Outlier Detection for Multidimensional Time Series using Deep Neural Networks

Tung Kieu, Bin Yang, Christian S. Jensen
Department of Computer Science, Aalborg University
{tungkvt, byang, csj}@cs.aau.dk

***Abstract*—Due to the continued digitization of industrial and societal processes, including the deployment of networked sensors, we are witnessing a rapid proliferation of time-ordered observations, known as time series. For example, the behavior of drivers can be captured by GPS or accelerometer as a time series of speeds, directions, and accelerations. We propose a framework for outlier detection in time series that, for example, can be used for identifying dangerous driving behavior and hazardous road locations. Specifically, we first propose a method that generates statistical features to enrich the feature space of raw time series. Next, we utilize an autoencoder to reconstruct the enriched time series. The autoencoder performs dimensionality reduction to capture, using a small feature space, the most representative features of the enriched time series. As a result, the reconstructed time series only capture representative features, whereas outliers often have non-representative features. Therefore, deviations of the enriched time series from the reconstructed time series can be taken as indicators of outliers. We propose and study autoencoders based on convolutional neural networks and long-short term memory neural networks. In addition, we show that embedding of contextual information into the framework has the potential to further improve the accuracy of identifying outliers. We report on empirical studies with multiple time series data sets, which offers insight into the design properties of the proposed framework, indicating that it is effective at detecting outliers.**

I. Introduction

Many application domains, ranging from finance and neurology to geology and transportation, produce large volumes of sequences of multivariate timestamped observations, i.e., multidimensional time series. Analyzing such time series helps us understand the underlying systems that produce the time series, which enables us to predict future trends and behaviors of the systems as well as helps us detect outliers. In this paper, we focus on detecting outliers from multidimensional time series, which has a range of real-world applications.

For example, in intelligent transportation systems [1], [2], identifying hazardous road locations [3] becomes an increasingly important task as can help people improve the design of roads and road surfaces, which may prevent accidents. Detecting outliers in time series that capture the behaviors of drivers [4] is a promising approach to identify hazardous road locations. In particular, GPS and accelerometer sensors installed in vehicles produce time series of observations that capture the speeds, accelerations, and directions of vehicles [5], [6]. Such time series have three dimensions that capture speed, acceleration, and direction. When drivers encounter hazardous road locations, they may use the brakes hard or may make sudden direction changes. Such maneuvers then show up in the time series as outliers.

We study the use of deep neural network based autoencoders to identify outliers in multidimensional time series data. An autoencoder is an unsupervised learning method that fits well with the purpose of identifying outliers. In particular, an autoencoder accepts input data that is represented in a large feature space, performs dimensionality reduction, thereby capturing the most representative features of the input data using a small feature space, and reconstructs the input data based on the features in the small feature space. Since outliers often correspond to non-representative features, an autoencoder is likely to fail to reconstruct outliers using the small feature space. Therefore, deviations between the original input data and the reconstructed data can be taken as indicators of outliers.

Based on this idea, we propose an outlier detection framework that incorporates several deep neural network based autoencoders. First, we propose a method to enrich multidimensional time series with *derived features* and *statistical features* that capture different aspects of the temporal changes in the time series. This design also enriches the feature space of the raw time series. Next, we introduce two deep neural network based autoencoders, a 2D convolutional neural network based autoencoder and a long short term memory based autoencoder. We use these to reconstruct the enriched time series. The deviations between the reconstructed enriched time series and the original enriched time series are used as indicators of outliers. Further, we embed contextual information into the deep neural network based autoencoders, which further improves the accuracy of outlier detection.

The paper makes three contributions. It proposes a deep neural network based framework to identify outliers for enriched, multidimensional time series; it proposes two deep neural network based autoencoders, together with a context embedding method; and it reports on extensive experiments on several time series data sets to evaluate the design choices made and the effectiveness of the proposed outlier detection framework.

The rest of the paper is organized as follows. Section 2 describes preliminaries and autoencoders and formulates the paper's problem. Section 3 proposes the deep neural network based framework for outlier detection. Experimental results are reported in Section 4, and related work is covered in Section 5. Finally, Section 6 concludes the paper.

2375-0324/18/$31.00 ©2018 IEEE
DOI 10.1109/MDM.2018.00029

II. Preliminaries

We introduce important concepts and formalize the problem. We use lower case letter, e.g., s, to denote scalars; upper case letter, e.g., S, to denote vectors; and bold upper case letter, e.g., $\mathbf{W}$, to denote matrices.

A. Multidimensional Time Series

A multidimensional time series (or multivariate time series) is a sequence of vectors, where each vector represents the state of an entity at a specific time point.

In particular, a k-dimensional time series T is a sequence of k-dimensional vectors $T = \langle S_1, S_2, \ldots, S_C \rangle$ where $S_i = (s_i^1, s_i^2, \ldots, s_i^k)$ is a k-dimensional vector that describes the state of an entity at time t_i, for $1 \leq i \leq C$. Each dimension corresponds to a feature. The time gaps between two consecutive points are often the same, i.e., $t_{i+1} - t_i = t_{j+1} - t_j$, for $1 \leq i, j < C$.

For example, a driver's behavior can be captured by a 2-dimensional time series $T = \langle S_1, S_2, \ldots, S_C \rangle$, where $S_i = (s_i^1, s_i^2)$ represents two features—the speed s_i^1 and the driving direction s_i^2 of the driver at time t_i.

B. Problem Definition

Given a k-dimensional time series $T = \langle S_1, S_2, \ldots, S_C \rangle$, we aim at assigning vector S_i an *outlier score* such that the higher the outlier score S_i has, the more likely it is that S_i is an outlier.

Based on the outlier scores, we can rank the vectors in a time series such that the outlier vectors are ranked higher. Alternatively, we can consider the top $\alpha\%$ (e.g., 5%) of the vectors, according to their outlier scores, as outliers.

For example, given a driver's time series, the outlier vectors may represent the occasions where the driver behaves differently from other occasions, e.g., when the driver encounters a dangerous situation.

C. Autoencoders

Since we intend to use autoencoders to define outlier scores, we first present the basic idea of an autoencoder.

An autoencoder [7] aims at reproducing an input m-dimensional vector $X = (x_1, x_2, \ldots, x_m)$ by an output m-dimensional vector $\hat{X} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m)$. An autoencoder comprises two components—an encoder and a decoder. The encoder maps an input vector X into an intermediate n-dimensional vector F, and the decoder maps F to an output vector $\hat{X}$ that is expected to approximate the input vector X. Note that vectors X and $\hat{X}$ have the same dimension, i.e., m, which may be different from, and often larger than, the number of dimensions in F. This means that it is often the case that $m \neq n$ and $m > n$.

Formally, the encoder and the decoder can be defined as functions ϕ and ψ, as shown in Equation 1.

$$\begin{aligned} \text{Encoder } \phi &: \mathbb{R}^m \to \mathbb{R}^n \\ \text{Decoder } \psi &: \mathbb{R}^n \to \mathbb{R}^m \end{aligned} \tag{1}$$

Then, the objective of an autoencoder is described as follows.

$$\operatorname*{argmin}_{\phi,\psi} ||X - \psi(\phi(X))||_2^2$$

This means that we aim at identifying appropriate functions ϕ and ψ such that the difference between input vector X and output vector $\hat{X} = \psi(\phi(X))$ is minimized.

