DeepVix: Explaining Long Short-Term Memory Network With High Dimensional Time Series Data

Tommy Dang
tommy.dang@ttu.edu
Texas Tech University
Lubbock, Texas, US

Hao Van
hao.van@ttu.edu
Texas Tech University
Lubbock, Texas, US

Huyen Nguyen
huyen.nguyen@ttu.edu
Texas Tech University
Lubbock, Texas, US

Vung Pham
vung.pham@ttu.edu
Texas Tech University
Lubbock, Texas, US

Rattikorn Hewett
rattikorn.hewett@ttu.edu
Texas Tech University
Lubbock, Texas, US

ABSTRACT

Neural networks are known for their enormous predictive capability, leading to vast applications in various domains. However, the explainability of the neural network model stills remains enigmatic, especially when the model comes short in learning a certain pattern or features. In this work, we introduce a visual explainable long-short term memory network framework, which focuses on the interpretability of the model on time series data. The hindrance to the training process is highlighted by the irregular instances throughout the whole architecture, from input to intermediate layers and output. Interactive features support users to customize and rearrange the structure to obtain different network representation and to perform what-if analysis. To evaluate the usefulness of our approach, we demonstrate the application of DeepVix on the dataset of multivariate measurements of a medium-size High-Performance Computing center.

ACM Reference Format:
Tommy Dang, Hao Van, Huyen Nguyen, Vung Pham, and Rattikorn Hewett. 2020. DeepVix: Explaining Long Short-Term Memory Network With High Dimensional Time Series Data. In *The 11th International Conference on Advances in Information Technology (IAIT2020), July 01–03, 2020, Bangkok, Thailand*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The large scale of time series data in terms of sources and their application domains, such as cybersecurity, scientific, social, and financial sectors, brings an excellent possibility for research and practical aspects. Time series data contains the readings, or values, of observed variables collected over time. The observations of a single variable over time make univariate time series. On the other hand, many real-world applications generate multiple variables to have high-quality, reliable, and statistically sound information. These observations of multiple variables over time make multivariate time series. The analysis of univariate time series is thus the simpler, and there are well-known statistical methods to do this.

On the contrary, analyzing time series with multiple channels is more challenging due to the increased complexity and over-fitting problems [26]. The traditional methods can be adapted to analyze the multivariate case. Recent advancements of Artificial Neural Networks (ANNs) prove their strong potential to be effective methods in analyzing univariate and multivariate time series. However, the "black box" nature of ANNs requires thorough understanding before deploying. This paper aims to make the long short-term memory (LSTM) network – a class of neural networks with temporal sequence to be transparent – to the users, especially to the domain experts who might be able to inject their knowledge in the reasoning process and customize the learning algorithm to adapt to new requirements and changes. The contributions of this paper are three-fold:

- We introduce a visual framework, called DeepVix, for visualizing the LSTMs. The visualization and visual encodings provide a summary view of LSTM structure, LSTM flows, and learning process, which can be expanded into detailed views via user interactions with the interface.
- We propose an interactive approach to investigate and explain the underlying rationale of LSTM.
- We demonstrate our visual interface on a real-life dataset: Multivariate measurements of a High-Performance Computing Center, the S&P500 stock data over the past 39 years, and the United States (US) monthly unemployment rates. Through these visual examples and use cases, we provide the intuitions on our visual representations (of the neural network snapshots, as well as the entire learning process), which helps to mitigate the hidden nature of the complex LSTM black boxes.

This paper is organized as follows. The next section presents related research in multivariate time series analytics and explainable artificial intelligence. Section 3 discusses our motivation, design choices, system overview, and details on major components of our system. Use cases of DeepVix is demonstrated in Section 4. Lastly, Section 5 concludes the paper and presents future direction for this work.

IAIT2020, July 01–03, 2020, Bangkok, Thailand

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

2 RELATED WORK

2.1 Multivariate Time Series Analytics

Popular algorithms to analyze univariate cases include Exponentially Weighted Moving Averages (Holt-Winters method) [53] and Autoregressive Integrated Moving Average (ARIMA) [31]. However, there is often no single definitive factor for the forecast outcome in real-world problems where multiple factors are required. Besides the temporal dependency, these data attributes are often interdependent on one another. One example of variable interdependency is CPU temperature readings from a high-performance computing center. The *temperatures of CPU* on system computers are subject to their *power consumption*, the *CPU load*, and *memory usage* measurements in the recent past and current time steps.

Traditional statistical methods can be adopted to analyze multivariate time series by reducing the dimensions or converting them into a univariate time series. Nguyen et al. proposed a simple framework [36] to visualize multivariate time series data by utilizing low-dimensional (2-D or 3-D) projection and maintain the ability to capture characteristics of the underlying data, enhance the interpretability of high-dimensional data. Other techniques include using statistical feature extraction [52] or dimensional reduction methods, such as Principal Component Analysis [54] and subspace analysis [8, 40]. Subspace analysis techniques rely on the observation that a subset of intrinsic dimensions often can represent the ambient dimensions in high-dimensional data [30]. In other words, we expect that data patterns are prominent only in a small subset of dimensions. Subspace clustering [1, 44] detect clusters and a set of prominent dimensions for each cluster. More recent subspace analysis approaches have been designed for users to intuitively navigate among different views between and within subspaces [10, 30]. Another approach is to use multivariate variations of these, such as Vector Autoagressive Moving Average [23].

Recent advancements in the machine learning field show its strong potential application in forecasting problems. Thus, there are works in the literature that proved the superiority of machine learning techniques over the traditional methods [25, 56] in time series prediction tasks. A profound empirical comparison of eight machine learning models for time series forecasting is presented in [2]. Their observation indicated that the performance depends on domain-specific data type, and no specific feature of the time series favors a particular model.

2.2 Long Short-Term Memory Networks

Artificial neural networks (ANNs) have become an active research area during the past few decades, containing a diverse set of network architectures. One of the most popular networks is feed-forward neural networks [47], which feeds information straight through and does not formulate any concepts of order in time. Recurrent networks are distinguished from feed-forward models by the feedback loop connected to their past decisions, as by giving particular weight to events that occur in a series [38]. In the class of recurrent networks, long short-term memory (LSTM) has emerged as an effective and scalable model for various domains related to sequential data [13], allows to explore and learn both long and short patterns and eliminates the problem of vanishing gradient [18, 45].

LSTMs networks have demonstrated great ability in learning long term correlations in a sequence [33], hence its broad applications on time series data. The scope of LSTM utilization ranges from anomaly detection [33], time series classification [27] to prediction and forecasting in various domains such as traffic flows [11], remaining useful life for aircraft engines [29] or water table depth [57].

2.3 Explainable Neural Networks

Besides the predictive power of neural network architectures, explainability remains a major limitation, as these architectures are essentially black boxes with respect to human understanding of their predictions [21]. Machine learning models are non-intuitive [14], leading to difficulty in understanding and explaining why a model works or fails. In the absence of a full understanding of how the model works, users would not know when to trust the model or how to correct an error. Understanding the neural network architecture includes knowing the architecture (e.g., layers, hidden units, layer type), learning paradigm (e.g., back-propagation method and learning rate), and other hyper parameters [25] (e.g., epochs and batch size).

