Deep Embedding Kernel

Linh Le [1]   **Ying Xie** [2]

Abstract

In this paper, we propose a novel supervised learning method that is called Deep Embedding Kernel (DEK). DEK combines the advantages of deep learning and kernel methods in a unified framework. More specifically, DEK is a learnable kernel represented by a newly designed deep architecture. Compared with pre-defined kernels, this kernel can be explicitly trained to map data to an optimized high-level feature space where data may have favorable features toward the application. Compared with typical deep learning using SoftMax or logistic regression as the top layer, DEK is expected to be more generalizable to new data. Experimental results show that DEK has superior performance than typical machine learning methods in identity detection, classification, regression, dimension reduction, and transfer learning.

1. Introduction

We consider two major branches of machine learning, kernel methods (Hofmann et al., 2008) and deep learning (Schmidhuber, 2015). Kernel methods center around the kernel trick (Hofmann et al., 2008) – using a pre-defined kernel function to implicitly map data to a new feature space. However, this implicit mapping is rather heuristic in that there is no guarantee that the pre-defined kernel can lead to a more favorable feature space where data has better distribution towards the application. Hyper-parameter tuning algorithms like grid-search may improve the model performance (i.e. less prediction errors), but this brutal-force strategy does not fundamentally solve the problem of using pre-defined kernels.

Deep learning, on the other hand, utilizes a high number of parameters structured by layers of neural networks to

map the data to an explicit feature space with specified dimensionality (Schmidhuber, 2015). The parameters of the network that determines the mapping are typically tuned based on an explicit learning objective. In other words, by deep learning, the mapping of data into high-level representations is directly guided by the given learning objective through some top-down learning processes such as gradient descend. Therefore, learning objectives play critical roles in the quality of mapping. Frequently used learning objectives try to minimize training errors, which may not have the desired generalization ability according statistical learning theory (Vapnik, 1999). The work in (Tang, 2013) tries to improve generalization ability of deep learning by using linear SVM at the top layer, but the computational complexity of integrating SVM to deep learning is high. Another restriction of deep learning is that the dimensionality of the mapped feature space is pre-specified, instead of being learned.

In this paper, we try to address the problems of both kernel machines and deep learning by proposing a new supervised learning method called Deep Embedding Kernel (DEK) that is able to utilize the strengths of each method to address the weakness of the other in a unified framework. First of all, DEK does not explicitly map data to a feature space with pre-specified dimensionality, nor implicitly map data through a pre-defined kernel; instead, DEK uses a newly designed deep architecture to represent a learnable kernel. In other words, DEK utilizes the learning power of deep learning to train a kernel, which in turn implicitly maps data to a high dimensional feature space. The learning objective of DEK specifies a desired relationship of data in the mapped feature space. Then the kernel represented by DEK trained by the learning objective is expected to implicitly map data to such a feature space. Therefore, the whole mapped feature space, including its dimensionality, is learned via deep learning. Using deep architectures to learn a kernel, instead of directly learn the feature space also has the advantages of flexibility in that the learned kernel can be applied to a wide range of supervised learning tasks including identity detection, general classification, dimension reduction, regression, and other kernel based machine learning applications.

The architecture of DEK integrates two learning networks, namely **kernel network** and **embedding network**. The kernel network directly represents the parameterized kernel

---

[1]Institute of Analytics and Data Science, Kennesaw State University, Georgia, USA [2]Department of Information Technology, Kennesaw State University, Georgia, USA . Correspondence to: Linh Le <lle12@students.kennesaw.edu>, Ying Xie <yxie2@kennesaw.edu>.

trained from data, while the embedding network tries to learn optimized data representations to feed into the kernel network. The training of both networks is done in a single gradient descent process with the same learning objective that specifies an optimized relationship of data in the desired feature space.

DEK can be easily extended to work on unstructured data by laying itself on top of deep architectures designed for certain type of unstructured data, such as Convolutional Neural Network (CNN) for image data, Recurrent Neural Network (RNN) for sequential data, or the combination of CNN and RNN for video data. By this extension, the particular deep architecture used on unstructured data will learn vector embedding from unstructured data in the same learning process where embedding network and kernel network of DEK are trained via gradient descent. Moreover, DEK can be used to boost the learning power of transfer learning by being laid over a trained deep network that outputs vector embedding.

In this paper, we will demonstrate that DEK has superior performance over other typical supervised learning methods, such as Kernel Support Vector Machines, Gradient Boosting Trees, Random Forests, and Neural Networks on multiple learning tasks, including identity detection, general classification, regression, dimension reduction, and transfer learning.

