Feature Selection using Multiple Auto-Encoders

Xinyu Guo
Department of Electrical Engineering and Computing Systems
University of Cincinnati
Cincinnati, OH, USA
guox2@mail.uc.edu

Ali A. Minai
Department of Electrical Engineering and Computing Systems
University of Cincinnati
Cincinnati, OH, USA
ali.minai@uc.edu

Long J. Lu
Division of Biomedical Informatics
Cincinnati Children's Hospital Research Foundation
Cincinnati, OH, USA
long.lu@cchmc.org

***Abstract* — Real-world data such as medical images and sensor measurements is usually high-dimensional and limited. Using such datasets directly in machine learning tasks can lead to poor generalization. Feature learning is a general approach for transforming high-dimensional data points to a representational space with lower dimensionality. Machine learning models can be trained efficiently with such representations. In this paper, a novel feature selection method based on multiple trained sparse auto-encoders (SAEs) is described. It works by selecting diverse, non-redundant features from multiple pinched SAEs with very narrow hidden layers, and then using these features in a more appropriately sized classifier without further feature tuning. The feature learning ability of the method is evaluated in a handwritten digits recognition task. Results show that this type of feature selection provides improved representations for a softmax classifier, and that using pinched SAEs produces results equal to or better than regular SAEs.**

Keywords— Machine learning, feature learning, handwritten digits recognition, sparse auto-encoder, feature diversity, feature selection

I. INTRODUCTION

The idea of auto-encoders (AEs) has been part of the historical landscape of neural networks for decades [1, 2]. They can be considered a special case of feedforward neural networks that are trained to reconstruct their input as the output[3]. More specifically, an AE learns an encoder function which is capable of mapping the input space to a representation space, and a decoder function which can reconstruct the original input from representations [4]. It can be trained with gradient descent using back-propagation [5-7]. However, the goal is not simply to learn an identity function [8], but to squeeze redundant information out of the data by learning under strong constraints [9] – first by using hidden layers of size much smaller than the dimensionality of the data, and further through regularization, which reduces the degrees of freedom available to the system during learning, e.g., by enforcing sparsity of activity on neurons [10, 11]. Such networks have been applied successfully to data dimension reduction. For instance, Hinton et al. [12] designed a deep regularized AE with 4 gradually smaller hidden layers for learning a language model from data, and found that the input reconstruction error was lower than that for principal components analysis with 30 dimensions. In addition, the learned representation was qualitatively easier to interpret. More importantly, however, the mappings learned by hidden units in an AE can be used as features for constructing classifiers on the original input space.

A *sparse auto-encoder* (SAE) is a type of regularized AE which makes the representation layer sparse by adding a penalty term to the loss function during training. SAEs have been applied widely in machine learning applications for feature learning from data including images, video, and speech. Deng et al. [13] developed an SAE-based transfer learning framework for speech emotion recognition and evaluated it on six standard databases. Results showed that the approach performed better than when each source domain was learned independently. Zeiler et al. [14] proposed a very simple SAE method that can learn quite interpretable and discriminative features when using rectified linear units (ReLUs). Baccouche et al. [15] built a spatio-temporal convolutional SAE for video sequence classification. Compared to hand-crafted features, their neural models were able to automatically learn a sparse shift-invariant representation of the local $2D+t$ salient information.

Features learned automatically by SAEs often outperform handcrafted features. However, to obtain satisfactory performance, a large amount of data is needed for training the networks. For instance, in order to build high-level, class-specific feature detectors from only unlabeled data, Ziegler et al. [16] designed a deep SAE and trained it on a large dataset of images. The model had 1 billion connections while the dataset had 10 million 200×200 pixel images downloaded from the Internet. In some cases such as medical imaging, it is hard to get such a large amount of training data due to labor, expense, and lack of sources.

Recently, researchers have tried using multiple AEs to learn better representations of the data. For example, Zhao et al. [17] developed a multi-modal deep neural network-based framework to select informative, heterogeneous features from different feature groups. In this framework, each deep NN consists of several denoising AEs. They evaluated the framework on three image classification datasets, and the results showed that the approach was effective in selecting the relevant feature groups, and achieved classification performance competitive with several recent methods. Zhu et al. [18] developed a deep learning framework to learn multi-channel feature representations for the face recognition task. In their design, multiple deep learning networks are applied to extract representations of different parts of the face, e.g. eyes, nose and mouth, and the combined representations are used to

train a classification model. Good results were obtained for four datasets. Although multiple AEs have been used in the approaches described above, they were used as components of the deep learning framework. In addition, the training datasets in the tasks were of sufficient size. In the method we describe, the aim is to develop a feature selection method based only on multiple SAEs rather than several deep NNs, and the focus is more on the situation where the training dataset is relatively small.

II. SPARSE AUTO-ENCODER

A. *Sparse Auto-Encoder (SAE)*

An auto-encoder (AE) is a three-layer feed-forward neural network (NN) as shown in **Fig. 1.**

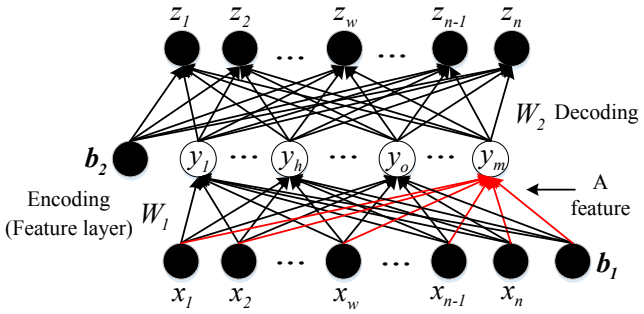

Fig. 1. The schematic structure of the AE.

