| 2     | 0.732 ± 0.022 | 0.636 ± 0.012 | 0.545 ± 0.016 |
| 3     | 0.755 ± 0.016 | 0.647 ± 0.009 | 0.566 ± 0.016 |
| 4     | 0.751 ± 0.013 | 0.643 ± 0.008 | 0.560 ± 0.011 |

Table 5. Effect of network depth on CIFAR-10. We report mean and standard deviation over 5 trials.

Effect of Network Width. We vary the network width (i.e., number of neurons in each hidden as well as output layers of the key and query MLPs) in the range of $p \in [8, 2048]$ while fixing the number of layers to be 3. The parameter $\alpha$ is set to be $1/p$. We report the mean (as solid lines) and standard deviation (as shaded areas) for ACC, NMI, ARI and SRE in Fig. 5. We can see that the performance continues to improve when the network becomes wider.

![Figure 5. Effect of network width on CIFAR-10. We report mean and standard deviation over 5 trials.](image)

Effects of Batch Size. We evaluate the effect of batch size in the training algorithm and report the mean and standard deviation of ACC, NMI, ARI and SRE in Fig. 6. The figure shows that the best batch size should be neither too large nor too small and a batch size of $\sim 100$ produces the best performance in terms of ACC and ARI.

B.2. Learning Curves

We plot the curves of the total loss $L$, the reconstruction loss $L_{\text{rec}}$ and the regularization term $L_{\text{reg}}$ during training for experiments on the MNIST, Fashion-MNIST and CIFAR-10 datasets as discussed in Section 4.2. The number of training data is fixed to 2,000. The results are shown in
Fig. 4, Fig. 9 and Fig. 10. We can observe that the total loss $\mathcal{L}$ and the reconstruction loss $\mathcal{L}_{\text{rec}}$ are steadily decreasing during training on all the three datasets. Nonetheless, the trends of the regularizer term $\mathcal{L}_{\text{reg}}$ are slightly different in these three datasets. The $\mathcal{L}_{\text{reg}}$ term shows a rapid growth at the early stage of the training in all three datasets but it tends to slightly increase on FashionMNIST and CIFAR-10 and slightly decrease on MNIST.

For CIFAR-10, we further plot ACC and SRE on the sampled training data (containing 2,000 data points), as well as on the entire CIFAR-10 dataset (containing 60,000 data points). The mean and standard deviation of the results computed over 5 trials are reported in Fig. 7. We can see that the ACC increases with the number of iterations while the SRE decreases first and then stabilizes.
B.3. Time Evaluation in Detail

Recall that Fig. 3 reports the running time of SENet for clustering MNIST, Fashion-MNIST and CIFAR-10 datasets. The running time can be decomposed into two parts: the first part is the training time for SENet on the subsampled training set, and the second part is the inference time for computing the self-expressive coefficients on the entire dataset with a trained SENet.

We report the training and inference time with varying sizes for the training set in Table 6. The result demonstrates that the inference speed of SENet is very fast and it also has a reasonably small training time. Together, this enables SENet to efficiently handle large-scale clustering problems.

B.4. Comparison of Algorithm 1 and Algorithm 2

In all aforementioned experiments, we train SENet with Algorithm 1. As discussed in Section 3.3, the memory consumption of Algorithm 1 grows with the training data size, preventing its application to large-scale datasets. In contrast, Algorithm 2, while being equivalent to Algorithm 1, has a fixed memory consumption and in principle can handle arbitrarily large datasets. While memory requirement does not become an issue for experiments in this paper, this section provides additional experiments to validate the advantage of Algorithm 2 over Algorithm 1, hence demonstrates the promise of SENet in handling data with larger scale than those used in this paper.

To illustrate the effect of training data size, we conduct experiments with $N \in \{1000, 2000, 5000, 10000, 20000\}$ data points sampled from MNIST-train. We set batch size (i.e., the number of samples in step 4 of Algorithm 1 and step 4 of Algorithm 2) to 100 in both algorithms. While the for-loops at both step 8 and step 16 of Algorithm 2 process one data point at each iteration, they can both be parallelized into processing multiple data points (referred to as a block) to reduce computational overheads hence accelerate the algorithm. For simplicity of presentation we fix the block size to be 1000 in our experiments, but point out that block size controls a trade-off between computation efficiency and memory requirement hence needs to be chosen according to the availability of computational resources in practice.

We conduct 4 independent trials with Algorithm 1 and Algorithm 2 and report running time per iteration, the GPU memory usage and clustering accuracy in Fig. 11. We can observe that the GPU memory consumption of Algorithm 2 does not change with the size of training data. Meanwhile, Algorithm 2 is slightly slower than Algorithm 1 since 1) it has the overhead of processing blocks at step 8 and step 16 rather than processing all data at once (as in Algorithm 1), and 2) it takes an additional forward pass. Finally, both algorithms yield almost the same clustering accuracy, hence verifying their equivalence.
Figure 9. Learning curves for training SENet on FashionMNIST.

Figure 10. Learning curves for training SENet on CIFAR-10.

Table 6. Training time (sec.) on MNIST-full, Fashion-MNIST-full and CIFAR-10.

| Methods | | MNIST | | Fashion-MNIST | | CIFAR-10 |
|---------|| Training | Inference | Total | Training | Inference | Total | Training | Inference | Total |
| EnSC | | - | - | 10252.71 | - | - | 10583.55 | - | - | 2237.14 |
| SENet | 200 | 1164.63 | 300.37 | 1465.00 | 1081.53 | 299.83 | 1381.36 | 499.31 | 13.53 | 512.85 |
| | 500 | 1070.80 | 276.20 | 1347.00 | 1180.33 | 304.13 | 1484.45 | 497.24 | 12.57 | 509.81 |
| | 1000 | 1213.36 | 271.89 | 1485.25 | 1102.98 | 279.98 | 1382.95 | 510.09 | 19.59 | 529.67 |
| | 2000 | 1423.43 | 254.59 | 1678.01 | 1384.23 | 310.21 | 1694.45 | 617.43 | 12.57 | 636.75 |
| | 5000 | 2331.42 | 267.80 | 2599.21 | 2277.91 | 330.74 | 2608.64 | 892.46 | 21.79 | 914.24 |
| | 10000 | 3921.22 | 282.14 | 4203.35 | 3865.35 | 329.59 | 4194.95 | 1509.32 | 19.27 | 1528.59 |
| | 20000 | 7316.09 | 285.82 | 7601.91 | 7161.92 | 343.26 | 7505.19 | 2982.29 | 20.38 | 3002.66 |

Figure 11. Comparison of Algorithm 1 and Algorithm 2 with varying dataset size on MNIST.