2. Related Works

Various attempts were made to stack kernels to form deep architectures in (Zhuang et al., 2011), (Strobl & Visweswaran, 2013), (Jose et al., 2013), (Jiu & Sahbi, 2017), and (Sahbi, 2017). The output of this type of deep architecture is typically a highly nonlinear combination of input kernels. The learning process of stacking kernels involves jointly training a SVM classifier and modifying network weights as well as kernel parameters using gradient descent. Some limitations of these works include 1) using pre-defined kernels (such as RBF kernel) as input neurons limits the flexibility and capacity of learning by the deep architecture; 2) using SVM optimization as the learning objective for training the deep architecture is computationally expensive. The proposed DEK tries to maximize the learning by first learning an optimal high-level representation of data, followed by learning a highly non-linear kernel, which is in turn based on dimension-wise relationships of the high-level representations. In other words, DEK forms the kernel based on much finer granularity of relationship between data instead of starting with pre-defined kernel functions on the whole set or subsets of dimensions of data. Furthermore, the learning objective of DEK can be evaluated online by each pair of data without the need of quadratic programming on at least a batch of data.

Similarly, stacking SVMs to deepen the model architecture was discussed in (Wiering & Schomaker, 2014). The authors of this work use different SVMs to extract latent features in different subsets of dimensions in the data. A global SVM is then used to aggregate all SVMs to form a final decision layer. However, because of computational expenses of SVMs, it is not practical to form a deep architecture by simply stacking SVMs. Therefore, the extent to which this type of stacking takes advantages of deep learning is rather limited. On the contrary, DEK can fully embrace the learning power of deep learning, given that DEK itself is a true deep architecture without any add-on restriction on depths of the network. Instead of stacking SVMs, the work in (Tang, 2013) tried to improve generalization ability of deep learning by using linear SVM classifier at the top layer to define the learning objective. But this architecture strictly ties with classification tasks and training a SVM at the top layer is still non-trivial as it requires quadratic programming on a batch of data.

There were works computing similarity of data using deep architectures on image data in (Zbontar & LeCun, 2015) and (Zagoruyko & Komodakis, 2015). However, their similarity computing is specialized on a particular task and unable to be generalized to other learning tasks. Moreover, their output similarities do not necessarily possess the character of symmetricity, therefore cannot be used as kernels.

Googles FaceNet uses a cost function that is called triplet loss on facial identification (Schroff et al., 2015). Each evaluation of triplet loss involves selecting three instances x_i, x_i^+, x_i^- that satisfies the following criteria: x_i is an anchor point, x_i^+ is another data point with the same class as x_i, x_i^- is a data point with a different class than x_i, and the following inequality holds.

$$\|x_i - x_i^+\|_2^2 > \|x_i - x_i^-\|_2^2 \tag{1}$$

The deep network is then trying to learn an mapping $f(\cdot)$ such that

$$\|f(x_i) - f(x_i^+)\|_2^2 < \|f(x_i) - f(x_i^-)\|_2^2 \qquad \forall i \tag{2}$$

Therefore, the learning objective of the deep learning can be expressed as minimize the following cost function:

$$L = \sum_{i=1}^{N}(\|f(x_i) - f(x_i^+)\|_2^2 - \|f(x_i) - f(x_i^-)\|_2^2 + \alpha) \tag{3}$$

with α being a margin parameter. The Triplet Loss function was extended to other identity detection tasks such as voice

recognition (Bredin, 2017). An issue with triplet loss based cost function, according to (Hermans et al., 2017), is that the training of the network requires a large training data that contains a sufficient amount of triplets that satisfies the described criteria. In contrast, DEK can evaluate the learning object online by using every pair from the training data (though it is not necessary to use every pair if the training data is large enough). From another perspective, DEK may even be able to solve the "Small Training Data" problem by forming n^2 training pairs from just n instances.

Lastly, we would like to mention transfer learning. In the context of deep learning, transfer learning aims to reuse a deep network that is trained for one application to another relevant task (Pan & Yang, 2010) and (Bengio, 2012). A popular way of doing transfer learning is to replace the decision layer(s) of the trained deep network with a new one for the new task. DEK can work as a general decision layers to be laid on top of a pre-trained network. Experimental results demonstrate that DEK has better performance than Multi-layer Perceptron with triplet loss for being used as the decision layers in transfer learning.

3. Methodology

The goal of our methodology is to learn an optimized feature space of data with desired features for the application. This optimized space is determined by DEK, a learnable kernel that is represented by a deep architecture. When we design DEK, we consider the following factors. First, since it represents a kernel, DEK takes a pair of data instances as input and output their similarity. Similarity of data can be computed based on different representations of data at different abstraction levels. We want the DEK to be able to learn data similarity based on optimized data representations. Then based on the given data representation, we want the DEK to be able to learn a similarity function that is complex enough to map data to an optimized space with desired data distributions. Therefore, DEK is designed to have two learning components, namely embedding network and kernel network, integrated in a unified deep architecture. These two learning components will be trained using the same learning objective in a single learning process. The overall architecture of DEK is shown in Figure 1.

