NEIGHBOR-ENCODER

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a novel unsupervised representation learning framework called *neighbor-encoder* in which domain knowledge can be trivially incorporated into the learning process without modifying the general encoder-decoder architecture. In contrast to autoencoder, which reconstructs the input *data*, neighbor-encoder reconstructs the input data's *neighbors*. The proposed neighbor-encoder can be considered as a generalization of autoencoder as the input data can be treated as the nearest neighbor of itself with zero distance. By reformulating the representation learning problem as a neighbor reconstruction problem, domain knowledge can be easily supplied with appropriate definition of similarity or distance between objects. As such, any existing similarity search algorithms can be easily integrated into our framework. Applications of other algorithms (e.g., association rule mining) in our framework is also possible since the concept of "neighbor" is an abstraction which can be appropriately defined differently in different contexts. We have demonstrated the effectiveness of our framework in various domains, including images, time series, music, etc., with various neighbor definitions. Experimental results show that neighbor-encoder outperforms autoencoder in all scenarios we considered.

1 INTRODUCTION

Unsupervised representation learning has been shown effective in tasks such as dimension reduction, clustering, visualization, information retrieval, and semi-supervised learning (Goodfellow et al., 2016; Yang et al., 2017). While domain-specific unsupervised representation learning methods like word2vec (Mikolov et al., 2013a;b) and video-based representation learning (Agrawal et al., 2015; Jayaraman & Grauman, 2015; Wang & Gupta, 2015; Pathak et al., 2017) have been widely adopted in their respective domains, their success cannot be directly transferred to other domains as their assumptions do not hold for other types of data. In contrast, general unsupervised representation learning methods such as autoencoder (Bengio et al., 2007; Huang et al., 2007; Vincent et al., 2008; 2010) can be effortlessly applied to data from various domains, but the performance of general methods is usually inferior to those that utilizes domain knowledge (Mikolov et al., 2013a;b; Agrawal et al., 2015; Jayaraman & Grauman, 2015; Wang & Gupta, 2015; Pathak et al., 2017).

In this work, we propose an unsupervised representation learning framework (i.e., neighbor-encoder) which is *general* as it can be applied to various types of data and *versatile* since domain knowledge can be trivially added by adopting various "off-the-shelf" data mining algorithms for finding neighbors. Figure 1 previews the t-Distributed Stochastic Neighbor Embedding (t-SNE) (Maaten & Hinton, 2008) visualization produced from a human physical activity data set (see Section 4.2 for details). The embedding is generated by projecting representation learned by neighbor-encoder, representation learned by autoencoder, and raw data to $2D$. By using a suitable neighbor finding algorithm, the representation learned by neighbor-encoder provides a more meaningful visualization than its rival methods.

In summary, our major contributions include:

- We propose a general and versatile framework, the neighbor-encoder, which can be used to trivially combine a large family of similarity search techniques with unsupervised representation learning to incorporate domain knowledge.
- We demonstrate the superior performance of the representations learned by neighbor-encoder, compared to representations learned by autoencoder in handwritten digit data,

human physical activity data, and instrumental sound data for various machine learning tasks including classification, clustering, and visualization.

- We demonstrate that the neighbor-encoder framework can considerably outperform autoencoder with a proper neighbor definition.

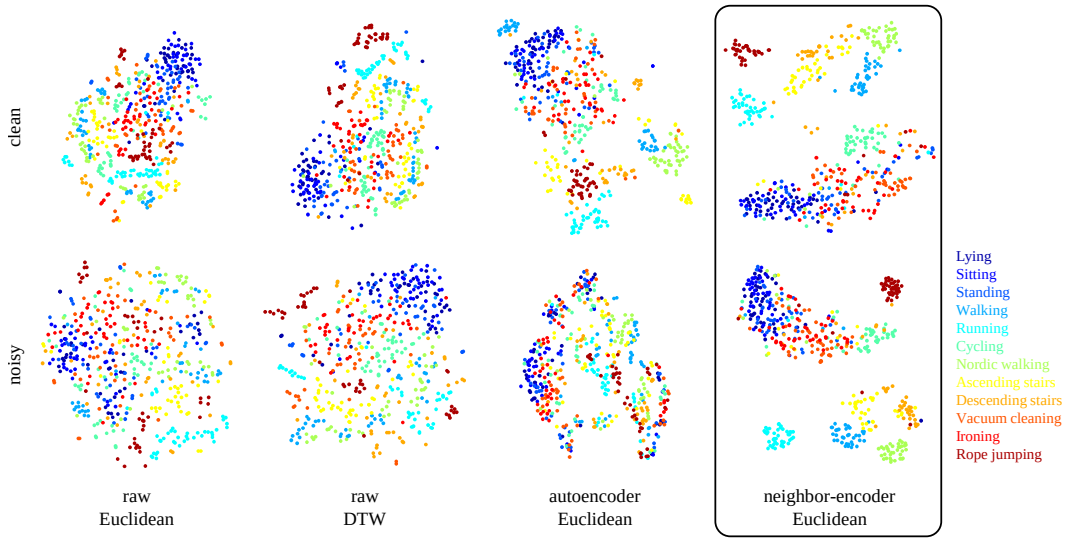


Figure 1: Visualizing the learned representation versus the raw time series on PAMAP2 (human physical activity) data set (Reiss & Stricker, 2012a;b) using t-SNE (Maaten & Hinton, 2008) with either Euclidean or dynamic time warping (DTW) distance (Nguyen et al., 2017). If we manually select 27 dimensions of the time series that are *clean* and relevant (acceleration, gyroscope, magnetometer, etc.), the representation learned by both autoencoder and neighbor-encoder achieves better class separation than raw data. However, if the data include *noisy* and/or irrelevant dimensions (heart rate, temperature, etc.), neighbor-encoder outperforms autoencoder noticeably.