Deep neural networks themselves that provide state-of-the-art performance may produce misclassification on adversarial examples – the ones that are intentionally modified to cause perturbations from correctly classified examples [12, 34, 39]. These examples are different from data generated by the augmentation method, in the nature of the unlikelihood to appear naturally throughout the dataset [12]. Regarding classification results adversarial as test data, state-of-the-art deep neural networks believe them to be recognizable objects with over 99% confidence [35]. Neural network policies in reinforcement learning are also vulnerable to adversarial attacks [7, 22].

In early research on adversarial examples, Goodfellow et. al [12] discovered that the direction of perturbation matters most. Hence these examples can be generated by applying gradient information. Rather than making changes on values of many dimensions, Su et al. [46] suggested that one pixel can be modified to originate adversarial attacks, using Differential Evolution: generating 'children' from parent samples. Training with adversarial examples can result in a regularized neural network [48]; even further regularization than dropout [12]. In more recent research, not only on the problem of image classification, speech recognition with neural network models can be vulnerable to adversarial examples [4]. The interpretability that explainable neural networks offer would benefit a broad domain. That said, the internal structure of a neural network – usually lies under the "hidden" components, can be more explicit in terms of architecture and parameters, aiding researchers to determine where in the process such a misclassification originates.

The approaches in explainable neural networks can be categorized in four types, namely attention mechanism, modular networks, feature identification, and learn to explain [14]. An excellent example of an attention mechanism approach is the work from [43] in giving captions to images. The proposed method can highlight the regions of the image that impacts its decision to make corresponding words in the caption. The modular approach jointly constructs several neural networks for individual tasks and composes them into deep networks for question answering task [5]. This approach

makes the deep neural networks more manageable and explainable. On the other hand, the feature identification approach finds ways to associate detected useful features with human-friendly languages or labels [14]. Lastly, the learn-to-explain method generates a deep model to explain in human-friendly words about the features produced by another model [14].

Recently, there are increasing attempts to enable explainable neural networks. Vaughan et al. presented a neural network that is designed specially to learn interpretable features [51]. To evaluate the statistical significance of the feature variables of a neural network, a significance test is developed by Horal et al. [20]. Regarding multivariate time series data, time and feature importance values are visualized in the form of heatmaps [6, 15]. Despite the wide application of LSTMs, there is a very limited number of study that inspects explainable LSTMs. Akerman et al. [3] proposed a technique on convolutional LSTM encoder-decoder models to highlight the most suspicious areas within an analyzed image. Overall, the visualization for partial components are presented in [6, 15] and [3] but not the whole network architecture itself. The complete network is explained in terms of image classifier for convolutional network [19] or typical feedforward network [49], but no consideration to LSTM models explicitly. To the best of our knowledge, there has been no prior work that considers the entire LSTM architecture in the light of visualization and explainability. To fill this gap, we propose a visualization-based technique, called DeepVix, to explain the LSTM network focusing on high-dimensional time series data.

3 THE DEEPVIX APPROACH

3.1 Justification

Machine learning utilizes machine computing power to automate analytical model building, which can then learn, analyze, and even make the decision on their own. However, machine learning is currently still a magic box to the end-users: users input the data, the system extracts, learns, and builds an intelligent system that gives the outputs to the users. Users are unaware of the underlying process and therefore left with many questions in their minds: how did the machines come up with these outputs, what are the critical factors in their decision, how much should we trust the system, and can we still improve the accuracy of the results. Without the capability to explain the underlying model, these questions are left unanswered to the users, and the users merely accept whatever the machine gives them.

While machines are faster, more stable, and tireless, the human is more flexible and adaptive to new changes. More importantly, the human should be the one who makes final decisions of the analysis. Therefore, the intelligent black-box needs to be transparent to users so that they can understand the rationale behind the outcomes and possibly modify the algorithm to improve the accuracy and better fit their needs. The main goals of this work are:

- To create a visual interface that can explain the emerging LSTM neural networks and their rationales.
- To support interactive operations that allow users to perform what-if analysis and understand essential factors (i.e., variables, neural nodes, and layers) and its major flows.
- To visually customize the neural network structure, compare the results, and make the decision on critical considerations, such as the trade-offs between time and accuracy.

This paper focuses on multivariate time series data, which is produced in various applications from the social to scientific domains. In the next section, we discuss the design choices for visual representations of multivariate time series, neural networks, and supported interactive operations.

3.2 Design choices

Line-graphs are frequently used in visualizing time series because of its ability to show trends and clear differences among items over time. However, for multivariate time series with a large number of channels, line-graphs induce visual cluttering issues. Heatmap visualization can help alleviate this problem by representing time series data in rows and columns and by leveraging other visualization attributes such as cell colors and borders [42]. In our cases, heatmaps aid the visualization with saving vertical and horizontal space by placing data points (or heatmap cells) closely, also prevent overplotting for large time series data [6]. Also, heatmaps would help users to convey the general pattern quickly by color distribution [15]. On the other hand, scatterplots are suitable for visualizing relationships among data points [41]. Our approach uses line-graphs to visualize the training and testing losses (mean squared error) over the training batches and epochs. We use heatmaps for displaying multivariate time series of input data and sequencing, hidden layer types such as Long Short-Term Memory (LSTM) layer [18]. DeepVix also uses scatterplots to visualize point-wise data, such as intermediate outputs of Dense hidden layers, as in the Core Layers in Keras Python Deep Learning library [28] and training and testing results. With proper orderings, the use of scatterplots in these places helps to reveal the correlation between the predicted outputs (of the intermediate layers, training, and testing results) and the actual training and testing values.

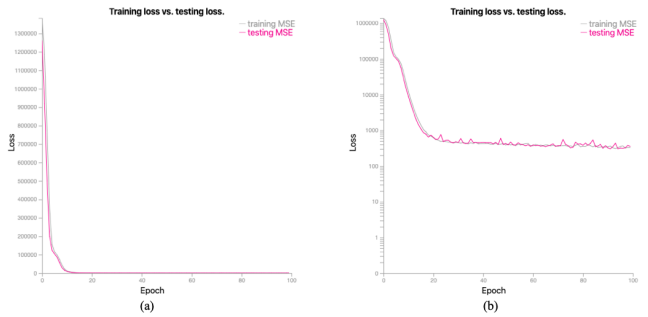

Figure 1: Training and testing loss of a LSTM model over training epochs (x-axis): (a) Normal loss scale (b) *Log* loss scale displayed on demand. The log scale on y-axis reveals more details on the MSEs when the trained model becomes more stable (at the later epochs).