3.1. Kernel Network

As shown in Figure 1, the input of the kernel network is denoted as U, which is formed by the outputs of the two branches of the embedding network, which are o_i and o_j respectively. More specifically, U can be expressed as the following function of o_i and o_j

$$U = \left\{ \begin{array}{l} o_{i_1} * o_{j_1}, o_{i_2} * o_{j_2}, \ldots o_{i_d} * o_{j_d}, \\ |o_{i_1} - o_{j_1}|, \ldots |o_{i_d} - o_{j_d}| \end{array} \right\} \quad (4)$$
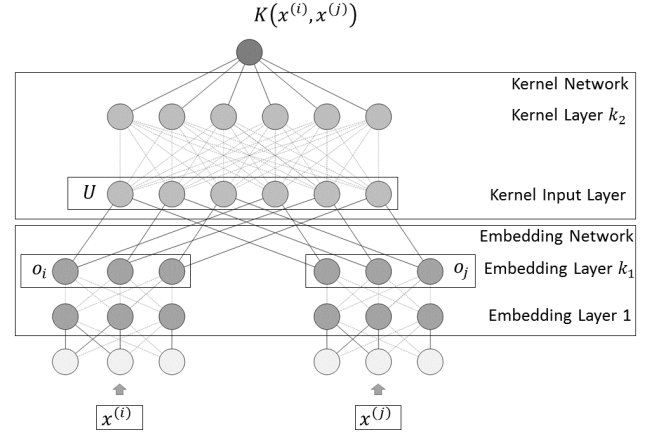

Figure 1. The Structure of DEK

Where o_{i_k} denotes the kth dimension of o_i, and d is the dimensionality of o_i and o_j . In other words, each neuron in the input layer of the kernel network represents a symmetric relationship of o_i and o_j on a single dimension. The use of fine granularity of relationship on each individual dimension as input provides more room for learning, compared with directly using different pre-defined kernel functions on o_i and o_j as inputs. Essentially, this design of inputs allows the kernel network to learn a kernel that is a highly nonlinear combination of angles and distances of the data pairs in the space that is learned by the underneath embedding network. Furthermore, this design of inputs guarantees the output similarity is symmetric.

The output of the kernel network is the probability that sample i and j belong to the same class. Formally, given sample i and j, the output can be expressed as

$$K(o_i, o_j) = P(y^{(i)} = y^{(j)} | x^{(i)}, x^{(j)})$$
$$= sigmoid(W_{out}^{(K)} \cdot H_{out}^{(K)} + b_{out}^{(K)}) \quad (5)$$

With $W_{out}^{(K)}$ and $b_{out}^{(K)}$ being the parameters of the output layer, and $H_{out}^{(K)}$ being the input, of the kernel network. Obviously we have $K(o_i, o_j) = K(o_j, o_i) > 0$. Therefore, $K(\cdot)$ is a kernel function.

To train the kernel network (as well as the whole DEK), we label each pair in the following way.

$$\begin{cases} Y^{(i,j)} = 1 \iff y^{(i)} = y^{(j)} \\ Y^{(i,j)} = 0 \iff y^{(i)} \neq y^{(j)} \end{cases} \quad (6)$$

That is, if instance i and j belong to the same class, the label for the pair of i and j is 1, otherwise it is 0. Then we define

the learning objective of training DEK (including kernel network) is to minimize the following cost function.

$$L = \sum_{data} (Y^{(i,j)} \log(K(o_i, o_j) + \\ (1 - Y^{(i,j)}) \log(1 - K(o_i, o_j))) \tag{7}$$

3.2. Embedding Network

The purpose of the embedding network is to learn optimized high-level representations of data to feed into the kernel network as inputs. Let the mapping made by the embedding network be $E(\cdot)$ then the high-level representation of sample $x^{(i)}$ can be represented as $o_i = E(x^{(i)})$. The goal of designing the embedding network is to increase the learning capacity of the final kernel. Experimental results demonstrate that the embedding network positively contributes to the performance of DEK.

The training of embedding network is in the same gradient descent process using the same cost function as in Equation (7).

3.3. Overall Design

Suppose the embedding network has k_1 hidden layers $H_1^{(e)} \dots H_{k_1}^{(e)}$ and the kernel network has k_2 hidden layers $H_1^{(K)} \dots H_{k_2}^{(K)}$. Also suppose the input layer of the embedding network is $H_0^{(e)}$ and of the kernel network is $H_0^{(K)}$, and the weights and bias of layer i of network j are $W_i^{(j)}$ and $b_i^{(j)}$. The computational flow from a sample pair $(x^{(i)}, x^{(j)})$ can be expressed as

- The embedding of $x^{(i)}$:

$$H_0^{(e)}(i) = x^{(i)}$$
$$H_1^{(e)}(i) = \sigma(W_0^{(e)} \cdot H_0^{(e)}(i) + b_0^{(e)})$$
$$\dots$$
$$o_i = H_{k_1}^{(e)}(i) = \sigma(W_{k_1-1}^{(e)} H_{k_1-1}^{(e)}(i) + b_{k_1-1}^{(e)})$$

- The embedding of $x^{(j)}$:

$$H_0^{(e)}(j) = x^{(j)}$$
$$H_1^{(e)}(j) = \sigma(W_0^{(e)} \cdot H_0^{(e)}(j) + b_0^{(e)})$$
$$\dots$$
$$o_j = H_{k_1}^{(e)}(j) = \sigma(W_{k_1-1}^{(e)} H_{k_1-1}^{(e)}(j) + b_{k_1-1}^{(e)})$$