2 Related Work

Unsupervised representation learning is usually achieved by optimizing either domain specific objectives or general unsupervised objectives. For example, in the domain of computer vision and music processing, unsupervised representation learning can be formulated as a supervised learning problem with surrogate labels generated by exploiting the temporal coherence in videos and music (Agrawal et al., 2015; Jayaraman & Grauman, 2015; Wang & Gupta, 2015; Pathak et al., 2017; Huang et al., 2017); in the case of natural language processing, word embedding can be obtained by optimizing an objective function that "pushes" words occurring in a similar context (i.e., surrounded by similar words) closer in the embedding space (Mikolov et al., 2013a;b). Alternatively, general unsupervised objectives are also useful for unsupervised representation learning. For example, both autoencoder (Bengio et al., 2007; Huang et al., 2007; Vincent et al., 2008; 2010) and dictionary learning (Mairal et al., 2009) are based on minimizing the self-reconstruction error, while optimizing the k-means objective is shown effective in Coates & Ng (2012) and Yang et al. (2017). Other objectives, such as self-organizing map criteria (Kohonen, 1982; Bojanowski & Joulin, 2017) and adversarial training (Goodfellow et al., 2014; Donahue et al., 2016; Radford et al., 2015) are also demonstrated as effective objectives for unsupervised representation learning.

Autoencoder is a decade-old unsupervised learning framework for dimension reduction, representation learning, and deep hierarchical model pre-training; many variants have been proposed since its initial introduction (Bengio et al., 2007; Goodfellow et al., 2016). For example, the denoising autoencoder reconstructs the input data from its corrupted version; such modification improves the robustness of the learned representation (Vincent et al., 2010). Variational autoencoder (VAE) regularizes the learning process by imposing a standard normal prior over the latent variable (i.e., representation), and such constraints help the autoencoder learn a valid generative model (Kingma

& Welling, 2013; Rezende et al., 2014). Makhzani et al. (2015) and Larsen et al. (2015) further improved generative model learning by combining VAE with adversarial training. Sparsity constraints on the learned representation are another form of regularization for autoencoder to learn a more discriminating representation for classification, both the k-sparse autoencoder (Makhzani & Frey, 2013; 2015) and k-competitive autoencoder (Chen & Zaki, 2017) incorporate such ideas.

3 NEIGHBOR-ENCODER FRAMEWORK

In this section, the proposed neighbor-encoder framework is introduced and compared with autoencoder. Figure 2 shows different encoder-decoder configurations for both neighbor-encoder and autoencoder. In the following sections, we will discuss the motivation and design of each encoder-decoder configuration in detail.

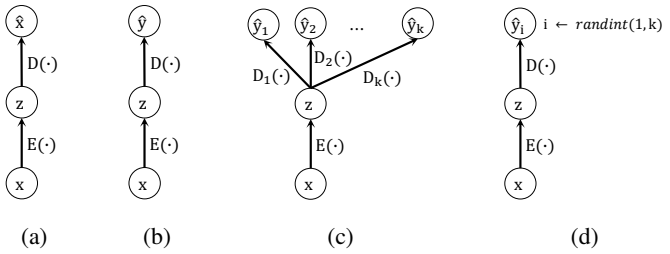


Figure 2: Various encoder-decoder configurations for training autoencoder and neighbor-encoder: a) autoencoder, b) neighbor-encoder, c) k-neighbor-encoder with k decoder, and d) k-neighbor-encoder with 1 decoder.

3.1 AUTOENCODER

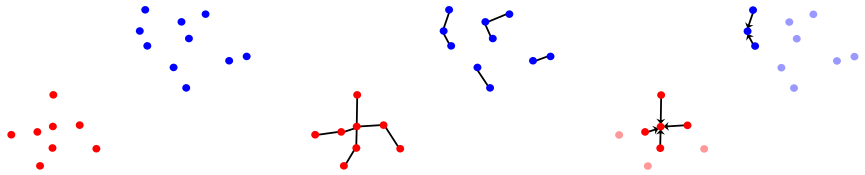
The overall architecture of autoencoder consists of two components: an *encoder* and a *decoder*. Given input data x, the encoder $E(\cdot)$ is a function that encodes x into a latent representation z (usually in a lower dimensional space), and the decoder $D(\cdot)$ is a function that decodes z in order to reconstruct x. Figure 2a shows the feed-forward path of an autoencoder where $z = E(x)$ and $\hat{x} = D(z)$. We train the autoencoder by minimizing the difference between the input data x and the reconstructed data $\hat{x}$. Formally, given a set of training data X, the parameters in $E(\cdot)$ and $D(\cdot)$ are learned by minimizing the objective function $\sum_{x \in X} loss(x, \hat{x})$ where $\hat{x} = D(E(x))$. The particular loss function we used in this work is mean square error. Other loss function like mean absolute error can also work, but mean square error is picked for its popularity. Once the autoencoder is learned, any given data can be projected to the latent representation space with $E(\cdot)$. Both the encoder and the decoder can adopt any existing neural network architecture such as multilayer perceptron (Bengio et al., 2007), convolutional net (Huang et al., 2007), or long short-term memory (Hochreiter & Schmidhuber, 1997; Srivastava et al., 2015).