Neural networks are often employed to construct autoencoders. For example, Figure 1 shows the architecture of an autoencoder using a neural network with one hidden layer.

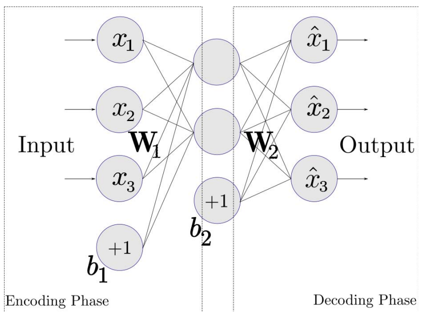


Fig. 1: Neural Network based Autoencoder

In the encoding phase, the neural network takes as input a vector $X \in \mathbb{R}^m$ and maps it to $F \in \mathbb{R}^n$, where $n < m$ using the function shown in Equation 2.

$$F = \sigma_1(\mathbf{W}_1 X + B_1) \tag{2}$$

Here, $\mathbf{W}_1 \in \mathbb{R}^{n \times m}$ is a weight matrix, $B_1 \in \mathbb{R}^n$ is a bias vector, and σ_1 is an activation function, e.g., the *ReLU* or *sigmoid* function [8].

In the decoding phase the neural network maps $F \in \mathbb{R}^n$ to $\hat{X} \in \mathbb{R}^m$ with the aim of reconstructing X through the function shown in Equation 3.

$$\hat{X} = \sigma_2(\mathbf{W}_2 F + B_2) \tag{3}$$

As in Equation 2, $\mathbf{W}_2 \in \mathbb{R}^{m \times n}$ is a weight matrix, $B_2 \in \mathbb{R}^n$ is a bias vector, and σ_2 is an activation function.

To measure if the reconstructed $\hat{X}$ is similar with the original input vector X, we compute the reconstruction error between X and $\hat{X}$ using the L_2 distance as shown in Equation 4.

$$E(X, \hat{X}) = ||X - \hat{X}||_2^2 = \sum_{i=1}^{m} (x_i - \hat{x}_i)^2 \tag{4}$$

Based on the above, our aim is to minimize the reconstruction error $E(X, \hat{X})$. This amounts to learning appropriate weight matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ and bias vectors B_1 and B_2 such that $E(X, \hat{X})$ is minimized. Existing learning algorithms, e.g., gradient descent and back-propagation, can be employed to learn the weight matrices and bias vectors.

D. Outlier Detection using Autoencoders

An autoencoder can be regarded as a dimensionality reduction method [9]. In particular, input vector X that is represented in a large feature space, i.e., using m features, is reduced to a vector F that is represented in a small feature space, i.e., using n features, where $m > n$.

The idea of using autoencoder to detect outliers comes from empirical observations that outliers are much more difficult to represent than inliers when using a smaller feature space [10]. Intuitively, an autoencoder tries to reduce the dimensionality of the input data by mapping the input data into a smaller feature space, but it is unable to reconstruct all the data well because the hidden layers act like an information bottleneck [11], due to their limited feature space. An autoencoder tries to reconstruct the normal data, i.e., inliers, well, but it is often unable to reconstruct the abnormal data, i.e., outliers, well. Thus, when an autoencoder meets outliers, the reconstruction errors are large.

Based on the above, the outlier score $E(S_i, \hat{S}_i)$ of a vector S_i in a multidimensional time series $T = \langle S_1, S_2, \ldots, S_C \rangle$ is defined as the reconstruction error of the vector.

III. Framework and Methodology

We propose a framework that utilizes autoencoders to identify outliers. We first give an overview of the framework and then provide specifics on each module in the framework.

A. Framework Overview

Figure 2 shows an overview of the framework. The input is a multidimensional time series. Before detecting outliers, we use a *Time Series Enrichment* module to enrich the features of each vector in the multidimensional time series. In particular, we generate a wide variety of statistical features to be associated with each vector in the time series. We call a time series that also includes the statistical features in its vectors an *enriched time series*, and we call an original input time series a *raw time series*.

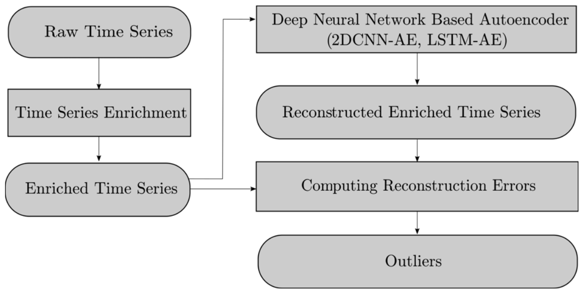

Raw Time Series
Time Series Enrichment
Enriched Time Series
Deep Neural Network Based Autoencoder (2DCNN-AE, LSTM-AE)
Reconstructed Enriched Time Series
Computing Reconstruction Errors
Outliers

Fig. 2: Basic Framework

Next, the enriched time series are fed into the core module of the framework—the one with the deep neural network based autoencoders. In particular, we propose a 2D convolutional neural network based autoencoder (2DCNN-AE), and a long short term memory based autoencoder (LSTM-AE). Then, the reconstructed enriched time series are passed to a module that also received the original enriched time series and that then computes the reconstruction errors for each vector in the enriched time series, which can be used for identifying outliers.

In addition, an optional embedding module can be plugged into the autoencoders to take into account contextual information (e.g., road types for driving behavior time series), which helps reconstruct the input time series and thus also helps improve the accuracy of identifying outliers. This contributes to an advanced framework, which is discussed in Section III-D.

B. Enriching Raw Time Series

The vectors in a time series are often inter-dependent rather than being independent. Following a recent study [12], we employ sliding windows to account for dependencies and compute statistical features in each window to obtain the so-called enriched time series. In the enriched time series, the feature space is much larger than in the raw time series, which helps the autoencoder identify the most representative features for a small space.

We follow a two-step procedure to enrich a raw time series. **Step 1:** In the first step, we consider a sliding window with size b ($b > 1$) where two consecutive windows overlap by $\lfloor \frac{b}{2} \rfloor$. Thus, for a raw time series with C vectors, we get $C' = \lfloor \frac{2 \cdot C - b}{b} \rfloor$ windows. For example, Figure 3 shows a time series with $C = 14$ vectors and window size $b = 4$. Then, the first window considers interval $[t_1, t_4]$, the second window considers interval $[t_3, t_6]$, the third window considers interval $[t_5, t_8]$, etc. In total, there are $C' = 6$ windows.