- Input to the kernel network:

$$U = o_i \bullet o_j = H_0^{(K)}$$
$$H_1^{(K)} = \sigma(W_0^{(K)} \cdot H_0^{(K)} + b_0^{(K)})$$

$$\dots$$

$$H_{k_2}^{(K)} = \sigma(W_{k_2-1}^{(K)} \cdot H_{k_2-1}^{(K)} + b_{k_2-1}^{(K)})$$
$$K(x^{(i)}, x^{(j)}) = s(W_{k_2}^{(K)} \cdot H_{k_2}^{(K)} + b_{k_2}^{(K)})$$

with $\sigma(\cdot)$ being the activation function, $s(\cdot)$ being the output function, and $\bullet$ being the dimension-wise similarity operator as discussed:

$$o_i \bullet o_j = \left\{ \begin{array}{c} o_{i_1} * o_{j_1}, o_{i_2} * o_{j_2}, \dots o_{i_d} * o_{j_d}, \\ |o_{i_1} - o_{j_1}|, \dots |o_{i_d} - o_{j_d}| \end{array} \right\}$$

Layers in both component network are updated with gradient descent:

$$W_i^{(j)} \leftarrow W_i^{(j)} - \alpha \frac{\partial L}{\partial W_i^{(j)}}$$
$$b_i^{(j)} \leftarrow b_i^{(j)} - \alpha \frac{\partial L}{\partial b_i^{(j)}} \tag{8}$$

A unified structure is currently being employed on all layers to simplify the training process. In detail, all embedding layers have k hidden neurons, and all kernel layers have $2k$ neurons, where $k = \alpha d$ with d being the dimensionality of the original data and α being an integer factor (typically, we use $\alpha \in \{1, 2, 3, 4\}$).

4. DEK for Unstructured Data

If data is not in the form of structured records (such as images, sequential data, or text), we can lay DEK on top of CNN, RNN, or other deep architectures that are suitable for the given unstructured data to form a unified deep neural network for supervised learning. The deep neural network with DEK on top for both image data and sequential data are shown in Figure 2.

In this type of deep architecture, there are three integrated learning components that will be trained in the same learning process. The first is to learn an optimized vector embedding of the unstructured data; the second is to learn an optimized high-level embedding based on the bottom vector embedding; and the last is to learn a complex similarity function based on the high-level embedding of the data. Again, all these components will be trained in the same learning process with the same learning objective.

The deep architecture, shown in 2, can be viewed as a framework for transfer learning. In other words, the bottom component of vector embedding can be replaced by a network that is trained from data with similar natures.
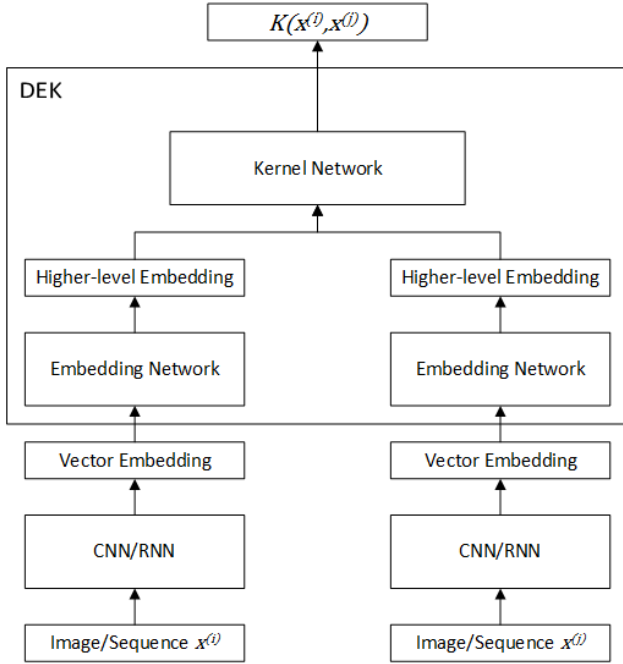
Figure 2. The DEK Architecture for Unstructured Data

Figure 3. Accuracy Rates of Speaker Identification Models

5. DEK for Different Types of Supervised Learning

In this section, we describe different applications of DEK on supervised learning including identity detection, classification, regression, dimension reduction, and transfer learning. All experiments for each of the above tasks are conducted in Python version 2.7.12. Deep models are implemented using the package Theano (Bergstra et al., 2010), other machine learning models (including the regular MLP) are from the Sci-Kit Learn (Pedregosa et al., 2011) package. Visualizations are generated using the Matplotlib library (Hunter, 2007).