3.2 NEIGHBOR-ENCODER

Similar to the autoencoder, neighbor-encoder also consists of an encoder and a decoder. Both the encoder and the decoder in neighbor-encoder work similarly as their counterpart in autoencoder; the major difference is in the objective function. Given input data x, the encoder $E(\cdot)$ is a function that encodes x into a latent representation z, and the decoder $D(\cdot)$ is a function that reconstructs x's neighbor y by decoding z. Figure 2b shows the feed-forward path of a neighbor-encoder where $z = E(x)$ and $\hat{y} = D(z)$. Formally, given a set of training data X and a set of their associated neighbors Y, the neighbor-encoder is learned by minimizing the objective function $\sum_{x,y \in X,Y} loss(y, \hat{y})$, where $\hat{y} = D(E(x))$. Note that here "neighbor" can be defined in a variety of ways. We will introduce examples of different neighbor definitions later in Section 3.4.

We argue that neighbor-encoder can better retain the similarity between data samples in the latent representation space comparing to autoencoder. Figure 3 builds an intuition for this. As shown in Figure 3a, we assume the data set in interest consists of samples from two classes (i.e., blue

class and red class, and each class forms a cluster) in $2D$ space. Since the autoencoder is trained by mapping each data point to itself, the learned representation for this data set would most likely be a rotated and/or re-scaled version of Figure 3a. In contrast, the neighbor-encoder (trained with nearest neighbor relation shown in Figure 3b) would learn a representation with much less intra-class variation. As Figure 3c shows, when several similar data points share the same nearest neighbor, the objective function will force the network to generate exactly the same output for these similar data points, thus forcing their latent representation (which is the input of the decoder) to be very similar.



(a) (b) (c)

Figure 3: Intuition behind neighbor-encoder comparing to autoencoder. a) A simple $2D$ data set with two classes, b) the nearest neighbor graph constructed for the data set (arrowheads of the edges are removed for clarity), and c) an example of how neighbor-encoder would generate representation with smaller intra-class variation for highlighted data points. The neighbor-encoder learns similar representation for closely located data points by forcing these data points to reconstruct the same data point as these data points are most likely sharing the same nearest neighbor (shown by the arrows).

Since we are using neighbor finding algorithms to guide the representation learning process, one may argue that we could instead construct a graph using the neighbor finding algorithm, then apply various graph-based representation learning methods like the ones proposed in Perozzi et al. (2014), Tang et al. (2015), Grover & Leskovec (2016), Dong et al. (2017) or Ribeiro et al. (2017). Graph-based methods are indeed valid alternatives to neighbor-encoder; however, they have the following two limitations: 1) If one wishes to encode a newly obtained data, the out-of-sample problem would bring about additional complexity as these methods are not designed to handle such scenario. 2) It will be impossible to learn a generative model, as graph-based methods learn the representation by modeling the relationship between examples in a data set rather than modeling the example itself. As a result, the proposed neighbor-encoder is preferred over the graph-based methods when the above limitations are crucial.

3.3 k-NEIGHBOR-ENCODER

Similar to the idea of generalizing 1-nearest neighbor classifier to k-nearest neighbor classifier, neighbor-encoder can also be extended to k-neighbor-encoder by reconstructing k neighbors of the input data. This generalization can be achieved with two possible configurations: we can either train k decoders (see Figure 2c) which simultaneously reconstruct all k neighbors of the input, or train only one decoder (see Figure 2d) which reconstructs one randomly sampled neighbor[1] (out of the k neighbors) of the input. We call the former configuration as k-neighbor-encoder with k decoder, and the latter one as k-neighbor-encoder with 1 decoder. Given input data x, the encoder $E(\cdot)$ is a function that encodes x into a latent representation z. In the k decoder case, we have a set of k decoders $[D_i(\cdot)|\forall i \in \mathbb{Z} : 0 < i \leq k]$, in which each individual function $D_i(\cdot)$ decodes z in order to reconstruct x's ith neighbor y_i. In the 1 decoder case, the decoder $D(\cdot)$ is a function that decodes z in order to reconstruct one of x's randomly selected neighbor y_i where $i \leftarrow randint(1, k)$.

The learning process of k-neighbor encoder is slightly more complicated than the neighbor-encoder (i.e., 1-neighbor-encoder). Given a set of training data X and their associated neighbors Y (each $y \in Y$ is a set of k neighbors $[y_i|\forall i \in \mathbb{Z} : 0 < i \leq k]$ of the corresponding data $x \in X$), for the k decoder variant, the k-neighbor-encoder can be learned by minimizing $\sum_{x,y \in X,Y} \sum_{y_i \in y} loss(y_i, \hat{y}_i)$ where $\hat{y}_i = D_j(E(x))$ and $0 < j \leq k$. Note that since there are k decoders, we need to assign each

[1]The sampling process is performed at each iteration of stochastic gradient descent/SGD (or other variants of SGD); therefore, if a training data is sampled at two or more iterations, the data could have different reconstruction target (i.e., neighbor) each time it is sampled.