It consists of an input layer, a hidden layer and an output layer. Each hidden layer unit receives input from all neurons in the input layer, and projects to every neuron of the output layer. The function of the AE is to reproduce the input stimulus pattern on the output layer through the intermediate hidden layer. The number of nodes in the input layer and the output layer are equal, and are determined by the data dimension. The AE can be used for feature extraction and dimensionality reduction when the number of hidden layer nodes is smaller than the number of input layer nodes. More specifically, the encoding part of the AE can be seen as the feature layer which can transform raw data from the input layer into a lower-dimensional representation in the hidden layer. A feature is defined by the pattern of weights of all connections between a specific hidden layer node and all input nodes. The AE can be turned into an SAE by imposing a sparsity constraint on the mean activity of the nodes in the hidden layer [19] to reduce overfitting.

Let the number of nodes in the input and hidden layers be denoted by n and m, respectively. Let $W_1 \in R^{(m\times n)}$ and $W_2 \in R^{(n\times m)}$ respectively denote the encoding weight matrix and the decoding weight matrix, and $\boldsymbol{b_1} \in R^m$ and $\boldsymbol{b_2} \in R^n$, respectively, the hidden layer and output layer bias vectors. The SAE is able to learn a latent, compressed representation of the input through minimizing the reconstruction error between the input and the reconstructed output from the learnt representation [20].

Let $\boldsymbol{x} = [x_1, x_2, ..., x_w, \ldots, x_{n-1}, x_n]$ be the input vector of the SAE, $\boldsymbol{y} = [y_1, ..., y_h, \ldots y_o, ..., y_m]$ the compressed representation vector of $\boldsymbol{x}$ in the hidden layer, and $\boldsymbol{z} = [z_1, z_2, ..., z_w, \ldots, z_{n-1}, z_n]$ the output vector. The SAE transfers $\boldsymbol{x}$ to $\boldsymbol{y}$ through a linear composition function and a nonlinear activation function f, so that $\boldsymbol{y}$ is the activation value of the corresponding hidden node in response to input $\boldsymbol{x}$: $\boldsymbol{y} = f(W_1\boldsymbol{x} + \boldsymbol{b}_1)$. The activation function used is the logistic sigmoid function in (1) which is widely used in neural networks, machine learning and pattern recognition tasks [16, 21-23].

$$f(u) = 1/(1+exp(-u)) \tag{1}$$

The representation $\boldsymbol{y}$ in the hidden layer is mapped to the output $\boldsymbol{z}$ by another linear composition function and the same activation function is used as follows:

$$\boldsymbol{z} = f(W_2\boldsymbol{y} + \boldsymbol{b}_2) \tag{2}$$

The reconstruction error is quantified by:

$$J(W, \boldsymbol{b}; \boldsymbol{x}^{(i)}, \boldsymbol{z}^{(i)}) = \frac{1}{2} \| \boldsymbol{z}^{(i)} - \boldsymbol{x}^{(i)} \|_2^2 \tag{3}$$

where $\boldsymbol{x}^{(i)}$ is the ith input vector from the training dataset and $\boldsymbol{z}^{(i)}$ is the corresponding output of the network for the given input. The cost function of the SAE is shown in (4):

$$Cost = J(W, \boldsymbol{b}) + \beta \sum_{j=1}^{m} KL(\rho \,\|\, \rho_j^{'}) \tag{4}$$

The first term $J(W, \boldsymbol{b})$ also has two parts, and is shown in (5).

$$J(W, \boldsymbol{b}) = \frac{1}{M} \sum_{i=1}^{M} J(W, \boldsymbol{b}; \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}) + \frac{\lambda}{2} \sum_{l=1}^{n_l - 1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_l + 1} (W_{ji}^{(l)})^2 \tag{5}$$

M denotes the total number of input vectors in the training dataset; n_l is the number of layers in the network (here n_l=3); s_l denotes the number of nodes in layer l; and $W_{ij}^{(l)}$ represents the weight associated with the connection between node j in layer l and unit i in layer $l+1$. The first term is for evaluating the average reconstruction error across the whole training set. The second term is a regularization term called weight decay. It tends to decrease the magnitude of the weights, and helps prevent overfitting. Here, λ is a parameter that controls the relative importance of the two terms.

The second term in the *Cost* function is a sparseness constraint which also acts as a regularizer. A hidden layer node is considered to be active if its output value is close to 1, and to be inactive if its output value is close to 0. To keep representations of the hidden layer sparse, most of the hidden layer neurons are constrained to be relatively inactive for any given input. Let $y_h(x)$ denote the activation of the hidden node h when the network is fed with an input $\boldsymbol{x}$. Define (6) as the average activation of the hidden node h over the training dataset.

$$\rho_h^{'} = \frac{1}{M} \sum_{i=1}^{M} \left[y_h(\boldsymbol{x}^{(i)}) \right] \tag{6}$$

A sparsity parameter ρ is a small value (e.g., 0.05) such that $\rho_h^{'}$ is required to be close to ρ. This means that each individual hidden node must be inactive for most inputs (e.g., active only in about 5% of cases). In order to achieve this, an extra penalty term is added to the optimization objective that penalizes $\rho_h^{'}$ deviating significantly from ρ. Since ρ and $\rho_h^{'}$ can be seen as the probabilities of Bernoulli random variables to be 1, a Kullback-Leibler (KL) divergence [16], denoted by $KL(\rho \,||\, \rho_h^{'})$, is used as the penalty term:

$$KL(\rho \,||\, \rho_h^{'}) = \rho log \frac{\rho}{\rho_h^{'}} + (1-\rho) log \frac{1-\rho}{1-\rho_h^{'}} \tag{7}$$

This penalty term has the characteristic that $KL(\rho \,||\, \rho_h^{'}) = 0$ only if $\rho_h^{'} = \rho$, and the value increases as $\rho_h^{'}$ deviates from ρ. The optimization of the SAE seeks to minimize the cost function so that the average reconstruction error from training input vectors is small while the sparsity of SAE activity is also maintained. Here, β is a parameter controlling the relative significance of the sparsity penalty.