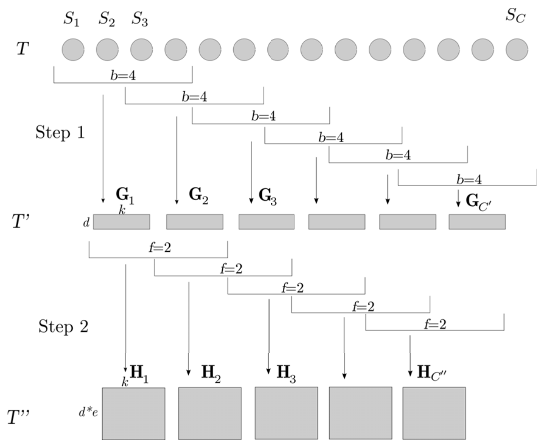

S_1 S_2 S_3
S_C
T
b=4
Step 1
$\mathbf{G}_1$ $\mathbf{G}_2$ $\mathbf{G}_3$ $\mathbf{G}_{C'}$
k
d
T'
f=2
Step 2
$\mathbf{H}_1$ $\mathbf{H}_2$ $\mathbf{H}_3$ $\mathbf{H}_{C''}$
$d{*}e$
T''

Fig. 3: Two Steps of Time Series Enrichment

For each window, we compute d ***derived features*** for each feature in the raw time series. Specifically, we consider $d = 2$ derived features—*norm (NOR)* and *difference of norm (DON)*. Next, we show how to compute the derived features.

Suppose that window wd_i represents interval $[t_i,\ t_{i+b-1}]$ and thus contains the following b vectors from the raw time series.

$$wd_i = \langle S_i, S_{i+1}, \ldots, S_{i+b-1} \rangle$$

Next, we compute *NOR* and *DON* for each feature in each vector. The norm *NOR* of the j-th feature for window wd_i is the Euclidean length of vector $wd_i^j = \langle s_i^j, s_{i+1}^j, \ldots, s_{i+b-1}^j \rangle$, defined as follows.

$$NOR^j(wd_i) = ||wd_i^j||_2 = \sqrt{(s_i^j)^2 + (s_{i+1}^j)^2 + \ldots + (s_{i+b-1}^j)^2}$$

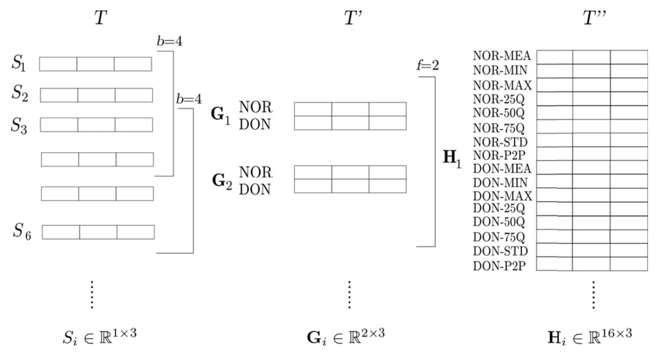


Fig. 4: Feature Space of Enriched Time Series, k=3, b=4, f=2

Then, the difference of norm *DON* of the j-th feature for window wd_i is the difference between the norm of window wd_i and the norm of its previous window wd_{i-1}, where both norms are also for the j-th feature.

$$DON^j(wd_i) = norm^j(wd_i) - norm^j(wd_{i-1})$$

The *norm* feature captures the magnitude of the features in a window, and the *difference of norm* feature captures the temporal change on the magnitude of the features in two consecutive windows, i.e., temporal dependencies.

Having a window size parameter b allows us to choose an appropriate granularity when modeling temporal changes. For example, when $b = 1$, we are able to model the temporal change at every timestamp. The design choice of having overlapping windows makes it possible to smooth the temporal changes.

After the computation, we obtain a new time series

$$T' = \langle \mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_{C'} \rangle,$$

where each $\mathbf{G}_i \in \mathbb{R}^{d \times k}$ is a matrix, where the x-th row corresponds to the x-th derived features. Figure 4 shows an example when considering two derived features and the vectors in raw time series are in a $k = 3$ dimensional feature space, e.g., speed, direction, and acceleration for a driving behavior time series. In particular, the first row corresponds to the norm and the second row corresponds to the difference of norm, and the three columns corresponds to the three different features in the raw time series.

Step 2: In the second step, we consider another sliding window with size f. As in the first step, two consecutive windows overlap by $\lfloor \frac{f}{2} \rfloor$.

For each window, we compute e *statistical features* for each of the derived features in T' that we have computed in the first step. The aim of computing these statistical features is to further capture the extent of changes of the derived features across time.

In particular, suppose that a window wd_i represents an interval $[t_i, t_{i+f-1}]$ and thus contains the following f matrices from time series T'.

$$wd_i = \langle \mathbf{G}_i, \mathbf{G}_{i+1}, \ldots, \mathbf{G}_{i+f-1} \rangle$$

Further, following a recommendation in a recent study [12], we compute $e = 8$ statistical features for each derived feature—*mean (MEA)*, *minimum (MIN)*, *maximum (MAX)*, *25%-quartiles (25Q)*, *50%-quartile (50Q)*, *75%-quartile (75Q)*, *standard deviation (STD)*, and *peak to peak (P2P)*.

In our running example, where we have 2 derived features—*NOR* and *DON*, we compute the above 8 statistical features for each, as also shown in Figure 4.

After the computation, we obtain the enriched time series

$$T'' = \langle \mathbf{H}_1, \mathbf{H}_2, \ldots, \mathbf{H}_{C''} \rangle,$$

where each $\mathbf{H}_i \in \mathbb{R}^{(e \cdot d) \times k}$ is a matrix, as shown in Figure 4.

C. Deep Neural Network based Autoencoders

Based on the enriched time series T'', we present two deep neural network based autoencoders to reconstruct the enriched time series as explained in Section III-A.

1) 2DCNN-AE: A 2D Convolutional Neural Network (2DCNN) is a popular deep learning structure that achieves impressive accuracy for data that can be represented as 2D matrices, e.g., images [13]. 2DCNNs are often used for classifying images, where each training image is associated with a label, often obtained from a human expert, that indicates the class of the image. Based on a set of training images, 2DCNNs are able to recognizes important features for classifying images through the use of convolutional, max-pooling, and sub-sampling layers [14], as shown in Figure 5.

A 2DCNN-AE is a combination of a 2DCNN and an autoencoder. In particular, a 2DCNN-AE has the same architecture as a classical 2DCNN. However, during training, instead of having a label for each input 2D matrix, the 2DCNN-AE employs the input matrix itself as the label. This way, a 2DCNN-AE is able to learn important features such that the input matrices can be reconstructed. The architecture of a 2DCNN-AE is shown in Figure 5.

Recall that in an enrich time series $T'' = \langle \mathbf{H}_1, \mathbf{H}_2, \ldots, \mathbf{H}_{C''} \rangle$, each element $\mathbf{H}_i$ is a $\mathbb{R}^{(d \cdot e) \times k}$ matrix. Since each $\mathbf{H}_i$ is a 2D matrix, we can use each matrix $\mathbf{H}_i$ directly as training input and $\mathbf{H}_i$ itself as the corresponding label to train a 2DCNN-AE.

In the encoding phase, we use multiple convolution and max-pooling layers. In a convolution layer, we use FI filters to convolute the input matrix $\mathbf{H}_i$ using Equation 5.

$$\mathbf{J}_i^k = \sigma(\mathbf{H}_i * \mathbf{W}^k + B^k) \tag{5}$$

Here, $\mathbf{W}^k$ is the k-th filter, which is often a small, e.g., 2×2, matrix with randomly initialized values; B^k is a bias; symbol $*$ denotes the 2D convolution operation; and σ is an activation function, where we use *sigmoid* for the last layer and *reLU* for all other layers. $\mathbf{J}_i^k$ is the output of the activation function, which is also called a feature map at k indexes of $\mathbf{H}_i$. Since we have FI filters in total, we obtain $FI \cdot C''$ feature maps as the output of the convolution layer.