$y_i \in y$ to a decoders. One possible decoder assignment strategy is simply choosing the decoder that provides the lowest reconstruction loss for each $y_i \in y$. This decoder assignment strategy would work if each training example has roughly k "modes" of neighbors. The 1 decoder variant do not have the decoder assignment problem as there is only one decoder. The k-neighbor-encoder in this case can be learned by simply minimizing $\sum_{x,y \in X,Y} loss(y_i, \hat{y})$ where $i \leftarrow randint(1, k)$ and $\hat{y} = D(E(x))$.

Based on our current experiment result, the k-neighbor-encoder with 1-decoder either outperform or is comparable to the k-decoder configuration; therefore, we recommend using the k-neighbor-encoder with 1-decoder configuration to train neighbor-encoder. However, since the conclusion is drawn from only three sets of experiments, there may still be data sets where the k-decoder configuration outperforms the 1-decoder configuration; thus, we cannot rule out the possibility of using the k-decoder configuration.

3.4 Neighborhood Function

To use any of the introduced neighbor-encoder configurations, we need to properly define the term neighbor. In this section, we discuss several possible neighborhood functions for the neighbor-encoder framework. Note that the functions listed in this section is just a small subset of all the available functions.

- **Simple Neighbor** is defined as the several few objects that are closest to a given object in Euclidean distance or other distances, assuming the distance between every two objects is computable. For example, given a set of objects $[x_1, x_2, x_3, ..., x_n]$ where each object is a real-value vector, the neighboring relationship among the objects under Euclidean distance can be approximately identified by construing a k-d tree.
- **Feature Space Neighbor** is very similar to *simple neighbor*, except that instead of computing the distance between objects in the space where the reconstruction is performed (e.g., the raw-data space), we compute the distance in an alternative representation or feature space. To give a more concrete example, suppose we have a set of objects $[x_1, x_2, x_3, ..., x_n]$ where each object is an audio clip in mel-frequency spectrum space. Instead of finding neighbors directly in the mel-frequency spectrum space, we transform the data into the Mel-frequency Cepstral Coefficient (MFCC) space as neighbors discovered in MFCC space is semantically more meaningful and searching in MFCC space is more efficient.
- **Spatial or Temporal Neighbor** defines the neighbor based on the spatial or temporal closeness of objects. Specifically, given a set of objects $[x_1, x_2, x_3, ..., x_n]$ where the subscript denotes the temporal (or spatial) arrival order, x_i and x_j are neighbors when $|i - j| < d$ where d is a predefined size of the neighborhood. The skip-gram model in word2vec (Mikolov et al., 2013a;b) is an example of spatial neighbor-encoder, as the skip-gram model can be regarded as reconstructing the spatial neighbors (in a form of one-hot vector) of a given word.
- **Time Series Subspace Neighbor** is only defined for multidimensional time series data as the similarity between two objects is measured by only a subset of all dimensions. By ignoring some dimensions, a time series could find neighbors with higher quality since it is very likely that some of the dimension contains irreverent or noisy information (i.e., room temperature in human physical activity data) (Yeh et al., 2017). Given a multidimensional time series, we can use mSTAMP (Yeh et al., 2017) to evaluate the neighboring relationship between all the subsequences within the time series.

4 Experiment

In this section, we show the effectiveness and versatility of neighbor-encoder compared to autoencoder by performing experiments on handwritten digits, human physical activities, and instrumental sounds data with different neighborhood functions. As the neighbor-encoder framework is a generalization of autoencoder, all the variants of autoencoder (e.g., denoising autoencoder (Vincent et al., 2010), variational autoencoder (Kingma & Welling, 2013; Rezende et al., 2014), k-sparse autoencoder (Makhzani & Frey, 2013; 2015), or adversarial autoencoder (Makhzani et al., 2015)) can be

directly ported to the neighbor-encoder framework. As a result, here we did not exhaustively test all variants of autoencoder, but only tested the most original definition of it. We leave the exhaustive comparison of the other variants for future work.

4.1 HANDWRITTEN DIGITS

The MNIST database is commonly used in the initial study of newly proposed methods due to its simplicity (LeCun et al., 1998). It contains $70,000$ images of handwritten digits (one digit per image); $10,000$ of these images are test data and the other $60,000$ are training data. The original task for the data set is multi-class classification. Since the proposed method is not a classifier but a representation learner (i.e., an encoder), we have evaluated our method using the following procedural: 1) we train the encoder with training data, 2) we encode both training data and test data into the learned representation space, 3) we train a simple classifier (i.e., linear support vector machine/SVM) with the representation of training data, then apply the classifier to the representation of test data and report the classification error, and 4) we also apply a clustering method (i.e., k-means) to the representation of test data and report the adjusted Rand index. As a proof of concept, we didn't put much effort in optimizing the structure of the encoder/encoder. We simply used a 4-layer $2D$ convolutional net as the encoder and a 4-layer transposed $2D$ convolutional net as the decoder. The detailed setting of the network architecture is summarized in Figure 4. We have tried several other convolutional net architectures as well; we draw the same conclusion from the experimental results with these alternative architectures.