5.1. Identity Detection

The problem of identity detection can be defined as assigning an identity to a query sample (e.g. a speech segment or a facial image). A common supervised learning strategy to solve this problem is to assign the identity to the query sample based on its nearest neighbors in the training set. Identity detection with DEK feeds the query sample and each of the training sample into the trained deep network and finds the nearest neighbors of the query sample using the outputted kernel values. In our experimental studies, we apply DEK to both speaker identification and facial recognition. Since both tasks are based on unstructured data (i.e. speech segments and facial images), we use the extended DEK framework discussed in section 4. In other words,
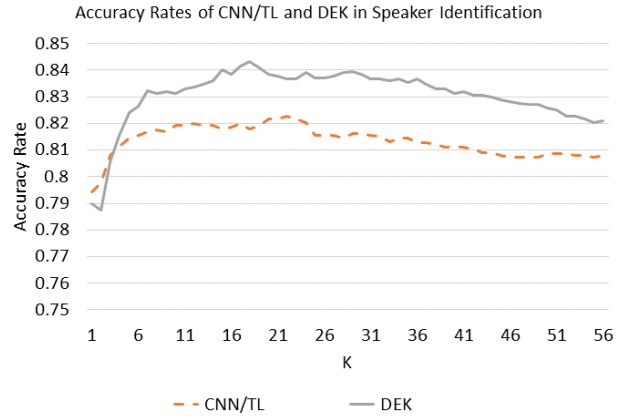
we lay DEK on top of deep architectures that are proper to underneath unstructured data.

Speaker Identification

Most speaker identification models work by first extracting features from the speech segments. We choose spectrograms as the feature set to be modeled in this task. In short, spectrograms are representations of audio segments in the time/frequency space and have characteristics similar to images. Therefore, CNN is a proper deep architecture to model spectrograms. In other words, we lay DEK on the top of CNN to model the similarities of the speech segments.

In our experiment on speaker identification, we use the Characterizing Individual Speakers (CHAINS) dataset - (Cummins et al., 2006). The data consists of speech segments from 36 persons in various speaking conditions. In our study, we use only segments recorded in studio where the speakers read scripts in normal talking speed. In the preprocessing phase, we first split the speech segments into syllables using silent gaps; then pad each of them to be 1.5-second-long; and finally transform them into spectrograms. We compare two models to identify the speakers, one is a CNN using Triplet Loss cost function (CNN/TL), and the other is extended DEK laid on top of the same CNN. The preprocessed data is split into 75% training and 25% testing.

The accuracy rates of the two models by number of nearest neighbors from 1 to 55 is shown in Figure 3. It can be seen that the DEK provides a significant lift in accuracy rate (over 2%) over the CNN/TL model.

Facial Recognition

We study the performance of DEK on transfer learning on facial recognition. The data we use is Indian Movie Face Database (IMFDB) – (Setty et al., 2013). This dataset contains facial images of Indian movie actors and actresses.
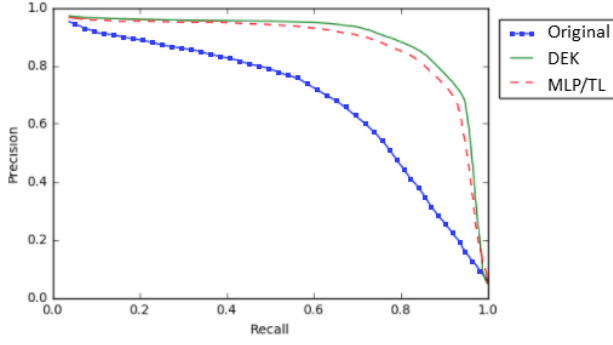
Figure 4. Performance of Transfer Learning Models with DEK and MLP/TL on Facial Recognition

Figure 5. Contributions of Embedding Layers to DEK Performance

We build two transfer learning models based on a pretrained Google FaceNet (available from `https://github.com/davidsandberg/facenet`). This version of FaceNet was trained from about 500,000 facial images. We build two transfer learning models based on the pretrained Facenet. One model lays a Multi-layer Perceptron using Triplet Loss (MLP/TL) on top of the pretrained FaceNet, the other lays DEK on the pretrained Facenet. Both models are trained and tested on the same subsets from the IMFDB data (75% training, 25% testing). The trained MLP/TL outputs vector embedding based on which we can compute the pairwise distances among images. The trained DEK outputs kernel values that can be interpreted as similarities.

To evaluate the two models, each image in the testing set is used as a query image to rank all images in the training set in the ascending order of their distances outputted by the MLP/TL model, and in the descending order of similarities outputted by DEK. We then plot the average precision-recall curve for these two rankings. We also plot the precision-recall curve generated by the pretrained FaceNet without transfer learning as the baseline. As shown in Figure 4, both transfer learning models make substantial improvements over the pretrained FaceNet. DEK makes further improvement over MLP/TL at almost every recall level. Given MLP/TL has already achieved near-perfect precisions, the further improvement made by DEK is significant. Therefore, DEK can be used as the desired solution to facial recognition in critical applications where very high accuracy is demanded.

We further study the contribution of the embedding network of DEK towards the performance in this experiment. More specifically, we build a transfer learning model by only laying the kernel network component of DEK on the top the pretrained FaceNet. We denote this model as DEK-EN. Both DEK and DEK-EN are trained independently on IMFDB. The precision-recall curves of both models are plotted in Figure 5. It can be seen that the embedding network
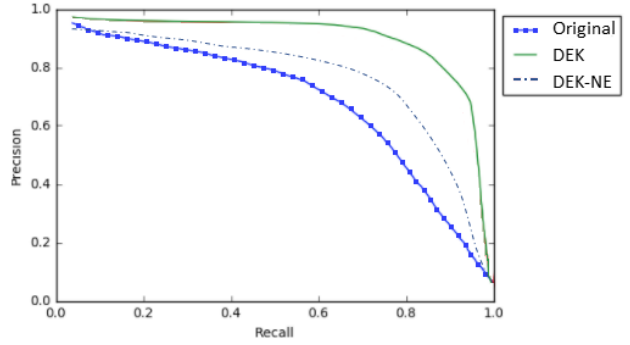
of DEK contribute significantly towards the performance. This experimental result re-enforces our hypothesis that the incorporating of the embedding network in DEK increases the learning capacity of the model.