Following the principles of convolutional neural networks, after the convolution layer, we use a max-pooling layer to

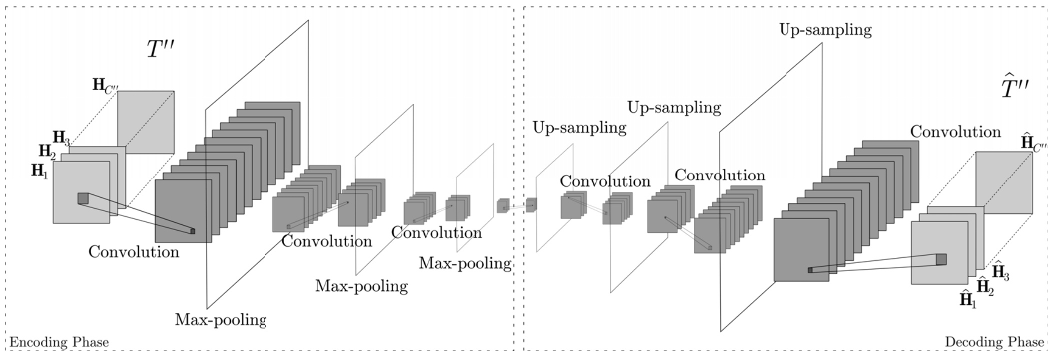


Fig. 5: 2D Convolutional Neural Network based Autoencoder (2DCNN-AE)

condense the data. Figure 5 shows that we use in total 3 convolution and max-pooling layers to encode the enriched time series. Max-pooling is a down-sampling method that usually takes the maximum values over non overlapping subregions of the feature map. The potential benefit of using max-pooling is to achieve improvements of the selectivity of different filters [15].

In the decoding phase, we use multiple convolution and up-sampling layers in a manner similar to what was done in the encoding phase. In contrast to max-pooling (i.e., down-sampling), up-sampling is an interpolation method that replicates values in feature maps. Thus, up-sampling replicates the maximum values to a larger region. In particular, the reconstruction is achieved using Equation 6.

$$\hat{\mathbf{H}}_i = \sigma(\sum_{k=1}^{|FI|} \mathbf{J}_i^k * \tilde{\mathbf{W}}^k + B^k) \tag{6}$$

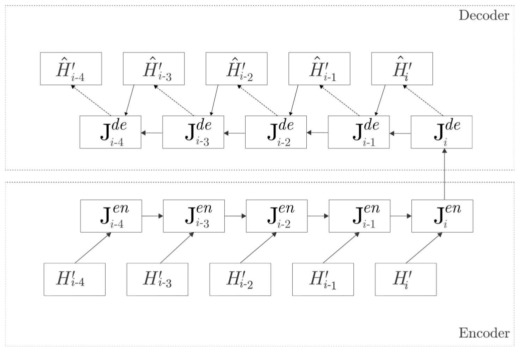
Here , $\mathbf{J}_i^k$ is the k-th feature map, $\tilde{\mathbf{W}}^k$ is obtained by flipping $\mathbf{W}^k$, and B^k is a bias.

2) LSTM-AE: Although each matrix $\mathbf{H}_i$ in an enriched time series already incorporates temporal changes of the raw time series, it is beneficial to also model the temporal dependencies among different matrices $\mathbf{H}_i$ in the enriched time series explicitly in an autoencoder. However, 2DCNN-AE is not able to model the temporal dependencies among different matrices $\mathbf{H}_i$ in the enriched time series. Rather, 2DCNN-AE considers each matrix in an enriched time series independently during training.

To overcome this limitation, we use a long short term memory (LSTM) neural network [16] to capture the temporal dependencies of the matrices in the enriched time series. LSTM augments a traditional neural network with recurrent connections between layers. These connections span adjacent timestamps and model explicitly the changes over time.

In particular, LSTM-AE is a hierarchical model that consists of an encoder and a decoder, each of which is an LSTM. Figure 6 shows the architecture of the LSTM-AE.

Given an enriched time series T'', we first transform each matrix $\mathbf{H}_i$ in T'' into a vector H'_i. In particular, given a $\mathbb{R}^{(d\cdot e)\times k}$ matrix $\mathbf{H}_i$, we concatenate the rows in $\mathbf{H}_i$ to obtain



Decoder
$\hat{H}'_{i\text{-}4}$
$\hat{H}'_{i\text{-}3}$
$\hat{H}'_{i\text{-}2}$
$\hat{H}'_{i\text{-}1}$
$\hat{H}'_i$
$\mathbf{J}^{de}_{i\text{-}4}$
$\mathbf{J}^{de}_{i\text{-}3}$
$\mathbf{J}^{de}_{i\text{-}2}$
$\mathbf{J}^{de}_{i\text{-}1}$
$\mathbf{J}^{de}_i$
$\mathbf{J}^{en}_{i\text{-}4}$
$\mathbf{J}^{en}_{i\text{-}3}$
$\mathbf{J}^{en}_{i\text{-}2}$
$\mathbf{J}^{en}_{i\text{-}1}$
$\mathbf{J}^{en}_i$
$H'_{i\text{-}4}$
$H'_{i\text{-}3}$
$H'_{i\text{-}2}$
$H'_{i\text{-}1}$
H'_i
Encoder

Fig. 6: LSTM-AE with Input Subsequences of Length $l = 5$

a vector H'_i of size $d \cdot e \cdot k$. Figure 7 shows the transformation from $\mathbf{H}_i$ to H'_i.

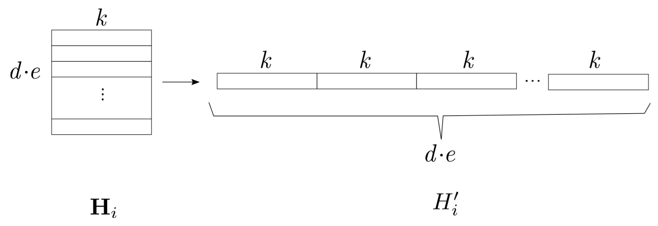


Fig. 7: Unfolding a 2D Matrix $\mathbf{H}_1$ to a 1D Vector H'_1

The LSTM-AE takes as input a subsequence of T'' with fixed length l to perform training. Specifically, each subsequence of length l from T'', e.g., $\langle H'_i, H'_{i-1}, \ldots, H'_{i-l+1}\rangle$, is fed into LSTM-AE as training data. Following the principle of autoencoders, the same subsequence is employed as the corresponding label. Figure 6 shows an example where subsequence of length $l = 5$ are employed.

Specifically, in the encoder phase, a hidden state $\mathbf{J}_k^{en}$ at time k, where $i-l+1 < k \le i$, is dependent on the input vector at time k, i.e., H'_k, and the hidden state at the previous time, i.e., $\mathbf{J}_{k-1}^{en}$. The last hidden state $\mathbf{J}_i^{en}$ is replicated into the decoder, i.e., $\mathbf{J}_i^{de} = \mathbf{J}_i^{en}$. In the decoder phase, each hidden state $\mathbf{J}_k^{de}$ produces an reconstructed vector $\hat{H}'_k$. Meanwhile, a hidden state $\mathbf{J}_k^{de}$ is dependent on the hidden state at the next time

$\mathbf{J}^{de}_{k+1}$ and the reconstructed vector at the next time $\hat{H}'_{k+1}$. The specifics of computing encoder and decoder hidden states and the reconstructed vectors follow a traditional LSTM [17].