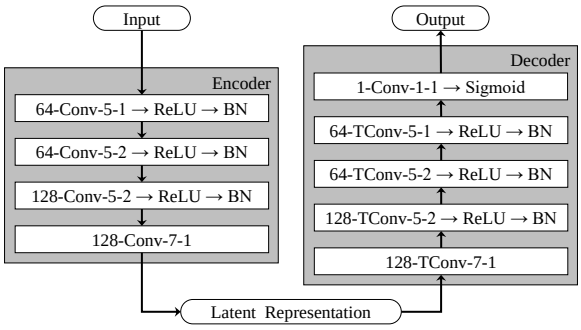
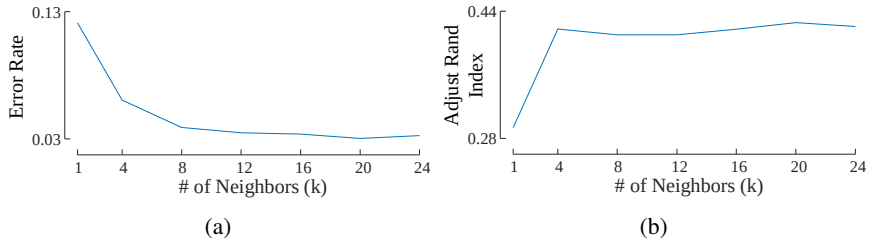


Figure 4: Network architecture for the decoder and decoder. 64-Conv-5-1 denotes $2D$ convolutional layer with 64 5×5 kernels and stride of 1. ReLU denotes rectified linear unit. BN denotes batch normalization. TConv denotes transposed $2D$ convolutional layer.

Here we use the k-neighbor-encoder with 1 decoder configuration (shown in Figure 2d) for our neighbor-encoder. Note that in this set of experiments, we assume that the nearest neighbor of an image is itself. In other words, when $k = 1$, the encoder-decoder network is an autoencoder, and when $k > 1$, the encoder-decoder network is a k-neighbor-encoder with 1 decoder where the reconstruction target is randomly selected from the k nearest neighbors of the input image at training time. Figure 5 shows the classification and clustering results for different setting of k. We can see that neighbor-encoder considerably outperforms autoencoder in both classification and clustering. Similar conclusions can be drawn from another set of experiments which we performed using a variant of the current neighbor definition on the same data set (see Appendix A for details).

To explain the performance difference between autoencoder and neighbor-encoder, we randomly selected 5 test examples from each class (see Figure 6a) and fed them through both the autoencoder and the neighbor-encoder ($k = 24$) trained in the previous experiment. The outputs are shown in Figure 6b (autoencoder) and Figure 6c (neighbor-encoder) respectively. As expected, the output of autoencoder is almost identical to the input image. In contrast, although the output of neighbor-encoder is still very similar to the input image, the intra-class variation is noticeably less than the output of autoencoder. This is because neighbor-encoder tends to reconstruct the same neighbor image from similar input data points (recall Figure 3c). As a result, the latent representation learned by neighbor-encoder achieves better classification/clustering performance.



(a) (b)

Figure 5: a) Classification error rate and b) clustering adjusted Rand index as we vary the number of neighbors for neighbor encoder. The $k = 1$ case is autoencoder.

(a) (b) (c)

Figure 6: a) the input to the encoder-decoder, b) the output of autoencoder, and c) the output of neighbor-encoder.

4.2 Human Physical Activities

In Section 3, we have introduced various configurations for neighbor-encoder. To evaluate the effectiveness of these configurations, here we test them on the PAMAP2 data set (Reiss & Stricker, 2012a;b) using the time series subspace neighbor definition (Yeh et al., 2017). We choose the subspace neighbor definition because 1) it addresses one of the commonly seen multidimensional time series problem scenarios (i.e., the existence of irrelevant/noisy dimensions), and 2) it is able to extract meaningful repeating patterns (Yeh et al., 2017).

The PAMAP2 data set was collected by mounting 3 inertial measurement units and a heart rate monitor on 9 subjects while they were performing 18 different physical activities (e.g., walking, running, playing soccer) during 9 recording sessions ranging from 0.5 hours to 1.9 hours (i.e., 1 session per subject). The subjects performed one activity for few minutes, took a short break, then continued performing another activity. In order to transfer the data set into a format that we can use for evaluation (i.e., a training/test split), for each subject (or recording session), we cut the data into segments according to their corresponding physical activities; then, within each activity segment, we generate training data from the first half and test data from the second half with a sliding window of length of 100 and step size of 1. We make sure that there is no overlap between training data and test data. After the reorganization, we end up with 9 data sets (i.e., 1 pair of training/test set per subject). We ran experiments on each data set independently and report averaged performance results.

The experiment procedural is very similar to the one presented in Section 4.1. We first train the encoder-decoder network with the training data, then measure the performance of the learned latent representation in classification (linear SVM) and clustering (k-means). Additionally, here we also evaluate the visualization capability of the learned latent representation. The visualization capability is measured with R-precision (Manning et al., 2008) by running an information retrieval experiment in visualization space (i.e., $2D$ space).

The encoder-decoder network architecture we used is summarized in Figure 7. Here we use a 5-layer $1D$ convolutional net as the encoder and a 5-layer $1D$ convolutional net as the decoder. Similar to in Section 4.1, we did not put much effort in optimizing the structure of this network architecture. We have tried few other convolutional net architectures, such as adding batch normalization, modify the number of layers, or modifying the number of filters for each layer, and the conclusion from

the experimental results remains virtually unchanged. During test time, we apply global averaging pooling to the output of the encoder to obtain the latent representation.