Figure 1(a) shows an example of a training and testing loss graph applied to multivariate health metrics from high-performance computing (HPC) center. There are only two channels (training and

testing losses) in this graph. Thus, using a line graph helps to show clear changes and quantitative differences between the two variables over the training sequence. As the training loss drops quickly in the first few epochs, users can switch to vertical *log* scale to reveal more loss details on the later epochs, as shown in Figure 1(b). On the other hand, Figure 2(a) shows a sample output of an extracted feature (*feature 7*) from an intermediate LSTM layer (*lstm 1*) applied to the HPCC dataset. This heatmap shows a clear pattern that the model could learn during the training process of the model for the forecasting task. Also, Figure 2(b) is a scatterplot depicting training output vs. the actual target value for the predicted variable (*CPU temperature*). This scatterplot presents the correlation between the predicted values vs. the actual outputs, thus allows the user to evaluate the model training accuracy qualitatively.

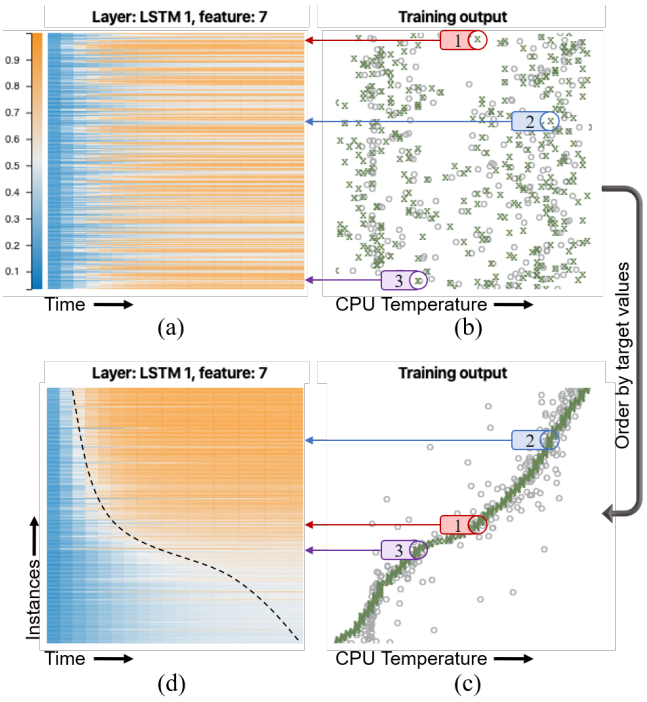

Figure 2: Visualizing an intermediate node and training output in the LSTM model: (a) Sample heatmap (for feature 7) of LSTM layer 1 and (b) The corresponding training output (c) The training output ordered by the target values and (d) The heatmap in (a) after vertically re-ordering. *CPU temperature* is the target variable that the model needs to predict.

3.2.1 Ordering. During the initial learning process, we have noticed that the heatmaps – representing input time series and intermediate features, each row depicts a data instance – are in the typical presentation where data instances are listed top-down as their input order. As depicted in Figure 2(a), there is no particular pattern can be observed in the heatmap. Figure 2(b) shows the corresponding training output, which is also in the same chaotic manner.

Ordering has been proved to be an efficient approach to improve the visualization [32, 50]. Here, we introduce a vertically ordering criterion by ascending target output values, first applied to final output (the resulting order is shown in Figure 2(c)), then we propagate this order to previous layers, which resulted in the presentation of heatmap as in Figure 2(d). We highlighted the three example instances to depict the effect of ordering by the target prediction values (*CPU temperature*). Notice that we only rearrange the result display after the learning process has finished, as an attempt to improve the visualization and hence enhance the interpretability, without taking any interferences on the training process. One can observe a possible dashed curve presented in the heatmap in Figure 2(d) and which resembles the green curve of target values in Figure 2(c): The curves appeared to be more precise and smoother as we go in deeper layers. This can be explained by a temporal signature that has been captured and learned over the training. Besides, the patterns are reflected relatively consistent throughout one layer; this possibly originates from the set of features that are learned in that layer.

3.2.2 Evolution of weights. In each epoch, the parameters – or weights, have been adjusted in the direction of minimizing loss. Observation of such evolution during the whole learning process will enable human in the loop machine learning, in a way that allows gaining instant insights from complex progress. In the design of DeepVix, we present the changes over time of each parameter via the horizontal section at the beginning of each gate: Changes in thickness and color illustrate the corresponding regulations in magnitude, direction, and importance value with regard to contribution to the final output. At the end of the horizontal section is the current weight, as it initiates the value for the rest of the line. Users can trace back to a specific snapshot of the model (to a certain batch) by selecting an epoch on training and testing loss in Figure 1. This feature is discussed further with use cases in section 4.

3.2.3 Other design choices. More neural network information visualized in the system gives a better understanding of the network architecture and hyper-parameters. However, it is nearly impossible to visualize every architectural and parameter details of the resulting model. This problem is especially true for complicated layer types, such as LSTM. Therefore, DeepVix does not try to visualize the sequence history weights (if there are) or biases. Furthermore, instead of visualizing all the weights for a flatten layer type (if there is), we accumulate these weights for each feature and visualize them. For instance, feeding an input sequence of 20 time-steps for eight extracted features in the previous layer to a flatten layer gives 20×8 output values. These *160* values lead to a large number of visual components if we choose to present them individually. Instead, this large number of weights can be accumulated per feature (8 features in the previous example) to represent the overall contributions of each one of them.

Even with the reduction strategy, complicated network structure still creates visual cluttering issues and leads to loss of focus on important information. One of the approaches to alleviate this cluttering issue is using interactions to highlight the interesting entities on demand. DeepVix uses interactive operations to filter the dense network and focus on the significant, contributing extracted features and essential weights. In particular, users can use a weight

filter slider to set a weight display threshold. So that, any weight with the absolute value scaled (at a specific layer) to the range [0, 1.0], which is smaller than the threshold, is faded out. Furthermore, the graphs of the input data or intermediate results without having any weights connected to them are not visible.

In addition, there is no obvious single best neural network model for any given real-life dataset. The analysts often need to try several models with different architectures and hyper-parameters. Therefore, our solution provides interactive features for the analysts to customize these configurations. Specifically, users could add, remove, or change input features, layers, layers' types, and the number of hidden units per layer and other hyperparameters such as training epochs and batch size.

Finally, it often takes a long time to train a neural network model. Also, due to the stochastic nature of the training process, there is no guarantee that the analysts could re-train and get the same model with the same set of training data and model configuration. Therefore, DeepVix provides the analysts with the options to save a trained model and load it at a later time either to use or to explore the trained model. Furthermore, the user can share the models with the other analysts or load the models generated from a different platform (e.g., Python) to this system to investigate it.

3.3 The DeepVix architecture

With all the design decisions as discussed, Figure 3 depicts how these are incorporated into our solution. DeepVix contains four main areas: neural network architecture configuration, training settings, training vs. testing loss overview, and exploration of weights and interactive features.

Saving and loading model. In the beginning, DeepVix gives the analyst a pre-built neural network with a default, sample configuration for each input dataset. The saving and loading panel can be seen in Figure 3 (A). During the process, the operation can be halted by a pausing option, and the trained model can be saved to local storage of the browser or the local machine for later investigation. The saved package includes three files for model topology, layer configurations, and corresponding weights. Users can also load a different model: The source for loading can be a server, the current browser's local storage, or the local machine.