5.2. General Classification

The learning objective of DEK (described in section 3.1) naturally fits into identity detection problems, in that the desired similarity of two samples belonging to the same identity is 1 and the desired similarity of two samples belonging to different identities is 0. However, for general classification problems, this learning objective may be over-strict, given that two samples belong to the same class may not necessarily have the same level of similarity as two belonging to the same identity. Therefore, to adapt DEK to general classification problems, a local pairing strategy is proposed and added to the learning process of DEK. In details, we use local pairing strategy to generate training pairs at certain interval of iterations. For example, local pairing strategy is applied to generate training pairs at the 1st, 51st, 101st, 151st, ... iterations. Other iterations between the interval use the same training pairs generated most recently. The local pairing strategy works as follows. First, all pairs of data are fed into DEK; each sample is used as reference to rank all other samples in descending order of kernel values outputted by DEK. A certain recall level (e.g., 0.1) is then used to determine the neighborhood of the reference sample. Within the neighborhood, we form positive pairs between the reference sample and the samples of the same class, and form negative pairs between the reference sample and the samples of different classes. The local pairing strategy is illustrated in Figure 6. By using local pairing strategy, we avoid to force the similarity of distant samples of the same class to be close to 1.

To study the performance of DEK with local pairing strategy on general classification, we compare SVM using DEK

Table 1. Accuracy Rates of Models in Experiments for Classification

DATASET	SVM/**DEK**	KNN/**DEK**	SVM/RBF	GB	RF	MLP
SEGMENT (ZHANG, 1992)	**0.9691**	*0.9678*	0.9647	0.9604	0.9610	0.9593
CARDIOTOCOGRAPHY (AYRES-DE CAMPOS ET AL., 2000)	*0.9893*	**0.9899**	0.9879	0.9825	0.9846	0.9850
MESSIDOR FEATURES (DECENCIÈRE ET AL., 2014)	**0.7803**	*0.7746*	0.7543	0.7110	0.7168	0.7222
WAVEFORM (BREIMAN ET AL., 1984)	*0.8696*	**0.8704**	0.8684	0.8488	0.8456	0.8672
PIMA DIABETE (SMITH ET AL., 1988)	*0.7839*	**0.7865**	0.7708	0.7396	0.7604	0.7630

Table 2. R^2 of Models in Experiments for Regression

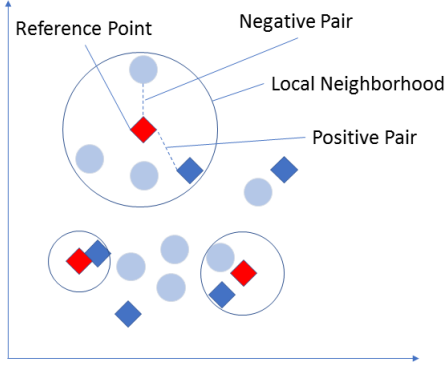| DATASET	SVR/**DEK**	KNN/**DEK**	SVR/RBF	GB	RF	MLP
CONCRETE (YEH, 1998)	0.8651	0.8980	0.8702	**0.9067**	0.8751	0.8119
AIRFOIL (BROOKS ET AL., 1989)	0.8242	**0.9195**	0.8371	0.8840	0.9047	0.8568
ENERGY EFFICIENCY (TSANAS & XIFARA, 2012)	0.9685	**0.9783**	0.9621	0.9775	0.9756	0.9470

Figure 6. Illustration of the Local Pairing Strategy

(SVM/DEK) and KNN using DEK (KNN/DEK) with other classification models including SVM using RBF kernel (SVM/RBF), Gradient Boosting Trees (GB) (Friedman, 2002), Random Forest (RF) (Liaw et al., 2002), and MLP on five datasets. The datasets are split into 50% training data and 50% testing data. The hyper-parameters used by RBF kernel used by SVM are optimized via grid-search on the trained dataset. Reported accuracy rates are computed in the testing set. Table 1 shows the test accuracy rates for all the models.

As can be seen, DEK-based SVM and DEK-based KNN achieve the best results in all datasets. The improvement is from 0.2% in the Cardiotocography data (comparing to SVM/RBF) to about 7% in the Messidor Features data (comparing to GB).