B. Feature Visuization

The MNIST digits dataset [24] was used to explore the proposed approach. This is a popular dataset of handwritten digits from 0 through 9 that has been widely used for testing NN-based machine learning frameworks, and has effectively become a benchmark. It is publicly available online, and has a training set of 60,000 images, and a test set of 10,000 images. It is part of a larger set available from NIST (National Institute of Standards and Technology). Each image in the MNIST dataset has been sized and normalized, and has been centered in a fixed-size (28×28) image. Research on handwritten digit recognition algorithms has been pursued actively in both industry and academia [25-29].

The activation of the *i*th node (i = 1, 2, …, m) in the hidden layer of an SAE is given by (8), where $W_{ij}^{(1)}$ denotes the connection weight from the *j*th pixel (j = 1, 2, …, n) in the input layer to the *i*th node in the hidden layer, p_j denotes the intensity of the *j*th pixel in the input vector, b_i denotes the bias for the *i*th hidden node, and f is the sigmoid function:

$$a_i = f(\sum_{j=1}^{n} W_{ij}^{(1)} p_j + b_i^{(1)}) \tag{8}$$

The activation a_i can be seen as a feature tuned to a particular input pattern $\boldsymbol{x}$ which maximizes a_i. To specify this pattern uniquely, the input pixels must be norm constrained as:

$$|| x ||^2 = \sum_{i=1}^{n} p_i^2 \le 1 \tag{9}$$

Then the input that maximally activates hidden unit *i* is given by setting pixel p_j (for all n pixels, $j = 1,\ldots n$) to (10).

$$p_j = \frac{W_{ij}^{(1)}}{\sqrt{\sum_{j=1}^{n} (W_{ij}^{(1)})^2}} \tag{10}$$

By displaying the image formed by these pixel intensity values, it is possible to visualize what feature each hidden unit *i* is looking for. It is clear from (10) that the feature for hidden unit *i* is simply a scaled version of its weight matrix $W_{ij}^{(1)}$ so that it is reasonable to consider that $W_{ij}^{(1)}$ are the elements of the feature that hidden node *i* learns from the data. **Fig. 2** shows visualized features from a trained SAE with 200 nodes in the hidden layer, each with 784 pixel intensity values representing the feature for each node.

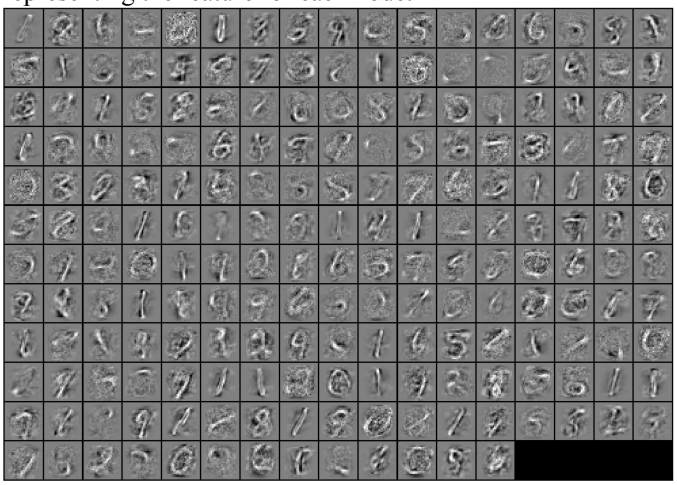

Fig. 2. Visualized features from a trained SAE.

III. FEATURE SELECTION METHOD

A. Conceptual Description of the Approach

The method we describe is motivated by the use of SAEs for feature learning in classification problems, as discussed in more detail in Section IV. In this task, the features learned by an SAE are used to map the high-dimensional input data to the lower-dimensional feature space before being provided to a classifier (e.g., a softmax classifier). For any given dataset, a sufficient number of features is needed to achieve reasonable classification performance. However, these features must not only be sufficient in number – they must also be sufficiently diverse to cover the representation space adequately for classification. While sparseness helps in achieving this diversity, a single SAE with a small hidden layer may not provide a sufficiently comprehensive set of features, while one with a large hidden layer often leads to duplicative features.

One possible way to obtain a sufficient set of diverse features is to harvest them from multiple SAEs trained independently on the same data. However, experience indicates that, while training multiple SAEs with large hidden layers will produce a sufficient diversity of features, they will also generate a lot of nearly identical redundant features, which is computationally wasteful. There are several potential ways to force multiple networks to produce a higher diversity of features, but in this study, we explore an especially simple approach motivated by the somewhat counter-intuitive hypothesis that SAEs that are too small (i.e., have insufficiently sized hidden layers) – and thus able to learn the

data only moderately well – are likely to *collectively* infer more diverse features because each such network is likely to make different choices under duress. The combined pool of features from several such suboptimal SAEs can then be sifted to obtain a sufficiently diverse set of features for use in a classifier of appropriate size. This heuristic is also computationally more efficient because the SAEs trained are smaller.

B. *Dissimilarity Analysis of Pairwise Features from Multiple SAEs*

The approach described above is operationalized by defining a class of *pinched SAE*s (pSAEs), which are sparse auto-encoders with very narrow hidden layers. These pSAEs are used to generate the feature pool from which a diverse set of features is selected. To compare the diversity between the feature pool generated by a set of normal SAEs and the pool produced by the same number of pinched SAEs, we conducted a dissimilarity analysis of pairwise features from multiple SAEs in each case.

Based on pilot trials, we chose our regular SAEs to have 200 hidden nodes and the pinched SAEs to have 40. The trials indicated that using 40 features can classify the data with about 65% accuracy, which is low but significantly better than random.