Configuring network architecture. The analysts can start exploring the model using the menu from Figure 3 (D). Otherwise, users can use the menu provided from Figure 3 (C) to configure a new network architecture. For instance, users click on a button next to each layer to delete it. Equally important, users can click on the plus icon next to the output layer. A popup dialog allows the user to add a new layer, set the layer type (LSTM or Dense), set the number of hidden units, and the activation function. On top of that, users can directly modify the currently available layers with such specifications to update. After choosing this option, the process automatically stops and then starts over with the latest update specification.

From Figure 3 (C), analysts can also click on the icon next to the input layer to decide which input variables to be fed and retrain the model. This functionality is useful when not all the features are contributing to the accuracy of the final forecast due to noise or overfitting. Thus, users can select to leave one or more features out of the training process. After setting the network and input configurations, users can customize the training parameters (e.g., epochs, and batch size) before clicking the play button to start the training process, as shown in Figure 3 (A).

Observation on training and testing loss. The overfitting issue is known to be associated with epochs and batch size. To explore the model in this aspect, Figure 3 (B) shows the training and testing losses, updated dynamically. System analysts can observe this graph to know the equilibrium point, at which the training loss continues to reduce, but the testing loss starts to increase or to reduce the training time if the model converges early; in this case, the losses remain relatively stable. Users can also use this plot to go back to a previous model snapshot by selecting the timeline.

Visual levels of granularity. Levels of granularity for detail have been incorporated to support users in getting a good grasp of the architecture of the network. Each line connecting any two layers represents the value of the corresponding weight. Users can filter out the weights and keep the desirable ones by utilizing the toggle menu or weight threshold slider, by which the weights in selected types remain on display. Regarding toggling, for instance, an LSTM layer has for type of weights: input gate, forget gate, cell state, and output gate, as shown in Figure 4 (A); a typical layer would have weights in negative or positive values, as shown in Figure 4 (B). For a global scope, DeepVix provides a slider for filtering out weights with absolute values in each layer, which would affect the network as a whole to reduce visual clutter.

The detail on demand for each node and graph is support with a close-up view. When the user clicks on an arbitrary node or graph, a detail view is provided with full view and complete information such as values alongside each axis, detail information of the node or graph, and explicit scaling range.

3.4 The DeepVix implementations

DeepVix is developed using JavaScript and in particular the D3.js library [9]. The online DeepVix prototype, the explanation and demo video, and source code on our project website at https://git.io/JeD2a. On this project website, the viewers can also find the analysis of a neural network for more complex time series, the usage of large model architectures, as well as their experimental results due to the space constraints.

Could univariate time series be analyzed by our approach? Univariate time series can be divided into a number of chunks, which can be considered as instances for training and testing purposes. For certain data (such as the stock values), long historical data should not have any influences on the current predictions. However, their seasonal patterns (weekly, monthly, or quarterly) might be useful for the predictive analysis process. In these cases, we should consider dividing the data into smaller chunks. We will demonstrate this approach in the next section.

4 USE CASE

In this section, we use three real-world datasets to demonstrate our DeepVix prototype. The first is computer measurements at an HPC center, the second is S&P500 stock data, and the last is a series of US unemployment data on various economy sectors.

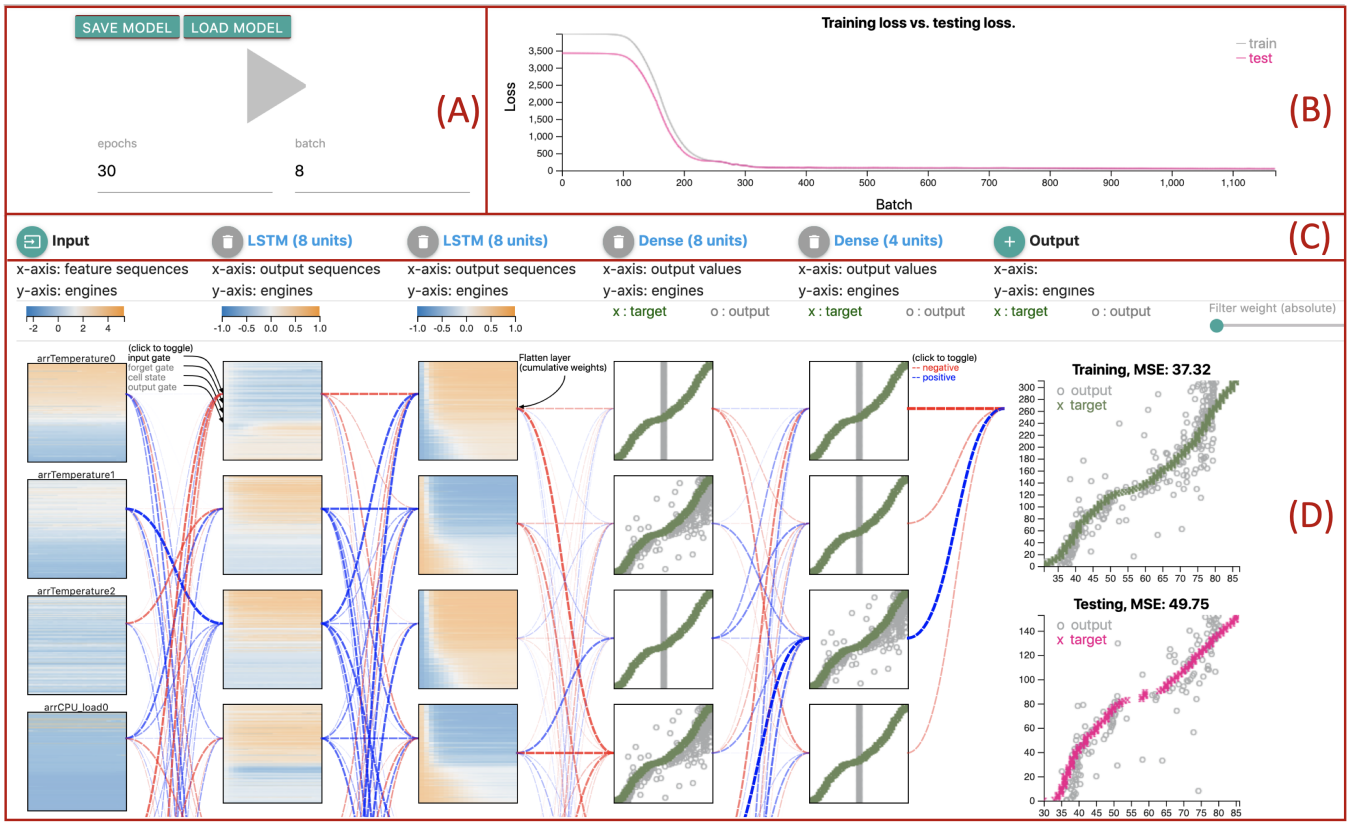

Figure 3: Schematic overview and major components of DeepVix: (A) Training settings, (B) Training vs. testing loss, (C) Configuration settings, and (D) LSTM model and interactive features.