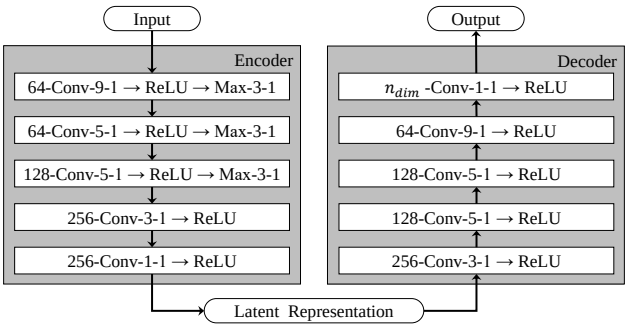


Figure 7: Network architecture for the encoder and the decoder. 64-Conv-9-1 denotes $1D$ convolutional layer with 64 sized 9 kernels and sized 1 stride. ReLU denotes rectified linear unit. Max-3-1 denotes max pooling layer with sized 3 pooling window and sized 1 stride. n_{dim} is the number of dimension for the input multidimensional time series.

In Table 1, we compare the classification/clustering/visualization capability results obtained in four different representation spaces, including the results in the raw data space, the results of the latent representation learned by three different encoder-decoder configurations (i.e., autoencoder, k-neighbor-encoder with k decoder and k-neighbor-encoder with 1 decoder), and the results of the latent representation learned by supervised method. The structure of the supervised network is the same as the encoder part in Figure 7, and a fully connected layer is used for label prediction which replaces the decoder. We further compare the results under two different scenarios: "clean" and "noisy". In the "clean" scenario, we manually deleted some dimensions of the data that are irrelevant (or harmful) to the classification/clustering tasks, while in the "noisy" scenario, all dimensions of the data are retained.

Table 1: Experimental results on the PAMAP2 data sets. The reported values are averaged across all subjects. The performance for classification (linear SVM) was measured with accuracy, the performance for clustering (k-means) was measured with adjusted Rand index, and the performance for visualization (PCA and t-SNE) was measured with R-precision. The best method for each task is indicated with bold font.

	task	raw	auto	k-neighbor k-decoder	k-neighbor 1-decoder	Supervised
clean	linear SVM	0.4290	0.6343	0.6358	**0.6438**	0.7738
	k-means	0.1284	0.2184	0.2175	**0.2282**	0.2728
	PCA	0.1883	0.3059	0.3045	**0.3194**	0.4239
	t-SNE	0.2801	0.3472	0.3517	**0.3937**	0.5729
noisy	linear SVM	0.4188	0.4860	0.6060	**0.6158**	0.7653
	k-means	0.0780	0.0973	0.2203	**0.2327**	0.2878
	PCA	0.1651	0.1785	0.3050	**0.3330**	0.4081
	t-SNE	0.2091	0.2057	0.3451	**0.3749**	0.5689

Overall, the learned representation of all three encoder-decoder configurations achieves better results than the raw data in all tasks/scenarios. Out of all three encoder-decoder configurations, k-neighbor-encoder with 1-decoder consistently outperforms the others across different tasks/scenarios; this implies that the k-neighbor-encoder with 1-decoder configuration may be the best setup for neighbor-encoder.

By comparing the results under the two different scenarios ("clean" and "noisy"), we can see that neighbor-encoder have almost identical performance in both scenarios while autoencoder suffers in the noisy scenario. This fact further demonstrates the robustness of neighbor-encoder: we do not need the data to be perfect for neighbor-encoder to work. Even if the data includes a lot of noisy and irrelevant dimensions (which is typically the case, except for contrived data sets), the neighbor-

encoder can benefit from a suitable neighborhood function (e.g., time series subspace neighbor) which automatically ignores the irrelevant dimensions.

Figure 1 further demonstrates this advantage of neighbor-encoder. Here we use t-SNE to project various representations of the data of subject 1 into $2D$ space. The representations include the raw data itself, the latent representation learned by autoencoder, and the latent representation learned by neighbor-encoder. We can see that the latent representation learned by neighbor-encoder provides a much more meaningful visualization of different classes than the two rival methods in the face of noisy/irrelevant dimensions.

4.3 INSTRUMENTAL SOUNDS

To further demonstrate the versatility of the neighbor-encoder framework, we conducted experiments on a data set for predominant instrumental sound recognition in polyphonic music, using a benchmark dataset collected by Fuhrmann (2012). The training data consists of $6,951$ 3-second long music clips, with each clip labeled with one of the following primary instruments: cello, clarinet, flute, acoustic guitar, electric guitar, Hammond organ, piano, saxophone, trumpet, violin, or singing voice. We performed classification, clustering, and visualization experiments by applying 10-fold cross validation on the training data. Similar to Section 4.2, we use accuracy, adjusted Rand index, and R-precision as the performance metric for classification, clustering, and visualization respectively.