Dissimilarity analysis requires defining the dissimilarity between features, which can be done in several ways. We considered three possibilities:

1. A *HOG dissimilarity* metric, which first transforms each of the features into a *histogram of oriented gradients* (HOG) representation [33], and then calculates their Euclidean distance.

2. A *spatial dissimilarity* metric, which uses the lack of pixel-wise correlation between pairs of features.

3. A *response dissimilarity* metric, which measures the lack of correlation between the activation of feature pairs over the training data set.

As shown by the ultimate results (See Section IV-B), we found the HOG dissimilarity metric to be the most useful. To demonstrate the increase in the diversity of the feature pool for pSAEs compared to SAEs, we obtained features from 10 networks of each class trained on all the MNIST training data (60,000 data points). This produced a pool of 2,000 features for the regular SAEs and a pool of 400 features for the pSAEs. In each set, we considered each feature produced by a specific network and calculated its minimum distance from the features of all other networks in the set. Given that each set had 10 networks, there were 9 minimum dissimilarity numbers for each of the 2,000 original features in the SAE pool, resulting in 18,000 minimum dissimilarity values in all. Similarly, the pSAE pool with its 400 features produced 3,600 values. The training process and parameter setting were exactly the same in the two cases. **Fig. 3** illustrates the cumulative distribution functions of minimum dissimilarity features pairs from 10 SAEs with 200 hidden nodes for 10 SAEs with 40 hidden nodes.

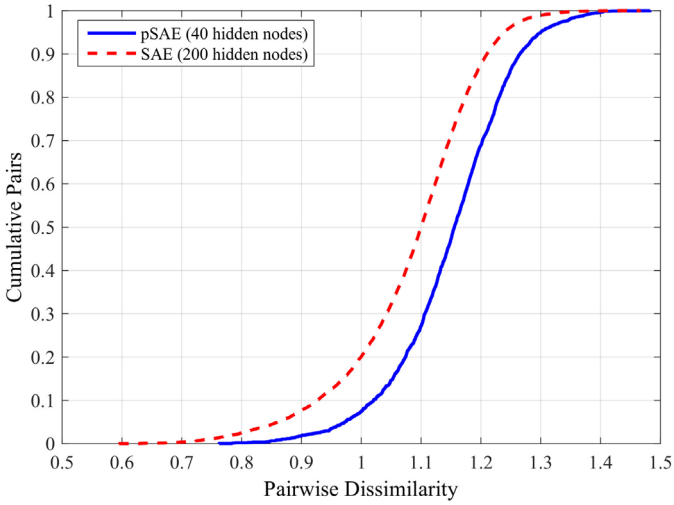

Fig. 3. The minimum pairwise features dissimilarity distribution. (Red dashed line: feature pairs from 10 SAEs with 200 hidden nodes; Blue sold line: feature pairs from 10 SAEs with 40 hidden nodes).

From the figure, it can be seen that when multiple networks with a relatively small number of hidden nodes (40) are trained on the same data, each network learns somewhat more distinctive features than when the trained networks have more hidden nodes (200). This leads to almost as much diversity in the pool of 400 features produced by the pSAEs as in the pool of 2,000 features generated by the SAEs, but with much lower redundancy, thus leading to a more efficient process.

C. *The Feature Selection Algorithm*

Taking advantage of the diverse feature pool generated by the multiple SAEs or pSAEs requires a feature selection mechanism that can choose a sufficient number of maximally diverse features from the pool. A novel algorithm for doing so is described in this section. **Figs. 4** and **5** illustrate the general procedure of the algorithm. First, a set of L SAEs (or pSAEs), each with m hidden units, is trained with the same *small* dataset. A typical value for L may be 10 or 15. To obtain the small training set, T images are selected randomly from the MNIST training dataset, with $T/10$ images per digit. We typically use T values between 1,000 and 10,000 in experiments. The *original feature set*, denoted as **O** with $N=L\times m$ *original features* is obtained using the weight matrix of every hidden neuron in each SAE. Next, the similarity between each pair of features is determined by applying one of three similarity metrics.

One of the similarity metrics is *HOG similarity*, defined based on a HOG feature representation. HOG features are extracted for all features in **O**. As the figure shows, a HOG feature can be visualized as a histogram, in which different directions are represented on the x-axis while the accumulated gradient value for pixels sharing the same direction is indicated on the vertical axis.

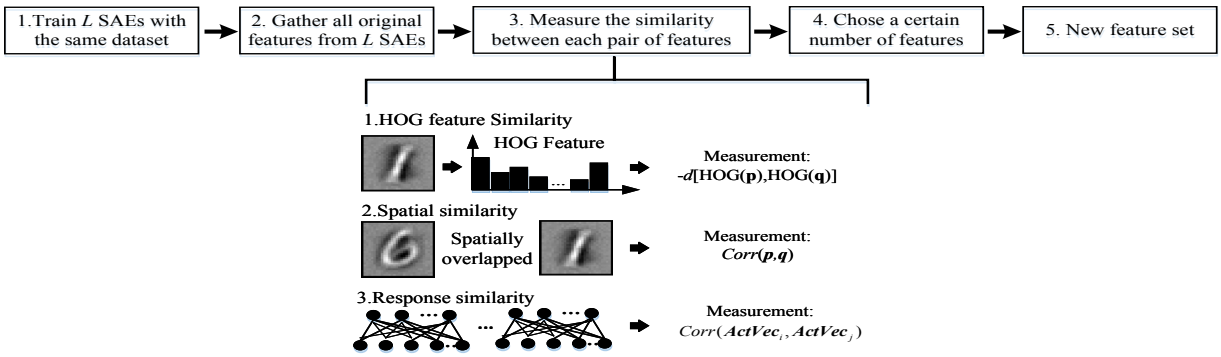

Fig. 4. Steps of the novel feature selection algorithm.