D. Embedding Contextual Information

The changes in a time series may be due to contextual changes. For example, a vehicle suddenly increasing its speed sharply may be due to moving from a residential road to a highway, which does not represent a dangerous behavior, i.e., an outlier. As another example, when driving in rainy conditions, drivers may drive at lower speeds than under normal conditions. Thus, taking contextual information [18], such as road types and weather, into account when identifying outliers may improve the outlier detection performance.

Contextual information is often described using categorical values. However, neural networks often treat inputs as numerical values. To contend with this, one-hot representation [19] is often employed to encode categorical contexts. In particular, the one-hot representation encodes each category into a bit-vector. If a context has four categories the one-hot representation of the 2-nd category is (0, 1, 0, 0). The one-hot representation has two limitations. First, it does not scale well w.r.t. the number of categories. When the number of categories is large, the computation cost of one-hot representation increase significantly. Second, it does not capture similarities between categories.

The embedding method [19] resolves the limitations of the one-hot representation using an additional mapping operation, where each category is transformed and represented in a low-dimensional space. The embedding layer is a matrix of weights $\mathbf{W}_{embeding} \in \mathbf{R}^{i\times o}$ where i is the total number of categories and o is the number of dimensions in the low dimensional space. Given a specific category v, we use $onehot(v) \in R^{1\times i}$ to denote the one-hot representation of v. Then, we use $embedded(v) = onehot(v) \times \mathbf{W}$ to represent the embedded representation of the category. Normally, we set $o \ll i$ such that when having many categories, similar categories eventually have the same embedded representation and such that the embedded representation is much smaller than the one-hot representation. This solves the two limitations of the one-hot representation.

We plug the embedding method into our basic framework to enable an advanced framework, as shown in Figure 8. The input is not only an enriched time series, but also the corresponding contextual information. In particular, the enriched time series are still fed into the *deep neural network based autoencoders*. This module outputs *reconstructed enriched time series*. The contextual information is first represented using the one-hot representation and is then fed into an *embedding layer*. The output from the embedding layer is an *embedded context*. Then, we concatenate with the reconstructed enriched time series with the embedding context and then feed them into a *fully connected layer* [8] that produces the final output. Following the principle of autoencoders, we use the enriched time series along with their contextual information as the label for the input data.

We do not need to train the advanced model from scratch. We can start from the basic model that has already been trained. Experimental results show that reusing parameters from the basic model makes the training more efficient.

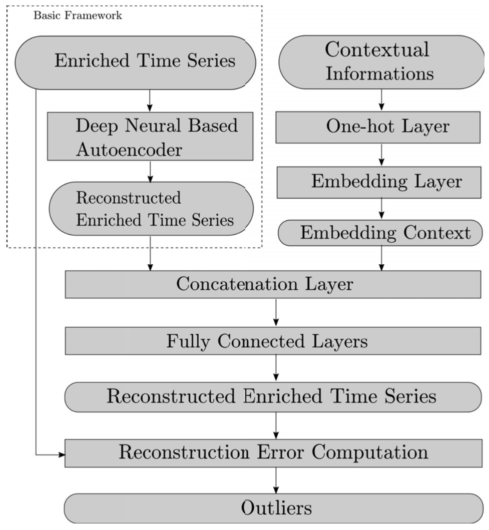


Fig. 8: Advanced Framework

IV. Experiments

We report on a comprehensive empirical study using different real-world time series.

A. Experimental Setup

1) Data Sets: We use three multidimensional time series data sets—UAH-DriveSet [20], Numenta Anomaly Benchmark (NAB) [21], and S5 Yahoo! Anomaly Detection Data Set (S5) [22].

UAH-DriveSet contains 6 driving behavior time series datasets from 6 drivers, where each driver has been asked to drive in a normal, a drowsy, or an aggressive manner. The time series are collected by multi-sensor devices in the Naturalistic Driving Study [23]. Each time series is 2-dimensional, capturing speed and direction. The frequency is 1 Hz, i.e., there is one 2-dimensional vector per second. The length of a time series ranges from 420 to 1080. Each vector is labeled with an aggressiveness score from expert annotaters. We only use data reflecting normal driving behavior. Thus, we use the aggressiveness scores as ground truth outlier scores. The higher the aggressiveness scores, the more likely it is that the corresponding vectors are outliers. In addition, each vector is associated with a road type label, indicating the specific type of the road that the vehicle was on. The road type labels are used as context in the advanced framework.

NAB contains 58 univariate time series, with 1,000 to 22,000 vectors per time series. The time series are collected from a wide variety of applications, ranging from network traffic monitoring and CPU utilization in cloud services to industrial machine operation monitoring and social media activities. For each time series, each vector is associated with a manually supplied Boolean outlier label, indicating whether the vector is an outlier.

S5 contains 371 time series that are organized into four groups, where two groups are for outlier detection and the remaining two groups are for change-point detection. We only use the 167 time series in the two outlier detection groups. These time series are univariate and have lengths that range from 1482 to 2922. Similar to *NAB*, each vector in the time series has a Boolean outlier label.

2) Methods: We consider two non-deep learning based baselines—Local Outlier Factor (LOF) [24] and One-Class Support Vector Machines (OC-SVM) [25]. Note that we cannot embed contextual information into LOF and OC-SVM because LOF and OC-SVM are unable to deal with categorical data types such as road types and traffic types. Both baselines are state-of-the-art outlier detection algorithms that can be used for raw time series data [26]. The settings for both baselines are set to default of Scikit-learn [27]. Thus, both methods are able to return an outlier score for each vector in a time series.

In addition to the two baselines, we consider the following method proposed in the paper: *2DCNN-AE*, and *LSTM-AE* are the proposed methods on enriched time series. Further, for *LSTM-AE*, we also consider two variants. First, *LSTM-AE-RAW* is LSTM-AE using raw time series (i.e., without using the time series enrichment module) and *LSTM-AE-ADV* is the LSTM-AE with contextual information, i.e., the advanced framework.

3) Evaluation Metrics: We consider two different sets of evaluation metrics. First, we consider the ground truth, Boolean outlier labels. Since the proposed methods and baselines are able to identify outlier vectors, we are able to measure *precision* [8], *recall* [8], and *F1-score* [8] against the ground truth, Boolean outlier labels. The larger the precision, recall, and F1-score are, the more accurate the given outlier detection method is.

Second, we use the ground truth outlier scores to obtain a ground truth ranking. Since the proposed methods and baselines are able to compute outlier scores, we are able to obtain another ranking. Then we employ *Kendall's tau* [8], to measure the consistency between the two rankings. In particular, we choose the top-30 outliers found by each method and compare with top-30 outliers based on the ground truth outlier scores. *Kendall's tau* returns a value between -1 and 1, where 1 indicates that the two rankings are fully consistent and -1 indicates that two rankings are totally opposite.