5.3. Regression

Unlike identity detection or classification models, determining the similarity of a sample pair in regression is not that obvious since the target value is now continuous. When applying DEK to regression, we model the similarity of sample pairs based on the similarity between their target values. In other words, let $K'(\cdot)$ be a similarity function defined on a pair of target values, then the network is trained so that $K(x^{(i)}, x^{(j)})$ approximates $K'(y^{(i)}, y^{(j)})$:

$$K(x^{(i)}, x^{(j)}) \approx K'(y^{(i)}, y^{(j)}) \tag{9}$$

We define $K'(y^{(i)}, y^{(j)}) = \exp\left(-\gamma|y^{(i)} - y^{(j)}|\right)$ with γ being a scale parameter. Accordingly, the output layer and cost function of the regression DEK are

$$K(x^{(i)}, x^{(j)}) = ReLU(W_{k_2}^{(K)} \cdot H_{k_2}^{(K)} + b_{k_2}^{(K)}) \tag{10}$$

$$L = \frac{1}{N} \sum_{i,j \in Data} (K(x^{(i)}, x^{(j)}) - K'(y^{(i)}, y^{(j)}))^2 \tag{11}$$

(with $ReLU(x) = \max(0, x)$)

To study the performance of DEK on regression, we compare Support Vector Regressor using DEK (SVR/DEK) and KNN using DEK (KNN/DEK) with other regression models including SVR using RBF kernel (hyper-parameters optimized via grid-search) (SVR/RBF), Gradient Boosting Trees Regressor (GB), Random Forest Regressor (RF), and MLP Regressor, on three datasets. A ratio of 50% training data and 50% testing data is also used. R^2 is used as the measurement to compare the models. Table 2 shows the performances of tested models in the regression task.

As can be seen, KNN/DEK achieves the best performance in two out of three datasets while being slightly behind the GB model in the Concrete dataset.

5.4. Dimension Reduction

As being a kernel function, a trained DEK can be used with kernel Principal Component Analysis (kPCA) to per-
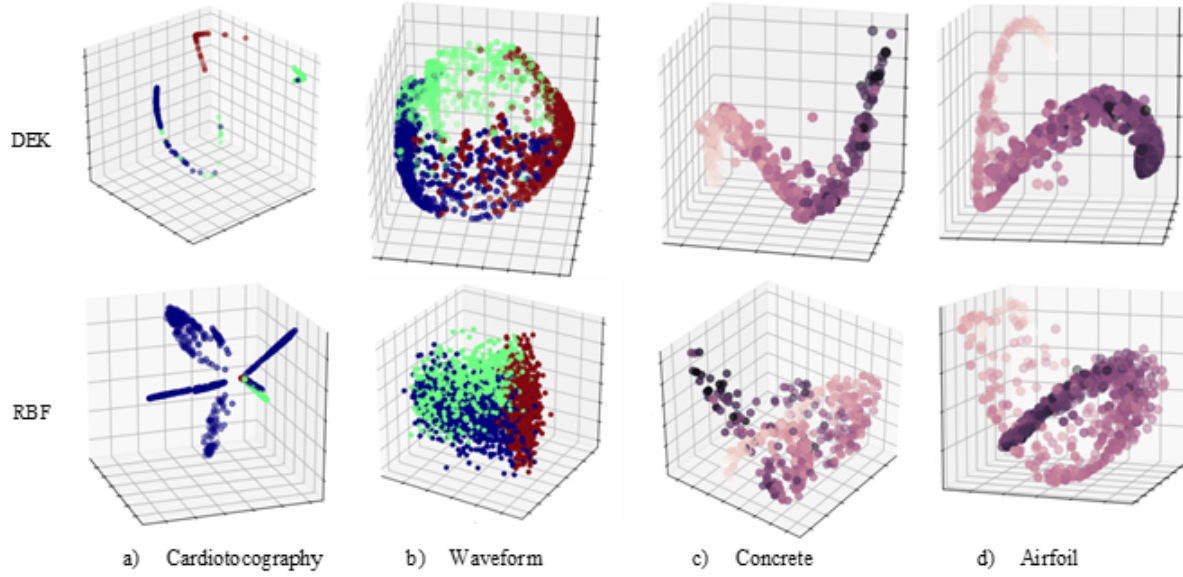
Figure 7. Visualization of Four Datasets in 3D Space

form dimension reduction on labeled data. We compare the performance of dimension reduction by kPCA with DEK and kPCA with RBF kernel (hyper-parameter optimized by grid-search with SVM) on two classification datasets (Cardiotocography and Waveform) and two regression datasets (Concrete and Airfoil). Figure 7 illustrates the four (testing) sets projected into 3D space by kPCA with a trained DEK and RBF kernel.

We can observe that, for the two classification datasets, DEK maps the data to a space where classes (represented by nodes colors) better match the geographical clusters. For the two regression datasets, the function patterns are clearer in the space mapped by DEK than RBF (the target values are represented by the shades of the nodes – darker nodes indicate higher target values).

6. Conclusion

In this paper, we propose a new learnable kernel that is called DEK to automatically learn an optimized feature space from training data. DEK is represented by a deep neural network that consists of two components: a deep embedding network and a deep kernel network. The integration of these two components in a unified framework is to maximize the learning power of the deep architecture. The deep embedding network is designed to learn high-level representations; while the deep kernel network is designed to further learn non-linear similarities. Besides the learning capabilities presented by the embedding network and the kernel network, DEK can also integrate deep architec-

tures for embedding learning on unstructured data. DEK can act as a general-purpose kernel function applicable in most supervised learning tasks including identity detection, classification, regression, and dimension reduction. DEK also achieves superior performance on transfer learning for facial recognition compared with frequently used fully connected layers built on pretrained deep networks. We plan to contribute DEK as an open source package on GitHub to promote its usages on different application domains.