To measure the dissimilarity between two elements in **O**, the Euclidean distance of their HOG features is calculated. More specifically, if ***p, q*** in the figure are any two different features from **O**, HOG(***p***) and HOG(***q***) represent their relative HOG features, and d[HOG(***p***),HOG(***q***)] denotes the Euclidean distance between their HOG features, which quantifies the dissimilarity of two features in **O**. The negative value of dissimilarity, $-d$[HOG(***p***),HOG(***q***)], is used to measure their similarity. HOG features were extracted using scikit-image [30] which is a PYTHON package for image processing.

The second metric is *spatial similarity*. Both features are first vectorized by column and the Pearson correlation coefficients of the two vectors calculated, denoted as $Corr(\boldsymbol{p},\boldsymbol{q})$. Thus, this metric measures how spatially similar the two original features are purely in terms of the spatial pattern (i.e., independent of mean intensity). The third metric is *response similarity*. This measures the degree to which the response of two features (hidden units) is similar across the same input data points. It is obtained by getting the outputs of the two features for all N data points in the same order and calculating the Pearson's correlation coefficient for the two vectors. The set **S** is then defined as the set of all feature pairs labeled with their similarities.

After **S** has been constructed based on one of the three metrics, in Step 3, a feature-picking process is applied to obtain the set **F** of the most distinct features based on **S** and **O**. The proposed algorithm ensures that every feature in **F** is quite different from other features. As **Fig. 5** shows, the elements in **S** are first sorted in ascending order of similarity, and all feature pairs with mutual similarity greater than a threshold eliminated. Then the set **F** is constructed from the sorted **S** as follows. First, the two features in the first element of sorted **S**, which are the least similar elements in **O**, are added to **F**. Next, each feature pair is considered in ascending order of similarity. In each case, the two corresponding features are obtained from **O** and their similarities with any existing feature in **F** calculated**.** If both features are sufficiently dissimilar to every feature in **F** (each similarity is smaller than a predefined threshold), they are added into **F**. Otherwise, the pair is rejected, and all feature pairs with these features are eliminated from the search. This step is repeated until a target number of features has been obtained or no remaining feature pair meets the threshold. This strategy for picking features is termed the *most distinct features* (MDF) *strategy*.

Algorithm: Selection of the most distinguishing features.

Input: The original feature set $\mathbf{O}=\{feature_n\}$, $n=\{1,2,\ldots,N\}$.

Output: The selected features set $\mathbf{F}=\{s\_feature_w\}$, $w=\{1,2,\ldots,W\}$.

Step 1: Calculate K pairwise similarities of feature pairs in **O**. $K=N(N-1)/2$. All similarity pairs form the similarity set $\mathbf{S}=\{s_{ij}^k\}$, $i=\{1,2,\ldots,N\}$, $j=\{1,2,\ldots,N\}$, $i\neq j$, $s_{ij}^k=s_{ji}^k$ and $k=\{1,2,\ldots,K\}$.

Step 2: Sort **S** in ascending order.

Step 3: *count* = 0 ; *count_f* =0; *threshold=a;k* =0; $\mathbf{F}=\{\varnothing\}$;

Delete all pairwise similarities in **S** *with similarity > a;*
Update K = number of remaining pairwise similarities;
Create $\mathbf{R}=\{\varnothing\}$ *= set of rejected features;*
while *count* < *K and count_f* < *W :*
Get i and j from s_{ij}^{count} *in* **S***; Get the* $\mathbf{O}_i$ *and* $\mathbf{O}_j$ *from* **O***;*
if $\mathbf{O}_i$ *or* $\mathbf{O}_j$ *are not in* **F** *or in* **R***:*
for *each element* $\mathbf{F}_k$ *in* **F***:*
Calculate similarities between $\mathbf{F}_k$ *and* $\mathbf{O}_i, \mathbf{O}_j$ *;*
if *both similarities* $\leq a$ *:*
put $\mathbf{O}_i$ *and* $\mathbf{O}_j$ *into* **F***;*
count_f=count_f+2;
end if
else *put* $\mathbf{O}_i$ *and* $\mathbf{O}_j$ *into* **R***;*
end for
end if
count = count +1
end while

Fig. 5. The most distinct features picking strategy.

To explore the relative benefits of different feature picking strategies, another feature selection method has also been devised to serve as a comparison. It is termed the *clustering-based* (CB) *strategy*. In this method, the original features in **O** are clustered into M clusters based on the chosen similarity metric. The agglomerative hierarchical clustering (AHC) algorithm [31] implemented in the scikit-learn machine learning package [32] written in PYHTON was used for the AHC analysis. After obtaining clusters of features, features are drawn in equal proportion from each cluster. Since the clusters represent distinct features, the reduced feature set obtained in this way preserves the diversity of **O**.

IV. EXPERIEMTNS & RESULTS

A. Experiments

For testing the proposed feature selection method, two frameworks were developed, as illustrated in **Fig.6**.

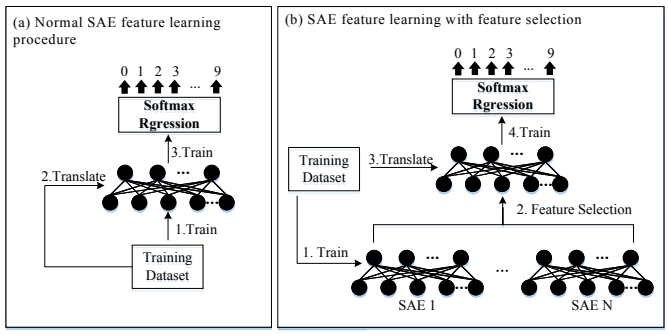

Fig. 6. Two handwritten digit recognition frameworks.