4) Implementation Details: The proposed methods are implemented in Python 3.5 with the help of deep learning library Theano 0.9.0 [28]. The baseline methods, i.e., LOF and One-class SVM, are also implemented in Python 3.5 using the Scikit-learn library [27]. Experiments are performed on a Linux server with 64 AMD CPU Cores and 512GB RAM.

5) Other Settings: To prevent over-fitting, Dropout method [28] is employed with probability 0.2 after the encoding component. We employ the Adaptive Moment Estimation (Adam) method [29] to train all our models. Adam is a robust mini-batch gradient-based optimization technique that suits our problem well. In the paper, we fix the batch size at 100. The learning rate and epochs are set to 0.01 and 500, respectively. For 2DCNN-AE, we use sigmoid function as the activation function for the last layer and relu as the activation function for the remaining layers. For LSTM-AE-RAW and LSTM-AE, we use tanh as the activation function.

We vary the two window sizes b and f used in the two-step time series enrichment module according to Table I.

TABLE I: Window Sizes

1	2	3	4	5	6	7	8	9
b=2, f=2	b=2, f=2	b=6, f=2	b=8, f=4	b=8, f=6	b=10, f=6	b=10, f=8	b=14, f=8	b=14, f=10

B. Experimental Results

1) UAH-DriveSet: We first report the average performance on the entire UAH-Driveset data set using average *Precision*, *Recall*, and *F1*, as shown in Table II. OC-SVM has the largest precision, but it has very low recall. Thus, OC-SVM also has a low F1 score. In contrast, the deep neural network based methods have precision comparable to that of OC-SVM and LOF, but have much higher recall, which also yields a high F1. In particular, LSTM-AE-ADV achieves the highest F1 score. This demonstrates that the proposed deep neural network based autoencoders are effective at identifying outliers.

In addition, LSTM-AE and LSTM-AE-ADV achieve higher precision, recall, and F1 than does LSTM-AE-RAW. This suggests that the proposed time series enrichment is effective.

Next, we report the average performance on the entire UAH-Driveset using average *Kendall's tau*, as shown in Table IV. Recall that a large Kendall's tau indicates that two rankings are more consistent. Table IV shows that all the deep neural network based methods have positive Kendall's tau values, while the two baselines have negative Kendall's tau values. This offers evidence that the proposed methods are most accurate. In addition, the methods based using enriched time series have larger Kendall's tau values compared to the method on raw time series. This also justifies the design choice of using enriched time series.

Next, we take a closer look at the UAH-Driveset by checking the accuracy for individual drivers, i.e., D_1, D_2, D_3, D_4, D_5, and D_6. For each driver's time series, we report

TABLE II: Average Precision, Recall, and F1, UAH-Driveset

				Basic Framework		Advaced Framework
	OC-SVM	LOF	LSTM-AE-RAW	2DCNN-AE	LSTM-AE	LSTM-AE-ADV
Precision	**0.491**	0.447	0.438	0.475	0.477	0.479
Recall	0.476	0.862	0.863	0.786	0.865	**0.957**
F1	0.483	0.588	0.581	0.592	0.615	**0.638**

TABLE III: Precision, Recall, and F1, UAH-Driveset

					Basic Framework		Advaced Framework
		OC-SVM	LOF	LSTM-AE-RAW	2DCNN-AE	LSTM-AE	LSTM-AE-ADV
D1	Precision	**0.976**	0.866	0.853	0.839	0.861	0.865
	Recall	0.567	0.903	0.942	0.794	0.918	**0.949**
	F1	0.717	0.884	0.895	0.816	0.889	**0.905**
D2	Precision	0.126	0.119	0.107	**0.200**	0.195	0.197
	Recall	0.286	0.733	0.715	0.745	0.740	**1.000**
	F1	0.175	0.204	0.187	0.315	0.309	**0.330**
D3	Precision	0.566	0.557	0.497	**0.625**	0.612	0.612
	Recall	0.453	0.843	0.750	0.807	0.862	**0.912**
	F1	0.503	0.671	0.598	0.703	0.716	**0.732**
D4	Precision	0.366	0.355	0.367	0.347	0.348	**0.380**
	Recall	0.461	0.884	**0.961**	0.708	0.902	0.953
	F1	0.408	0.507	0.531	0.466	0.502	**0.543**
D5	Precision	**0.380**	0.283	0.310	0.315	0.321	0.296
	Recall	0.583	0.909	0.902	0.797	0.857	**0.964**
	F1	0.460	0.431	**0.461**	0.451	0.441	0.453
D6	Precision	**0.531**	0.500	0.495	0.523	0.523	0.523
	Recall	0.503	0.902	0.905	0.865	0.913	**0.966**
	F1	0.517	0.643	0.604	0.652	0.665	**0.679**

TABLE IV: Average Kendall's tau, UAH-Driveset

				Basic Framework		Advaced Framework
	OC-SVM	LOF	LSTM-AE-RAW	2DCNN-AE	LSTM-AE	LSTM-AE-ADV
Kendall's tau	-0.048	-0.025	0.026	0.128	**0.139**	0.108

the precision, recall, and F1 in Tabel III and Kendall's tau in Figure 9.

Table III suggests that, although there are small variations across the time series from different drivers, we can still conclude the following: (1) LSTM-AE-ADV tends to be the best method, as it often achieves both high precision and recall, which produces the highest F1; (2) deep neural network based outlier detection is more accurate than the two baselines; (3) enrichment time series helps deep neural network based outlier detection achieve better performance.

In Figure 9, we observe larger variations across the different time series. For drivers D1 and D3, LSTM-AE-RAW achieves the highest Kendall's tau. For drivers D2 and D5, LSTM-AE-ADV achieves the highest Kendall's tau. LSTM-AE achieves the highest kendall's tau for drivers D4 and D6. Overall, the deep neural network based methods achieve higher Kendall's tau values than do the two baselines, indicating that the proposed methods are effective.

Next, we vary the window sizes b and f to understand their effect. We report the average F1 and Kendall's tau values

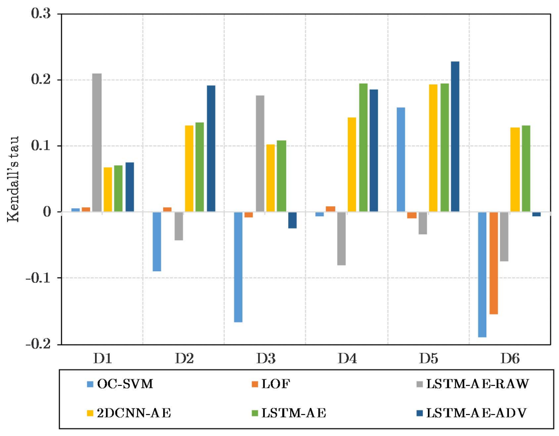


Fig. 9: Kendall's tau, UAH-Driveset

TABLE V: Average Precision, Recall, and F1, NAB (top) and S5 (bottom)