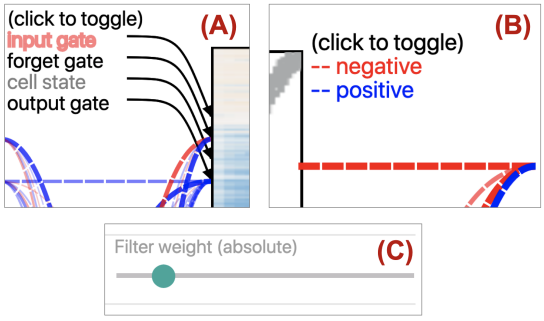

Figure 4: DeepVix allows users to filter values for a clear view of corresponding weights by: (A) toggle options for LSTM layer, (B) toggle negative or positive weights, (C) Filtering absolute weights (applied for the entire network).

4.1 CPU temperature monitoring in HPC system

The dataset contains ten variables, which are computer health readings for every 5-minute interval within 5 hours, including CPU load, fan speeds, memory usage, and power consumption. In total, we have 20 timesteps of 467 nodes in the cluster. Our target variable for this data is the *CPU temperature*. In this use-case, we are not striving for the best model with the highest prediction results. Instead, we would like to demonstrate how the data analyst could use DeepVix for model exploration purposes. We approach this use-case with a neural network architecture with two LSTM layers (eight hidden nodes each), then two Dense layers (eight and four hidden nodes correspondingly). For future reference, we name this configuration as *HPCC8884*, which somewhat reasonable configuration for the given time-series data [24]. Figure 5 shows the visualizations of a sample model trained using this architecture. It is noticeable that several generated features are similar in this model. Specifically, the elements at the blue arrows are identical, and the gray dots are displayed vertically (no useful information was learned); therefore, they can be removed from the current architecture without affecting the prediction results. Additionally, the two extracted features at the red arrows are not contributing much to the final predictions (the paths representing their contributions to the output are missing).

Computational experiments. Three different LSTM neural network architectures of various sizes are inspected with our framework to construct models for predicting health monitoring index of CPU in High-Performance Computing Center [37]. We first introduce a simple setting: Configuration 1 (8-4-2) has two LSTM layers – with 8 and 4 nodes respectively, and only one Dense layer containing

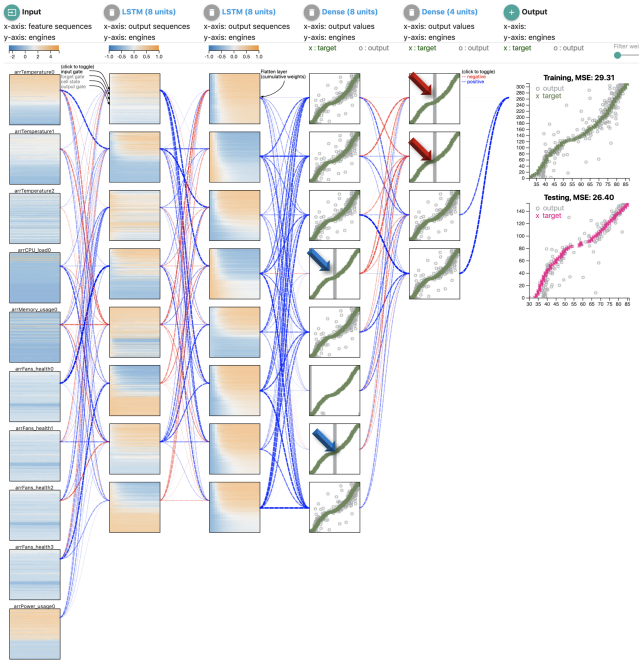

Figure 5: DeepVix visualization for Configuration 2: two LSTM layers (eight hidden units each), and two Dense layers (eight and four hidden units correspondingly).

two nodes. Configuration 2 (8-8-8-4) has two LSTM layers – eight nodes each, and two Dense layers – with 8 and 4 nodes respectively. We add another 16-node LSTM layer into Configuration 1 to form a more complex Configuration 3 (16-8-8-8-4). Table 2 lists the details of these configurations for our experiments.

	Input Features	LSTM 1	LSTM 2	LSTM 3	Dense 1	Dense 2
Config. 1	10	8	4	-	2	-
Config. 2	10	8	8	-	8	4
Config. 3	10	16	8	8	8	4

Table 1: Three LSTM neural network configurations experimented.

Table 2 presents the experimental results for0.8 the above neural network configurations. All tests were performed on 2.9 GHz Intel Core i5, MacOS Sierra Version 10.12.1, Memory 8 GB RAM. Each of the configurations was executed 31 times; its training time and MSE are recorded. Mean and standard deviation for each configuration are calculated afterward. The degrees of freedom df is 30. We discuss the results in pairs: Configuration 1 vs. 2, and Configuration 2 vs. 3 using t-test [17] to compare the means of two corresponding MSEs in order to determine whether there is statistical evidence that the means are similar. For the pair of Configuration 1 and 2, the result is significant at $p < 0.05$. The t-value is 12.36382, and the p-value is < 0.00001, meaning that there is a strong chance that the two sets of MSEs are significantly different. One can observe from Table 1, as well as in Figure 6, that training time increases

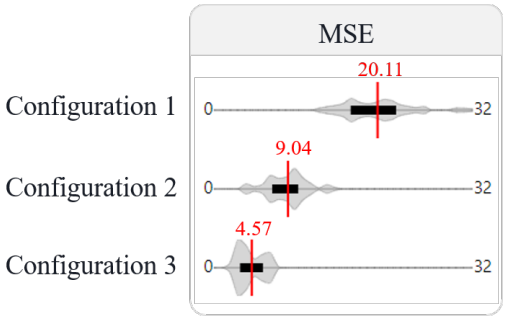

Figure 6: Comparison of three different configurations for the HPC readings data. More complex configuration (Configuration 3) generates smaller cost (MSE=4.57, or 2 degree different compared to the actual CPU temperature) at the cost of training time.

when we consider Configuration 1, 2, 3, whereas MSE decreases in that same order. The trade-off here is between accuracy and training time, which is a common issue in machine learning and has been discussed in previous literature [16]. Regarding the pair of Configuration 2 and 3, we also consider the significance level at 0.05. In this case, the t-value is 8.30513, and the p-value is < 0.00001. Compare two values of t-values, that of the Configuration 1-2 is higher than that of Configuration 2-3, we can observe the results produced by Configuration 2 have a higher chance to be similar to Configuration 3 than to Configuration 1. The more complex model yields better outcomes in terms of MSE, but the gain is less remarkable as we push further on the model complexity (from Configuration 2 to Configuration 3).

Model interaction and exploration. A useful interactive operation for the exploration of the learned model is weight filtering. This functionality helps to focus on the raw and extracted features with respect to significant contributions to the final prediction result. For instance, Figure 7, shows the same model as in Figure 5 but with the *output gates* weight filter threshold set to 0.75. It shows that the predicted *CPU temperature* strongly depends on other *CPU temperatures*, the *CPU load*, and the *power consumption*. Furthermore, these network contribution also suggest another exploration to improve training time or prediction performance, or could be both. These customization options are supported in our DeepVix interface as described in the previous section.