Fig. 6(a) shows a normal SAE-based classifier with softmax classification [33]. The SAE is used for the feature learning. After training, it is able to transform high dimensional images to low dimensional representations. The softmax layer is then trained to classify data in this low-dimensional representation. An SAE with better original features can generate better low-dimensional representation, and a softmax model trained on such a representation has higher recognition accuracy on the test dataset. **Fig. 6(b)** illustrates the proposed framework with the feature selection procedure. L SAEs are trained, the features from their hidden layer are pooled, and a subset of features is selected using one of the algorithms described above. Then a new feature layer is constructed using the selected features. The softmax regression model is then trained on the representations of training data points by these features *without further tuning* the features. In the framework, the SAE has 784 input and output nodes, and is trained by the standard backpropagation algorithm. The softmax model is also trained through gradient descent. The quality of the selected feature set in **Fig. 6(b)** is measured by the recognition accuracy of the trained softmax model on the test dataset. The recognition accuracy of both frameworks is also compared to evaluate the proposed feature selection methods. For setting the parameters of the SAEs, including the sparsity parameter ρ, weight decay parameter λ, and weight control parameter for the sparsity penalty term β appropriately, the approach from Andrew Ng's unsupervised feature learning and deep learning online tutorial [34] is followed. Thus, in the following experiments, parameters for all SAEs were set as follows: $\rho = 10^{-2}$, $\lambda = 10^{-4}$ and β=3. The only parameter of softmax regression model that needs to be chosen is the weight decay term control parameter λ. Following the tutorial, it was set to 10^{-3}.

B. Results

Simulations were performed to evaluate three distinct issues: 1) Which similarity metric works the best for feature selection? 2) How much improvement do the two feature selection methods provide over the equivalent no-selection case? and 3) How does the performance using pSAEs for feature selection compare with that obtained by using SAEs?

To answer the first two questions, SAEs were trained under 30 different scenarios: Using 1000, 3000, 5000, 7000, 9000 and 10000 images respectively for 60, 70, 80, 90, 100 epochs. Ten SAEs, each with 70 hidden units, were run for each scenario. The trained features from each SAE were then used in a softmax classifier as shown in the framework in **Fig. 6(a).** The recognition accuracy averaged over the 10 trials for each scenario is shown in Table 1, and quantifies the performance for that case. This provides the baseline case for answering questions 1 and 2. The SAE size was deliberately chosen to be relatively small so that there was significant room for improvement with feature selection.

Table 1 Performance of the Fig. 6(a) baseline architecture under different training situations.

Mean (standard deviation 10^{-3})	1×10^3 images	3×10^3 images	5×10^3 images	7×10^3 images	9×10^3 images	1×10^4 images
60 epochs	0.7562 (0.2320)	0.7775 (0.2006)	0.7863 (0.4096)	0.7980 (0.2842)	0.8090 (0.7230)	0.8612 (0.4872)
70 epochs	0.7642 (0.2300)	0.7811 (0.1476)	0.7931 (0.0667)	0.8065 (0.2875)	0.8147 (0.2098)	0.8235 (0.3)
80 epochs	0.7679 (0.1603)	0.7882 (0.1579)	0.7985 (0.7269)	0.8110 (0.0871)	0.8211 (0.2986)	0.8278 (1.7)
90 epochs	0.7708 (0.3256)	0.7902 (0.4120)	0.8008 (0.2321)	0.8149 (0.2277)	0.8222 (0.5401)	0.8303 (0.4)
100 epochs	0.7796 (0.3449)	0.7927 (0.5118)	0.8019 (0.2874)	0.8162 (0.4218)	0.8244 (0.1088)	0.8332 (0.35)

Not surprisingly, more training and larger datasets improved performance, but only slightly. In all cases, the standard deviation (SD) of the relative average accuracy was very small, indicating that the results were stable.

Next, the 700 features from the 10 SAEs trained in each scenario were pooled to form the *original feature set* **O** for that scenario. Three new sets of 70 features each were constructed for each scenario by selecting 70 features using the MDF feature selection algorithm (described in **Fig. 5)** with each of the three similarity metrics on the corresponding original features pool **O** of 700 features. The selected features in each of the three cases for each scenario were used with the training data to train the softmax model (without feature tuning). The performance of the resulting classifiers was evaluated on the test data set. The results are summarized in **Fig. 7**, all of them generated based on the framework in **Fig. 6(b).** The figure includes three tables, each of which shows the performance difference for each scenario compared to **Table 1**. Different colors indicate the polarity of the difference. Blue means that the performance with feature selection was better than the baseline shown in **Table 1** while red indicates lower performance. Color intensity indicates the degree of the difference. It is clear that the most significant accuracy improvement occurs in **Fig. 7(a),** which is based on HOG similarity, with accuracy increasing by as much as 5.89%. Moreover, *each* result in **Fig. 7(a)** is better than the average training accuracy in **Table 1** under the same training scenario, indicating that the improvement is *generic* and not due to some quirk of parameter setting. Importantly, the five most significant improvements, which are around 4-6%, happened when both the training data and the number of training epochs were small, which is very promising. Results for spatial similarity (**Fig. 7(b)**) were mixed, and showed minimal benefit at best, while those for response similarity (**Fig.7(c)**) were

poor, being significantly worse than baseline. This shows that: a) Feature selection does provide significant benefit; and b) HOG similarity is the best of the three metrics to use for feature selection.

To compare the MDF and CB feature selection strategies, the process described above was repeated for CB, using only HOG similarity. Each training scenario was tried for 2 to 5 clusters obtained with Ward linkage, minimizing the sum of squared differences within all clusters, which forces features in one cluster to be as similar as possible. In this experiment, results were best when the number of cluster was 3. **Fig. 8** compares the performance of the two feature picking strategies for this case, showing that MDF provided significantly greater improvement over baseline than CB, though the latter also provided a small but systematic improvement.