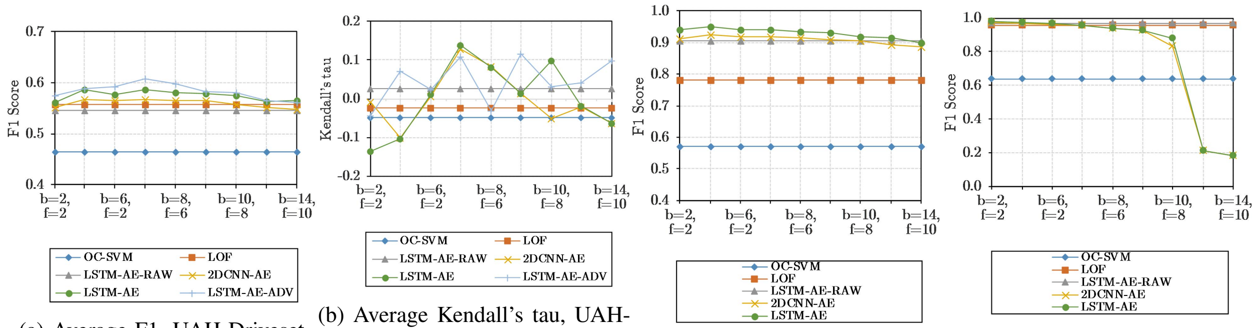
				Basic Framework	
	OC-SVM	LOF	LSTM-AE-RAW	2DCNN-AE	LSTM-AE
Precision	0.828	0.829	0.883	0.891	**0.908**
Recall	0.446	0.722	0.931	0.962	**0.988**
F1	0.580	0.772	0.906	0.925	**0.946**
Precision	0.993	0.983	0.994	0.995	**0.997**
Recall	0.509	0.966	0.983	0.982	**0.985**
F1	0.673	0.974	0.988	0.988	**0.991**

in Figures 10a and 10b. The curves for OC-SVM, LOF, and LSTM-AE-RAW are flat since they use raw time series, so changing the window sizes has no effect on these three methods. Figure 10a suggests that the setting with b=8 and f=4 gives the highest F1 score, so this is used as the default window sizes settings. Although the results on Kendall's tau show high variation, Figure 10b suggests that the same window size setting with b=8 and f=4 gives the highest Kendall's tau scores.

2) NAB and S5: Since the *NAB* and *S5* data sets do not contain additional contextual information, we cannot use the LSTM-AE-ADV method to embed contexts and omit it in the following experiments. Further, since both data sets only have Boolean ground truth outlier labels, not ground truth outlier scores, we cannot evaluate the accuracy using Kendall's tau. Thus, we report precision, recall, and F1.

We first report the average performance for *NAB* and *S5* in Table V. LSTM-AE is the best in terms of precision, recall, and F1 for both data sets. The deep neural networks based methods achieve better precision, recall, and F1 compared to the two baselines. Further, enrichment of the time series improves the accuracy of the deep neural networks based methods. Specifically, LSTM-AE performs better than LSTM-AE-RAW.

Next, we test the effect of window sizes b and f on the two



(a) Average F1, UAH-Driveset (b) Average Kendall's tau, UAH-Driveset (c) Average F1, NAB (d) Average F1, S5

Fig. 10: Effect of b and f

data sets. On *NAB*, the accuracy of the proposed methods is non sensitive to b and f—the proposed methods outperform the two baselines in all settings. On *S5*, the proposed methods do not perform well for large window sizes, due to the fact that large windows mask the temporal changes during a short time, which adversely affects accuracy.

Finally, we note that although the proposed deep-learning based framework is slower than non-deep learning baselines such as LOF and OC-SVM (i.e., typically by a factor from 100 to 200), it is possible to employ different parallelization strategies [30], [31] to improve efficiency.

V. Related Work

Outlier Detection on Time Series Data: Studies of outlier detection in time series can be organized into two categories. Studies in the first category identify outlier time series from a set of time series. Studies in the second category identify outlier points in single time series. This paper concerns the outlier detection in the second category.

We first consider studies in the first category. In the Nearest Neighbor based method (NNB) [32], the distance of one time series to its k nearest neighbors is the anomaly score of the time series. In the Clustering Based method (CB) [33], time series are clustered into a fixed number of clusters using a clustering method such as the CLARA k-medoids algorithm [34]. Then, the distance of a time series to the medoid of the closest cluster is used as the outlier score. In window based methods, an original time series is broken into overlapping subsequences, and the anomaly score of the original time series is computed from the scores of the subsequences. Cabrera et al. [35] use a dictionary that contains normal subsequences and one that contains abnormal subsequences. A time series is an outlier if its subsequences do not occur in the normal dictionary or occur in the abnormal dictionary.

Next, we review studies in the second category. Local outlier factor (LOF) [24] is a state-of-the-art method that is also based on clustering. In particular, LOF considers the local density of clusters and is able to detect local outliers effectively. Similarly, One-Class SVM [25] is used to detect outliers by discriminating normal vs. abnormal data by considering normal data as a single class. In addition, time series prediction models, e.g., ARMA, can be used for outlier detection by measuring the differences between predicted and actual values [36]. Large differences indicate possible outliers. Galeano et al. [37] propose a vector ARMA (VARMA) model that projects a multivariate time series into many univariate time series, which is more efficient when detecting outliers. Keogh et al. [38] propose a method that transforms time series into sequences of characters and then compute the self-matching score between substrings to detect outliers.

Deep Neural Networks: Deep neural networks have been studied extensively for solving supervised learning tasks such as classification [13] and prediction [39]. However, only few studies consider deep neural networks for resolving unsupervised learning tasks such as outlier detection. Xia et al. [11] study removing outliers from noisy image data. In particular, a deep autoencoder separates inliers from outliers according to reconstruction errors. This study shows that deep neural networks are capable of being more accurate than traditional methods such as LOF and One-class SVM. Lu et al. [40] propose a deep neural network to detect outliers in video, which outperforms existing, non-deep learning based methods. They propose an autoencoder that comprises two components—a traditional feedforward neural network to extract local features, and an LSTM neural network to compress sequential features. They apply the model to raw time series instead of enriched time series. To the best of our knowledge, our paper is the first study that applies deep neural networks to enriched time series to perform outlier detection in time series, this way achieving higher accuracy than deep neural networks applied to raw time series and non-deep learning outlier detection methods.

VI. Conclusion and Future Work

We propose a deep learning based framework for outlier detection on multidimensional time series. The framework represents the first attempt to use the combination of deep neural networks and enriched time series for outlier detection. First, we propose a method for enrichment of the feature spaces of raw time series. Second, two different deep learning approaches, 2D Convolutional Autoencoder, and Long-Short Term Memory Autoencoder, are proposed to detect outliers using the enriched time series. In addition, an advanced framework that is able to exploit contextual information is proposed. Finally, we conduct comprehensive experiments on three real-world data sets to investigate the accuracy of the proposed methods.

As future work, it is of interest to combine time series approximation method into our framework to reduce the runtime for learning the autoencoders. Further, it is of interest to study time series derived from GPS data [41]–[43] and study how outlier detections on such GPS based time series can be integrated with data-intensive routing [44]–[46] to improve routing qualities [47], [48].

ACKNOWLEDGMENTS

This research was supported in part by a grant from the Obel Family Foundation and by the DiCyPS project.