References

Ayres-de Campos, Diogo, Bernardes, Joao, Garrido, Antonio, Marques-de Sa, Joaquim, and Pereira-Leite, Luis. Sisporto 2.0: a program for automated analysis of cardiotocograms. *Journal of Maternal-Fetal Medicine*, 9(5): 311–318, 2000.

Bengio, Yoshua. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pp. 17–36, 2012.

Bergstra, James, Breuleux, Olivier, Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde-Farley, David, and Bengio, Yoshua. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pp. 1–7, 2010.

Bredin, Hervé. Tristounet: triplet loss for speaker turn embedding. In *Acoustics, Speech and Signal Processing*

(ICASSP), 2017 IEEE International Conference on, pp. 5430–5434. IEEE, 2017.

Breiman, Leo, Friedman, Jerome, Stone, Charles J, and Olshen, Richard A. *Classification and regression trees*. CRC press, 1984.

Brooks, Thomas F, Pope, D Stuart, and Marcolini, Michael A. Airfoil self-noise and prediction. 1989.

Cummins, Fred, Grimaldi, Marco, Leonard, Thomas, and Simko, Juraj. The chains corpus: Characterizing individual speakers. In *Proc of SPECOM*, volume 6, pp. 431–435. Citeseer, 2006.

Decencière, Etienne, Zhang, Xiwei, Cazuguel, Guy, Lay, Bruno, Cochener, Béatrice, Trone, Caroline, Gain, Philippe, Ordonez, Richard, Massin, Pascale, Erginay, Ali, et al. Feedback on a publicly distributed image database: the messidor database. *Image Analysis & Stereology*, 33(3):231–234, 2014.

Friedman, Jerome H. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

Hermans, Alexander, Beyer, Lucas, and Leibe, Bastian. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017.

Hofmann, Thomas, Schölkopf, Bernhard, and Smola, Alexander J. Kernel methods in machine learning. *The annals of statistics*, pp. 1171–1220, 2008.

Hunter, John D. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

Jiu, Mingyuan and Sahbi, Hichem. Nonlinear deep kernel learning for image annotation. *IEEE Transactions on Image Processing*, 26(4):1820–1832, 2017.

Jose, Cijo, Goyal, Prasoon, Aggrwal, Parv, and Varma, Manik. Local deep kernel learning for efficient non-linear svm prediction. In *International Conference on Machine Learning*, pp. 486–494, 2013.

Liaw, Andy, Wiener, Matthew, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

Pan, Sinno Jialin and Yang, Qiang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

Pedregosa, Fabian, Varoquaux, Gaël, Gramfort, Alexandre, Michel, Vincent, Thirion, Bertrand, Grisel, Olivier, Blondel, Mathieu, Prettenhofer, Peter, Weiss, Ron, Dubourg, Vincent, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

Sahbi, Hichem. Coarse-to-fine deep kernel networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1131–1139, 2017.

Schmidhuber, Jürgen. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

Schroff, Florian, Kalenichenko, Dmitry, and Philbin, James. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815–823, 2015.

Setty, Shankar, Husain, Moula, Beham, Parisa, Gudavalli, Jyothi, Kandasamy, Menaka, Vaddi, Radhesyam, Hemadri, Vidyagouri, Karure, JC, Raju, Raja, Rajan, B, et al. Indian movie face database: a benchmark for face recognition under wide variations. In *Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG), 2013 Fourth National Conference on*, pp. 1–5. IEEE, 2013.

Smith, Jack W, Everhart, JE, Dickson, WC, Knowler, WC, and Johannes, RS. Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, pp. 261. American Medical Informatics Association, 1988.

Strobl, Eric V and Visweswaran, Shyam. Deep multiple kernel learning. In *Machine Learning and Applications (ICMLA), 2013 12th International Conference on*, volume 1, pp. 414–417. IEEE, 2013.

Tang, Yichuan. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*, 2013.

Tsanas, Athanasios and Xifara, Angeliki. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49:560–567, 2012.

Vapnik, Vladimir Naumovich. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

Wiering, Marco A and Schomaker, Lambert RB. Multi-layer support vector machines. *Regularization, optimization, kernels, and support vector machines*, pp. 457–476, 2014.

Yeh, I-C. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, 28(12):1797–1808, 1998.

Zagoruyko, Sergey and Komodakis, Nikos. Learning to compare image patches via convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pp. 4353–4361. IEEE, 2015.

Zbontar, Jure and LeCun, Yann. Computing the stereo matching cost with a convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1592–1599, 2015.

Zhang, Jianping. Selecting typical instances in instance-based learning. In *Machine Learning Proceedings 1992*, pp. 470–479. Elsevier, 1992.

Zhuang, Jinfeng, Tsang, Ivor W, and Hoi, Steven CH. Two-layer multiple kernel learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 909–917, 2011.