(a) HOG Similarity metric

No. of epochs	No. of Images 10^3	3×10^3	5×10^3	7×10^3	9×10^3	10^4
60	2.40%	3.98%	2.69%	2.78%	2.23%	2.45%
70	1.72%	5.89%	4.47%	2.43%	2.07%	2.06%
80	2.10%	4.02%	4.41%	2.53%	2.01%	2.04%
90	2.03%	2.79%	2.42%	2.11%	1.83%	1.90%
100	1.64%	1.64%	1.64%	1.64%	1.64%	1.64%

Highest / Middle / Lowest

(b) Spatial Similarity metric

No. of epochs	No. of Images 10^3	3×10^3	5×10^3	7×10^3	9×10^3	10^4
60	-0.56%	2.91%	1.19%	1.26%	1.11%	1.64%
70	0.62%	2.12%	1.22%	1.18%	0.79%	1.19%
80	-0.51%	1.82%	0.11%	1.25%	0.98%	0.93%
90	-0.12%	1.76%	1.02%	1.12%	0.96%	0.92%
100	0.14%	1.16%	0.83%	0.71%	0.62%	-0.39%

(c) Temporal Similarity metric

No. of epochs	No. of Images 10^3	3×10^3	5×10^3	7×10^3	9×10^3	10^4
60	-1.20%	-0.86%	-0.46%	-0.31%	-0.20%	-0.25%
70	-0.06%	-0.11%	-0.12%	-0.18%	-0.19%	-0.12%
80	-0.14%	0.12%	0.47%	-0.29%	-0.15%	-0.16%
90	0.99%	0.89%	0.98%	1.23%	0.99%	0.89%
100	0.14%	0.80%	0.64%	0.64%	0.64%	0.64%

Fig. 7. Comparison of performance change with feature selection using the three feature similarity metrics. (a) HOG similarity metric. (b) Spatial similarity metric. (c) Temporal similarity metric.

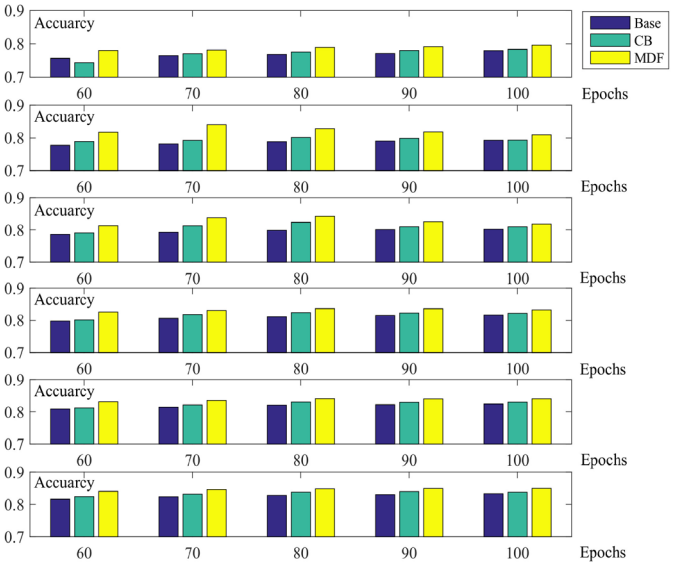

Fig. 8. Comparison of the MDF and CB feature picking strategies with the baseline.

Finally, to check whether using pinched SAEs to generate the feature pool actually helped improve the performance of feature selection, four cases were compared. In the *SAE-Base case*, fifteen networks of the type shown in **Fig. 6(a)**, each with a 200 feature SAE, were trained using only 3,000 training samples for 100 epochs. The average performance of these on each of the ten digits was determined using the test set. In the *SAE-MDF case*, the 3,000 features learned by the 15 networks in the SAE-Base case were pooled and 200 features selected from this pool using the MDF algorithm with HOG similarity. These features were then used without further tuning as input for training a softmax classifier as shown in **Fig. 6(b)**, and the mean performance of this classifier was calculated for each digit. The *pSAE-MDF case* repeated the procedure of the SAE-MDF case, but generated a pool of 600 features from fifteen 40-feature pSAEs. The final classifier used 200 features selected from this pool. Finally, the *pSAE-Base case* repeated the SAE-Base case, but with fifteen 40-feature pSAEs used with the architecture in **Fig. 6(a)**. All other parameter settings between all cases were kept the same. The results are shown in **Fig. 9**.

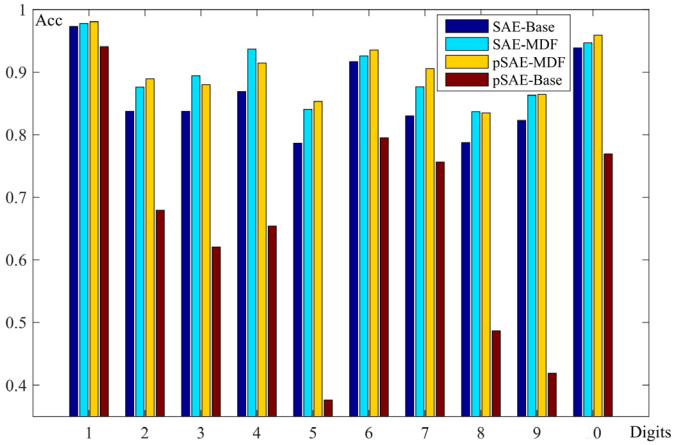

Fig. 9. The effect of MDF feature selection with SAEs and pSAEs.