4.2 S&P500 stock prediction

We retrieve the data based on the daily stock market price index for S&P500 (GSPC), extracted from the Yahoo Finance website [55]. The dataset covers stock records for five weekdays each week, in the period of 39 years, from 1980 to 2019. Each record contains the timestamp, stock price at “Open”, “High”, “Low”, “Close”, and “Volume” of the stock that day. During the training and testing process, we utilize the attributes of the stock price on Monday, Tuesday, Wednesday, and Thursday to predict Close price for Friday. Due to the dynamic nature of stock data, learning longer historical data is not desirable. Therefore, each node in the intermediate layer contains four time steps set horizontally, from Monday to Thursday.

	Config 1		Config 2		Config 3	
No.	Time	MSE	Time	MSE	Time	MSE
1	2.96	21.58	3.58	3.96	4.13	5.87
2	2.90	23.53	3.66	6.70	4.13	4.14
3	2.79	14.03	3.60	6.51	4.13	5.33
4	2.79	18.10	3.72	9.56	4.12	7.18
5	2.76	15.93	3.18	9.30	4.15	4.37
6	2.82	29.86	3.37	7.73	4.14	2.34
7	2.80	15.29	3.59	14.27	4.12	5.51
8	2.83	15.08	3.39	12.27	4.13	3.34
9	2.81	20.59	3.63	9.16	4.16	3.04
10	2.83	22.81	3.45	10.70	4.16	2.81
11	2.83	20.05	3.71	12.04	4.14	6.36
12	2.84	16.86	3.65	10.14	4.16	3.61
13	2.83	18.41	3.24	7.37	4.14	6.96
14	2.76	19.04	3.57	14.14	4.16	3.27
15	2.82	26.01	3.42	10.41	4.13	3.72
16	2.84	25.37	3.41	4.85	4.14	3.41
17	2.82	16.98	3.49	10.15	4.14	6.09
18	2.80	18.80	3.45	8.80	4.15	4.50
19	2.80	31.34	3.45	7.06	4.15	6.86
20	2.79	13.34	3.43	4.08	4.14	2.73
21	2.79	21.83	3.50	9.71	4.15	3.18
22	2.79	20.47	3.31	10.07	4.13	7.11
23	2.79	17.38	3.09	8.79	4.18	7.09
24	2.79	16.93	3.11	6.51	4.12	5.68
25	2.82	20.31	3.12	7.85	4.14	3.82
26	2.86	24.19	3.23	10.75	4.15	2.89
27	2.80	22.54	3.13	11.35	4.12	3.85
28	2.79	17.47	3.16	7.24	4.13	3.42
29	2.81	16.37	3.21	10.16	4.13	3.62
30	2.82	21.43	3.19	8.26	4.12	2.88
31	2.69	21.50	3.24	10.45	4.13	6.74
Mean	2.81	20.11	3.40	9.04	4.14	4.57
StDev	0.04	4.29	0.20	2.54	0.02	1.59

Table 2: Training time (in minutes) and MSE for the three neural network configurations. Note: Execution number (No); Standard deviation (StDev).

Figure 8 presents our DeepVix model for the S&P500 stock market price dataset through two system snapshots: At the third epoch and at the tenth epoch, with corresponding close up views for two weight evolution at position 1 and 2, respectively. The weight evolution visualization (examples in panels A1, A2, B1, B2) records the changes of weights from the starting point until the current timestamp. Over time, updates of weight are stack horizontally, as in panel A1 and B1: Epoch 3 shifted to the left as new epochs are added in. From panel B1, we can observe that the thickness of lines decreases significantly over time, indicating that there is a major reduction in the magnitude of these parameters reduces during the training process, hence less contribution of this node to the next output. Another interesting pattern is found in panel A2 and B2. In panel A2, there are several negative weights switch into positive

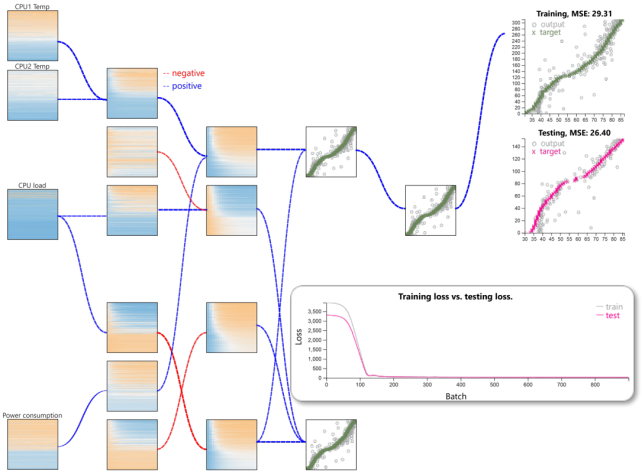

Figure 7: Weight filtering functionality allows the analysts to focus on the raw and extracted features with significant contributions to the final prediction results. We filtered out *output gates* with the absolute weights smaller than 0.75.

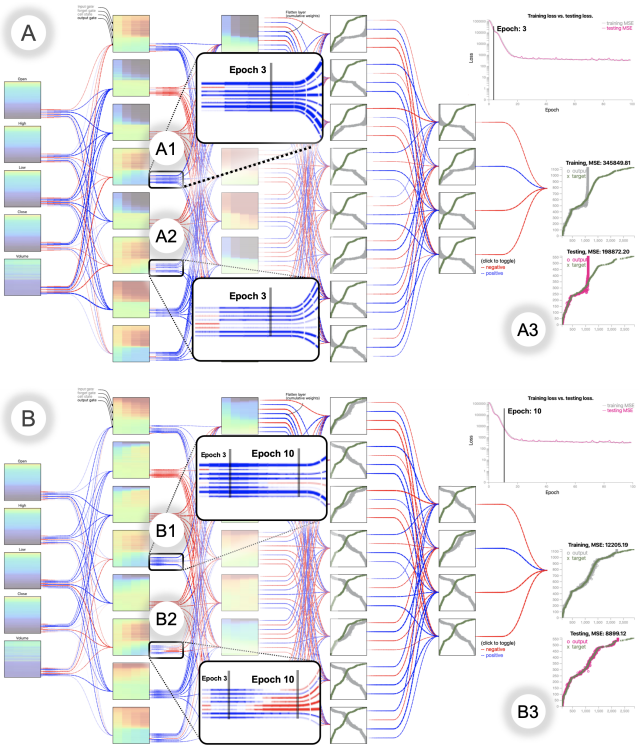

Figure 8: DeepVix visual model for the S&P500 stock prediction. (A) The model snapshot at the 3^{rd} epoch, with close up view for the evolution of weights, (B) The model snapshot at the 10^{th} epoch, with close up view for the evolution of corresponding weights.

right after the first epoch, resulted in an all-positive set of parameters in later epochs, hence the positive contribution to the following layer. However in panel B2, we can see major of the parameters have remarkable changes: The originally thick lines decrease their width, the originally thin, positive lines switch into negative and adjust to the larger magnitude. There is a corresponding movement in the training - testing loss line chart, where the curve witness a turning point (the 5^{th} or 6^{th} epoch) in both training and testing MSE curves.