REFERENCES

[1] C. Guo, C. S. Jensen, and B. Yang, "Towards total traffic awareness," *ACM SIGMOD Record*, vol. 43, no. 3, pp. 18–23, 2014.

[2] Z. Ding, B. Yang, Y. Chi, and L. Guo, "Enabling smart transportation systems: A parallel spatio-temporal database approach," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1377–1391, 2016.

[3] N. Agerholm, R. Jensen, and C. S. Andersen, "Identification of hazardous road locations on the basis of jerks," in *ICTCT*, 2015.

[4] J. Dai, B. Yang, C. Guo, and Z. Ding, "Personalized route recommendation using big trajectory data," in *ICDE*, 2015, pp. 543–554.

[5] J. Hu, B. Yang, C. Guo, and C. S. Jensen, "Risk-aware path selection with time-varying, uncertain travel costs: a time series approach," *VLDB J.*, vol. 27, no. 2, pp. 179–200, 2018.

[6] Z. Ding, B. Yang, R. H. Güting, and Y. Li, "Network-matched trajectory-based moving-object database: Models and applications," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 4, pp. 1918–1928, 2015.

[7] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[8] C. Sammut and G. I. Webb, Eds., *Encyclopedia of Machine Learning and Data Mining*. Springer, 2017.

[9] M. Leyli-Abadi, L. Labiod, and M. Nadif, "Denoising autoencoder as an effective dimensionality reduction and clustering of text data," in *PAKDD 2017*, pp. 801–813.

[10] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *MLSDA 2014*, pp. 4–11.

[11] Y. Xia, X. Cao, F. Wen, G. Hua, and J. Sun, "Learning discriminative reconstructions for unsupervised outlier removal," in *ICCV 2015*, pp. 1511–1519.

[12] W. Dong, T. Yuan, K. Yang, C. Li, and S. Zhang, "Autoencoder regularized network for driving style representation learning," in *IJCAI 2017*, pp. 1603–1609.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS 2012*, pp. 1106–1114.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR 2015*, pp. 1–9.

[15] D. Scherer, A. C. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *ICANN 2010*, pp. 92–101.

[16] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "LSTM-based encoder-decoder for multi-sensor anomaly detection," *CoRR*, vol. abs/1607.00148, 2016.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] B. Yang, C. Guo, Y. Ma, and C. S. Jensen, "Toward personalized, context-aware routing," *VLDB Journal*, vol. 24, no. 2, pp. 297–318, 2015.

[19] C. Guo and F. Berkhahn, "Entity embeddings of categorical variables," *CoRR*, vol. abs/1604.06737, 2016.

[20] E. Romera, L. M. Bergasa, and R. Arroyo, "Need data for driver behaviour analysis? Presenting the public UAH-DriveSet," in *ITSC 2016*, pp. 387–392.

[21] A. Lavin and S. Ahmad, "Evaluating real-time anomaly detection algorithms - the numenta anomaly benchmark," *CoRR*, vol. abs/1510.03336, 2015.

[22] M. Thill, W. Konen, and T. Bäck, "Online anomaly detection on the webscope S5 dataset: A comparative study," in *EAIS 2017*, pp. 1–8.

[23] A. Bender, J. R. Ward, S. Worrall, M. L. Moreyra, S. G. Konrad, F. R. Masson, and E. M. Nebot, "A flexible system architecture for acquisition and storage of naturalistic driving data," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 6, pp. 1748–1761, 2016.

[24] M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," in *ACM SIGMOD 2000*, pp. 93–104.

[25] L. M. Manevitz and M. Yousef, "One-Class SVMs for document classification," *JMLR*, vol. 2, pp. 139–154, 2001.

[26] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE TKDE*, vol. 26, no. 9, pp. 2250–2267, 2014.

[27] Scikit-learn Development Team, "Scikit-learn: Machine learning in Python," *JMLR*, vol. 12, pp. 2825–2830, 2011.

[28] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *CoRR*, vol. abs/1605.02688, 2016.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[30] B. Yang, Q. Ma, W. Qian, and A. Zhou, "TRUSTER: trajectory data processing on clusters," in *DASFAA*, 2009, pp. 768–771.

[31] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *ICML*, 2009, pp. 873–880.

[32] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," in *ACM SIGMOD 2000*, pp. 427–438.

[33] K. Sequeira and M. J. Zaki, "ADMIT: anomaly-based data mining for intrusions," in *ACM SIGKDD 2002*, pp. 386–395.

[34] A. McCallum, K. Nigam, and L. H. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching," in *ACM SIGKDD 2000*, pp. 169–178.

[35] J. B. D. Cabrera, L. M. Lewis, and R. K. Mehra, "Detection and classification of intrusions and faults using sequences of system calls," *ACM SIGMOD Record*, vol. 30, no. 4, pp. 25–34, 2001.

[36] Y. Sakurai, Y. Matsubara, and C. Faloutsos, "Mining and forecasting of big time-series data," in *ACM SIGMOD 2015*, pp. 919–922.

[37] P. Galeano, D. Pea, and R. S. Tsay, "Outlier detection in multivariate time series by projection pursuit," *Journal of the American Statistical Association*, vol. 101, no. 474, pp. 654–669, 2006.

[38] E. J. Keogh, J. Lin, and A. W. Fu, "HOT SAX: efficiently finding the most unusual time series subsequence," in *ICDM 2005*, pp. 226–233.

[39] A. Grover, A. Kapoor, and E. Horvitz, "A deep hybrid model for weather forecasting," in *ACM SIGKDD 2015*, pp. 379–386.

[40] W. Lu, Y. Cheng, C. Xiao, S. Chang, S. Huang, B. Liang, and T. S. Huang, "Unsupervised sequential outlier detection with deep architectures," *IEEE Trans. Image Process.*, vol. 26, no. 9, pp. 4321–4330, 2017.

[41] B. Yang, C. Guo, and C. S. Jensen, "Travel cost inference from sparse, spatio-temporally correlated time series using markov models," *PVLDB*, vol. 6, no. 9, pp. 769–780, 2013.

[42] J. Hu, B. Yang, C. S. Jensen, and Y. Ma, "Enabling time-dependent uncertain eco-weights for road networks," *GeoInformatica*, vol. 21, no. 1, pp. 57–88, 2017.

[43] J. Dai, B. Yang, C. Guo, C. S. Jensen, and J. Hu, "Path cost distribution estimation using trajectory data," *PVLDB*, vol. 10, no. 3, pp. 85–96, 2016.

[44] B. Yang, J. Dai, C. Guo, C. S. Jensen, and J. Hu, "PACE: a path-centric paradigm for stochastic path finding," *VLDB J.*, vol. 27, no. 2, pp. 153–178, 2018.

[45] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k optimal sequenced routes," in *ICDE*, 2018, pp. 569–580.

[46] C. Guo, B. Yang, O. Andersen, C. S. Jensen, and K. Torp, "Ecosky: Reducing vehicular environmental impact through eco-routing," in *ICDE*, 2015, pp. 1412–1415.

[47] C. Guo, B. Yang, J. Hu, and C. S. Jensen, "Learning to route with sparse trajectory sets," in *ICDE*, 2018, pp. 1073–1084.

[48] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k shortest paths with diversity," *IEEE TKDE*, vol. 30, no. 3, pp. 488–502, 2018.