The neighborhood function we adopted for this data set is the feature space neighbor definition. Assume that we have the MFCC of an audio clip (which has 20 coefficients and 130 time frames); we can then evaluate its mean μ and standard deviation σ across time, and use the resulting 40-dimensional vector (which is produced by concatenating μ and σ) to represent the audio clip. We define the nearest neighbors of this audio clip as the clips closest to it in this 40-dimensional space. However, note that this 40-dimensional feature vector is only used in the neighborhood function; after the nearest neighbors of an audio clip is found, we use the mel-frequency spectrum of the audio clip and its neighbors as the input/output features of the encoder-decoder network.

We use librosa to extract both MFCC and mel-frequency spectrum of the audio clips (McFee et al., 2015). The only non-default setting we use here is the windows size for short-time Fourier transform. The default value is $2,048$, and we use $1,024$. Under our feature extraction settings, the MFCC has 20 coefficients and 130 time frames and mel-frequency spectrum has 128 mel-frequency bins and 130 time frames.

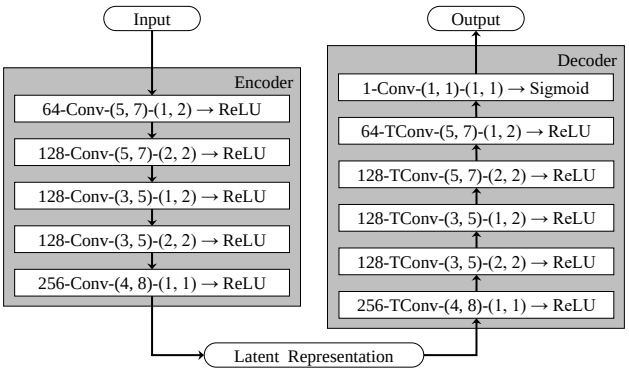


Figure 8: Network architecture for the decoder and decoder. 64-Conv-(5, 7)-(1, 2) denotes a $2D$ convolutional layer with 64 5×7 kernels and stride of $(1, 2)$. ReLU denotes rectified linear unit. TConv denotes transposed $2D$ convolutional layer. The first dimension within the parenthesis is time, and the second dimension is frequency.

We use a 5-layer $2D$ convolutional net as the encoder and a 5-layer transposed $2D$ convolutional net as the decoder. The encoder-decoder network architecture is summarized in Figure 8. In each iteration of SGD, we randomly selected a mini-batch of music clips, and within each clip, we randomly selected a segment of 16 consecutive frames. In other words, the network is trained on segments within the music clips instead of the music clip as a whole. The same sampling process is also

applied to their corresponding neighbors when $k > 1$. For the $k = 1$ case, the same 16 frames are used as the target as we are training autoencoder in this case. In contrast, the full clip is used at the test phase, and we apply global averaging pooling to the output of the encoder to obtain the latent representation. We have tried several other network architecture and training schemes (e.g., varying the length of segment); however, each time the conclusion from the experimental result is essentially unchanged.

Similar to Section 4.1, in this experiment we also regard each audio clip as the nearest neighbor of itself; in other words, when $k = 1$, the network is autoencoder. In addition to classification and clustering, we have also applied PCA and t-SNE as in Section 4.2 to the learned representation and evaluated the data visualization capability with R-precision. Figure 9 shows the results; the trend is similar to Figure 5 where the neighbor-encoder outperforms autoencoder in various setting of k ($k = 8, 16, 24$).

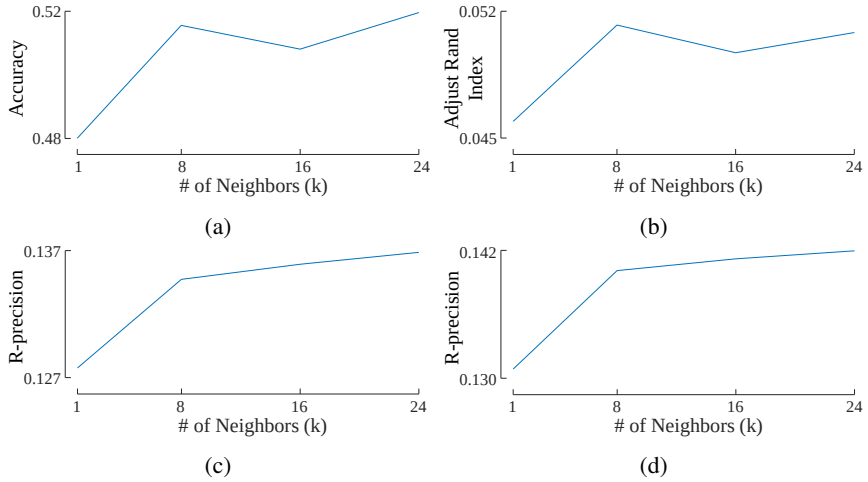


Figure 9: a) Classification accuracy, b) the adjusted Rand index result for clustering, c) visualization (PCA) R-precision, and d) visualization (t-SNE) R-precision as we vary k, the number of neighbors used in the neighbor-encoder. When $k = 1$, the network reduces to autoencoder.

The results demonstrate that neighbor-encoder outperforms autoencoder in various tasks with music data; furthermore, it is shown that the neighborhood function does not need to be defined in the same feature space as the input/output data of the neighbor-encoder network.