In the early stage of training process (epoch 3 - panel A3 of Figure 8), the scatterplots for training MSE and testing MSE both contain a vertical formation of outputs. This can be explained by the activation function ReLU: Negative input will result in zero output, as can be seen in the first, third and fourth nodes in the last Dense layer right before the final output A3. At this stage, the learning process just started and parameters are not tuned properly. As we move on to panel B3 at epoch 10, the outputs are now align with the target in better shape. Notice that at the last Dense layer, the only one node has positive weight is the second one from top down, with the outputs align in similar direction as target, whereas the other three nodes possess opposite orientation to the target, hence their negative weights. This observation correlates with the nature of neural network and machine learning in general: On the process of minimizing loss, there are rewards for positive contributions and penalties for negative contributions. Without proper arrangement and visual representation, we would not be able to discern these visual characteristics of parameters in complex neural networks.

4.3 US Employment data

The US unemployment data comprise monthly for 50 states over 20 years, from 1999 to 2018. The data were retrieved from the US Bureau of Labor Statistics. There are 15 in the collected data, including Total Nonfarm, Construction, Manufacturing, Education and Health, and Government. We want to explore the important economic factor associated with the target variable: the monthly unemployment rates of the states.

Figure 9 shows the final snapshot of the LSTM model with 3 LSTM layers (16-8-8) and 2 Dense layers (8-4) in our DeepVix visual interface. Within the LSTM layer, we can see the diagonal patterns progress from left to right. In training vs. testing loss plots, the MSE training is smaller than MSE testing as learning and predicting social behavior is a challenging task (compared to the physical or natural series, such as the CPU temperature in the first use case). The Dense layer at the arrows, we highlight two scatterplots where the predicted data points do not fit the green actual values well. Therefore, this trained LSTM model decides not to take any contributions from both nodes into the output.

Figure 10 enlarges the two heatmaps (at the blue and orange arrows in Figure 9) represent the original raw variables and one LSTM node of the trained model. The linear top-down gradient of the input data has been replaced by the diagonal patterns, which resemble the actual value curves (the green curve in the output scatterplot). The more resemblance these patterns are, the better contributions they are into the prediction mechanism. Notice that the color scales can be select by the users based on their preference.

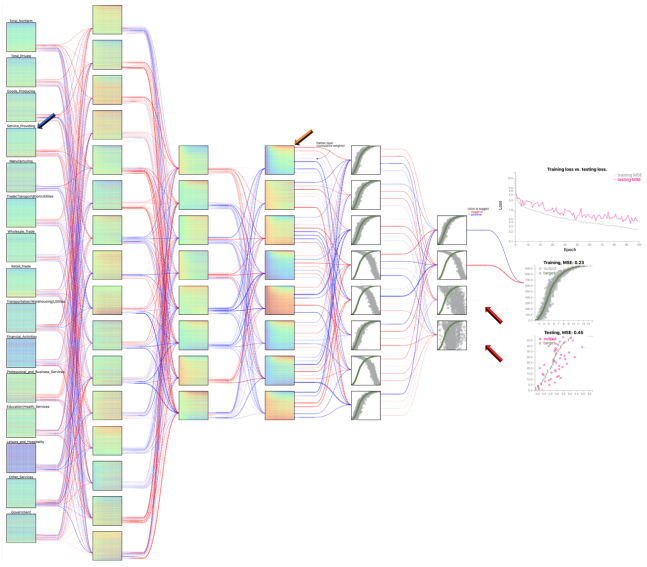

Figure 9: Our DeepVix visualization of the triained LSTM model for the US Employment data. Input layer on the left contains various economy sectors. Data instances in the heatmap are the 50 states.

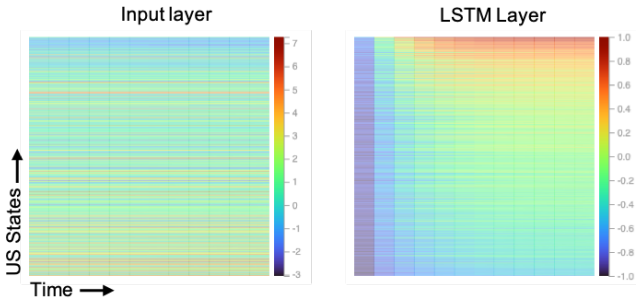

Figure 10: The heatmaps represent original variable (Service_Providing), on the left versus one sample learned feature at the last LSTM layer (feature 0, on the right). The diagonal pattern is clearly visible on the right.

5 CONCLUSION

This paper proposes a visual framework that aims to combine the strengths of both visual analytics and machine learning in analyzing and predicting multivariate time series data. The prototype presents intermediate steps of the LSTM model (every node is plotted as a heatmap or a scatterplot) and supports interactive operations (such as filtering, ordering, and details on demands) to explore, understand, and customize the NNs to fit their constraints, such as the trade-offs between training time and accuracy. Our DeepVix prototype also keep track and represent the learning information, such as the neural weights of various gates in the LSMT models. This allows users to discern the critical steps in the learning process quickly. Additionally, the framework allows users to inject their domain knowledge into the ANNs to make it more flexible and

adaptive to the new requirements or dynamic characteristics of the problem. In the Use case section, we demonstrate our DeepVix prototype on real-world datasets to predict the CPU temperatures of the High-Performance Computing Center, the S&P500 stock values, and the US employment rate of the states. We also provide some intuitions and findings which can make the first step into understanding and exploring the complex nature of the Neural neural, specifically the LSTM models for multivariate time series data in this case. In future work, we will apply the proposed approach for larger data sets of hundreds of dimensions and an extended time series with more complex LSTM architectures. We will also investigate the explainability for other types of data/applications, such as networks, geographic data, and visual representations.

REFERENCES

[1] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. 1998. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data* (Seattle, Washington, USA) *(SIGMOD '98)*. ACM, New York, NY, USA, 94–105. https://doi.org/10.1145/276304.276314

[2] Nesreen K Ahmed, Amir F Atiya, Neamat El Gayar, and Hisham El-Shishiny. 2010. An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews* 29, 5-6 (2010), 594–621.

[3] Sefi Akerman, Edan Habler, and Asaf Shabtai. 2019. VizADS-B: Analyzing Sequences of ADS-B Images Using Explainable Convolutional LSTM Encoder-Decoder to Detect Cyber Attacks. *arXiv preprint arXiv:1906.07921* (2019).

[4] Moustafa Alzantot, Bharathan Balaji, and Mani Srivastava. 2018. Did you hear that? adversarial examples against automatic speech recognition. *arXiv preprint arXiv:1801.00554* (2018).

[5] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 39–48.

[6] Roy Assaf and Anika Schumann. 2019. Explainable deep neural networks for multivariate time series predictions. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 6488–6490.