As the figure shows, SAE-MDF outperforms SAE-Base significantly for almost every digit, verifying the clear benefit of MDF feature selection. Even more significantly, the performance of pSAE-MDF is comparable with – and, on some digits, better than – the SAE-MDF performance. The latter result is important not only because it demonstrates the utility of using pSAEs, but especially because pSAEs are significantly cheaper to use from a computational perspective due to their smaller size. It is also interesting to compare the results of the pSAE-Base and pSAE-MDF cases. Though the individual pSAE networks gave very poor performance, selecting 200 features from them using MDF increased performance to far above what the much larger SAE-Base networks could produce, and to the same level as the more expensive SAE-MDF case. It is also worth noting that all these trials used only a small fraction of the available data (3,000 points out of 60,000) for training, thus validating the procedure for limited data situations.

V. Conclusion

In this paper, a novel feature selection method for feature learning based on multiple SAEs has been proposed and examined on a handwritten digits recognition task. The results indicate that: 1) Feature selection by pooling features from multiple SAEs and applying the MDF procedure leads to better classification performance; 2) The best similarity metric to use in this case is HOG similarity; and 3) Using the feature selection procedure with pSAEs instead of larger and more computationally expensive SAEs still provides the same performance boost, *even though the individual pSAEs have much poorer performance*.

The next step in this investigation is to study how far the size of the pSAEs can be pushed down, and whether using even less data with fewer training epochs in generating the pool might be sufficient. Clearly, the answers to these questions will depend on the classification task. Preliminary experiments show that, for the MNIST digits recognition task, pSAEs down to 30 nodes still provide adequate results with feature selection, but smaller ones do not. Results on this and other issues, as well as the application of the method to more complex datasets, will be reported in the future.

References

[1] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological cybernetics,* vol. 59, pp. 291-294, 1988.

[2] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length, and Helmholtz free energy," *Advances in neural information processing systems,* pp. 3-3, 1994.

[3] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks,* vol. 2, pp. 359-366, 1989.

[4] Y. Bengio, L. Yao, G. Alain, and P. Vincent, "Generalized denoising auto-encoders as generative models," in *Advances in Neural Information Processing Systems*, 2013, pp. 899-907.

[5] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural Networks, 1989. IJCNN., International Joint Conference on*, 1989, pp. 593-605.

[6] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," 1974.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling,* vol. 5, p. 1, 1988.

[8] P. Vincent, "A connection between score matching and denoising autoencoders," *Neural computation,* vol. 23, pp. 1661-1674, 2011.

[9] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *The Journal of Machine Learning Research,* vol. 11, pp. 3371-3408, 2010.

[10] G. Alain and Y. Bengio, "What regularized auto-encoders learn from the data-generating distribution," *The Journal of Machine Learning Research,* vol. 15, pp. 3563-3593, 2014.

[11] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 833-840.

[12] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science,* vol. 313, pp. 504-507, 2006.

[13] J. Deng, Z. Zhang, E. Marchi, and B. Schuller, "Sparse autoencoder-based feature transfer learning for speech emotion recognition," in *Affective Computing and Intelligent Interaction (ACII), 2013 Humaine Association Conference on*, 2013, pp. 511-516.

[14] M. D. Zeiler, M. A. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le*, et al.*, "On rectified linear units for speech processing," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, 2013, pp. 3517-3521.

[15] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt, "Spatio-Temporal Convolutional Sparse Auto-Encoder for Sequence Classification," in *BMVC*, 2012, pp. 1-12.

[16] H.-C. Shin, M. R. Orton, D. J. Collins, S. J. Doran, and M. O. Leach, "Stacked autoencoders for unsupervised feature learning and multiple organ detection in a pilot study using 4D patient data," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 35, pp. 1930-1943, 2013.

[17] L. Zhao, Q. Hu, and W. Wang, "Heterogeneous Feature Selection with Multi-Modal Deep Neural Networks and Sparse Group Lasso," *IEEE Transactions on Multimedia,* vol. 17, pp. 1936-1948, 2015.

[18] Z. Zhu, P. Luo, X. Wang, and X. Tang, "Deep learning multi-view representation for face recognition," *arXiv preprint arXiv:1406.6947,* 2014.

[19] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," *The Journal of Machine Learning Research,* vol. 10, pp. 1-40, 2009.

[20] H.-I. Suk, S.-W. Lee, D. Shen, and A. s. D. N. Initiative, "Latent feature representation with stacked auto-encoder for AD/MCI diagnosis," *Brain Structure and Function,* pp. 1-19, 2013.

[21] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems,* vol. 19, p. 153, 2007.

[22] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 689-696.

[23] H. Lee, C. Ekanadham, and A. Y. Ng, "Sparse deep belief net model for visual area V2," in *Advances in neural information processing systems*, 2008, pp. 873-880.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, pp. 2278-2324, 1998.

[25] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, "Deep learning via semi-supervised embedding," in *Neural Networks: Tricks of the Trade*, ed: Springer, 2012, pp. 639-655.

[26] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?," *The Journal of Machine Learning Research,* vol. 11, pp. 625-660, 2010.

[27] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1096-1103.

[28] K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," in *Computer Vision, 2009 IEEE 12th International Conference on*, 2009, pp. 2146-2153.

[29] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Convolutional neural network committees for handwritten character classification," in *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, 2011, pp. 1135-1139.

[30] S. Van Der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager*, et al.*, "scikit-image: image processing in Python," *PeerJ,* vol. 2, p. e453, 2014.

[31] F. Murtagh, "A survey of recent advances in hierarchical clustering algorithms," *The Computer Journal,* vol. 26, pp. 354-359, 1983.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel*, et al.*, "Scikit-learn: Machine learning in Python," *The Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097-1105.

[34] A. Ng, J. Ngiam, C. Y. Foo, Y. Mai, and C. Suen, "UFLDL tutorial," ed, 2012.