5 Conclusion

In this work, we have proposed an unsupervised learning framework called neighbor-encoder that is both *general*, in that it can easily be applied to data in various domains, and *versatile* as it can incorporate domain knowledge by utilizing different neighborhood functions. We have showcased the effectiveness of neighbor-encoder compared to autoencoder in various domains, including images, time series, music, etc. In future work, we plan to demonstrate the usefulness of the neighbor-encoder in more practical and applied tasks, including information retrieval.

References

Pulkit Agrawal, Joao Carreira, and Jitendra Malik. Learning to see by moving. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 37–45, 2015.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pp. 153–160, 2007.

Piotr Bojanowski and Armand Joulin. Unsupervised learning by predicting noise. In *Proceedings of the 34th international conference on Machine learning*, 2017.

Yu Chen and Mohammed J. Zaki. Kate: K-competitive autoencoder for text. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 85–94. ACM, 2017.

Adam Coates and Andrew Y Ng. Learning feature representations with k-means. In *Neural networks: Tricks of the trade*, pp. 561–580. Springer, 2012.

Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 135–144. ACM, 2017.

Ferdinand Fuhrmann. *Automatic musical instrument recognition from polyphonic music audio signals*. PhD thesis, Universitat Pompeu Fabra, 2012.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864. ACM, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Fu Jie Huang, Y-Lan Boureau, Yann LeCun, et al. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pp. 1–8. IEEE, 2007.

Yu-Siang Huang, Szu-Yu Chou, and Yi-Hsuan Yang. Similarity embedding network for unsupervised sequential pattern learning by playing music puzzle games. *arXiv preprint arXiv:1709.04384*, 2017.

Dinesh Jayaraman and Kristen Grauman. Learning image representations tied to ego-motion. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1413–1421, 2015.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69, 1982.

Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th annual international conference on machine learning*, pp. 689–696. ACM, 2009.

Alireza Makhzani and Brendan Frey. K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013.

Alireza Makhzani and Brendan J Frey. Winner-take-all autoencoders. In *Advances in Neural Information Processing Systems*, pp. 2791–2799, 2015.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Evaluation in information retrieval*, chapter 8, pp. 161. Cambridge University Press, 2008.

Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, pp. 18–25, 2015.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013b.

Minh Nguyen, Sanjay Purushotham, Hien To, and Cyrus Shahabi. m-tsne: A framework for visualizing high-dimensional multivariate time series. *arXiv preprint arXiv:1708.07942*, 2017.

Deepak Pathak, Ross Girshick, Piotr Dollár, Trevor Darrell, and Bharath Hariharan. Learning features by watching objects move. In *Computer Vision and Pattern Recognition, 2017. CVPR'17. IEEE Conference on*, 2017.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710. ACM, 2014.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Attila Reiss and Didier Stricker. Creating and benchmarking a new dataset for physical activity monitoring. In *Proceedings of the 5th International Conference on PErvasive Technologies Related to Assistive Environments*, pp. 40. ACM, 2012a.

Attila Reiss and Didier Stricker. Introducing a new benchmarked dataset for activity monitoring. In *Wearable Computers (ISWC), 2012 16th International Symposium on*, pp. 108–109. IEEE, 2012b.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 385–394. ACM, 2017.

Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International Conference on Machine Learning*, pp. 843–852, 2015.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pp. 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103. ACM, 2008.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.

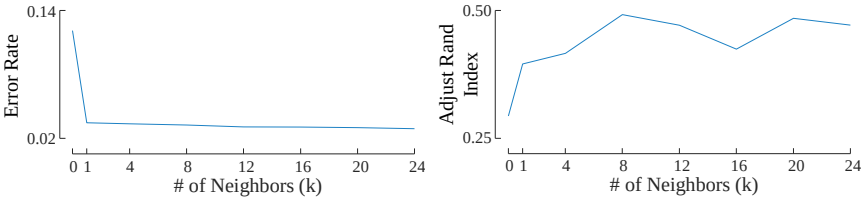
Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2794–2802, 2015.

Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *Proceedings of the 34th international conference on Machine learning*, 2017.

Chin-Chia Michael Yeh, Nickolas Kavantzas, and Eamonn Keogh. Matrix profile vi: Meaningful multidimensional motif discovery. In *2017 IEEE 17th International Conference on Data Mining (ICDM)*, 2017.

A Additional Results for Handwritten Digits

In Section 4.1, we have shown the result using a neighbor definition which counts the input data as the nearest neighbor of itself. Such neighbor definition can be used to evaluate autoencoder versus k-neighbor-encoder; however, the performance of simple 1-neighbor-encoder cannot be evaluated as 1-neighbor-encoder is autoencoder under this neighbor definition. In this section, we have repeat the same experiment on MNIST database as Section 4.1 with a slightly modified definition for neighbors. We have excluded the input data from the neighbor search which is a more conventional way to define neighbors. The experiment result is shown in Figure 10 with $k = 0$ being autoencoder. Surprisingly, the 1-neighbor-encoder outperforms autoencoder to a large extent by simply training to network to construct the nearest neighbor rather than the data itself. The overall trend is similar to the one shown in Figure 5 where more neighbors helps the network learns better encoder in terms of discriminability of the learned representation.



(a) (b)

Figure 10: a) Classification error rate and b) clustering adjusted Rand index as we vary the number of neighbor for neighbor encoder. The $k = 0$ case is autoencoder, $k = 1$ case is 1-neighbor-encoder, and the rest cases are k-neighbor-encoder with different settings of k.