[7] Vahid Behzadan and Arslan Munir. 2017. Vulnerability of deep reinforcement learning to policy induction attacks. In *International Conference on Machine Learning and Data Mining in Pattern Recognition*. Springer, 262–275.

[8] Enrico Bertini, Alessio Di Girolamo, and Giuseppe Santucci. 2007. See What You Know: Analyzing Data Distribution to Improve Density Map Visualization. In *Eurographics/ IEEE-VGTC Symposium on Visualization*, K. Museth, T. Moeller, and A. Ynnerman (Eds.). The Eurographics Association. https://doi.org/10.2312/VisSym/EuroVis07/163-170

[9] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D3 Data-Driven Documents. *IEEE Trans. Vis. Comput. Graph.* 17, 12 (2011), 2301–2309.

[10] Ines Farber, Andrada Tatu, Daniel Keim, Thomas Seidl, Fabian Maas, Tobias Schreck, and Enrico Bertini. 2012. Subspace Search and Visualization to Make Sense of Alternative Clusterings in High-dimensional Data. In *Proceedings of the 2012 IEEE Conference on Visual Analytics Science and Technology (VAST) (VAST '12)*. IEEE Computer Society, Washington, DC, USA, 63–72. https://doi.org/10.1109/VAST.2012.6400488

[11] Rui Fu, Zuo Zhang, and Li Li. 2016. Using LSTM and GRU neural network methods for traffic flow prediction. In *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. IEEE, 324–328.

[12] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[13] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2016. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems* 28, 10 (2016), 2222–2232.

[14] David Gunning. 2017. Explainable artificial intelligence (xai). *Defense Advanced Research Projects Agency (DARPA), nd Web* 2 (2017).

[15] Tian Guo, Tao Lin, and Nino Antulov-Fantulin. 2019. Exploring Interpretable LSTM Neural Networks over Multi-Variable Data. *arXiv preprint arXiv:1905.12034* (2019).

[16] Junfeng He, Shih-Fu Chang, Regunathan Radhakrishnan, and Claus Bauer. 2011. Compact hashing with joint optimization of search accuracy and time. In *CVPR 2011*. IEEE, 753–760.

[17] Timothy Heeren and Ralph D'Agostino. 1987. Robustness of the two independent samples t-test when applied to ordinal scaled data. *Statistics in medicine* 6, 1 (1987), 79–90.

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

[19] Fred Hohman, Haekyu Park, Caleb Robinson, and Duen Horng Chau. 2020. Summit: Scaling Deep Learning Interpretability by Visualizing Activation and Attribution Summarizations. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* (2020). https://fredhohman.com/summit/

[20] Enguerrand Horel and Kay Giesecke. 2019. Towards explainable ai: Significance tests for neural networks. *arXiv preprint arXiv:1902.06021* (2019).

[21] Ronghang Hu, Jacob Andreas, Trevor Darrell, and Kate Saenko. 2018. Explainable neural computation via stack neural module networks. In *Proceedings of the European conference on computer vision (ECCV)*. 53–69.

[22] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. 2017. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284* (2017).

[23] E. Isufi, A. Loukas, N. Perraudin, and G. Leus. 2019. Forecasting Time Series With VARMA Recursions on Graphs. *IEEE Transactions on Signal Processing* 67, 18 (Sep. 2019), 4870–4885. https://doi.org/10.1109/TSP.2019.2929930

[24] Lahiru Jayasinghe, Tharaka Samarasinghe, Chau Yuen, Jenny Chen Ni Low, and Shuzhi Sam Ge. 2018. Temporal convolutional memory networks for remaining useful life estimation of industrial machinery. *arXiv preprint arXiv:1810.05644* (2018).

[25] Dayhoff J.E. and DeLeo J.M. 2001. Artificial Neural Networks Opening the Black Box. *Cancer* 91, S8 (2001), 1615–1635. https://doi.org/10.1016/S0967-067X(01)00020-4

[26] Jobiya John and Sreeja Ashok. 2019. Process Framework for Modeling Multivariate Time Series Data. In *Advances in Intelligent Systems and Computing*. 577–588. https://doi.org/10.1007/978-981-13-0514-6_56

[27] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. 2017. LSTM fully convolutional networks for time series classification. *IEEE Access* 6 (2017), 1662–1669.

[28] Keras. 2019. Core Layers - Keras Documentation. https://keras.io/layers/core/.

[29] Dy D. Le, Vung Pham, Huyen N. Nguyen, and Tommy Dang. 2019. Visualization and Explainable Machine Learning for Efficient Manufacturing and System Operations. *Smart and Sustainable Manufacturing Systems* 3, 2 (Feb. 2019), 20190029. https://doi.org/10.1520/ssms20190029

[30] S. Liu, B. Wang, J. J. Thiagarajan, P.-T. Bremer, and V. Pascucci. 2015. Visual Exploration of High-Dimensional Data Through Subspace Analysis and Dynamic Projections. *Comput. Graph. Forum* 34, 3 (June 2015), 271–280. https://doi.org/10.1111/cgf.12639

[31] Alberto Luceño and Daniel Peña. 2008. *Autoregressive Integrated Moving Average (ARIMA) Modeling*. American Cancer Society. https://doi.org/10.1002/9780470061572.eqr276 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470061572.eqr276

[32] Sheng Ma and J Hellerstein. 1999. Ordering categorical data to improve visualization. *INFOVIS-99* (1999).

[33] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. 2015. Long short term memory networks for anomaly detection in time series. In *Proceedings*. Presses universitaires de Louvain, 89.

[34] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2574–2582.

[35] Anh Nguyen, Jason Yosinski, and Jeff Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 427–436.

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

[37] Ngan Nguyen and Tommy Dang. 2019. HiperViz: Interactive Visualization of CPU Temperatures in High Performance Computing Centers. In *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)* (Chicago, IL, USA) *(PEARC '19)*. ACM, New York, NY, USA, Article 129, 4 pages. https://doi.org/10.1145/3332186.3337959

[38] Chris Nicholson. 2019. A Beginner's Guide to LSTMs and Recurrent Neural Networks. *Skymind. Saatavissa: https://skymind. ai/wiki/lstm. Hakupäivä* 6 (2019), 2019.

[39] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 372–387.

[40] Lance Parsons, Ehtesham Haque, and Huan Liu. 2004. Subspace clustering for high dimensional data: a review. *SIGKDD Explor. Newsl.* 6, 1 (June 2004), 90–105. https://doi.org/10.1145/1007730.1007731

[41] Vung Pham and Tommy Dang. 2019. SOAViz: Visualization for Portable X-ray Fluorescence Soil Profiles. In *Workshop on Visualisation in Environmental Sciences (EnvirVis)*, Roxana Bujack, Kathrin Feige, Karsten Rink, and Dirk Zeckzer (Eds.). The Eurographics Association. https://doi.org/10.2312/envirvis.20191102

[42] V. V. Pham and T. Dang. 2018. MTDES: Multi-dimensional Temporal Data Exploration System; Strong Support for Exploratory Analysis Award in VAST 2018, Mini-Challenge 2. In *2018 IEEE Conference on Visual Analytics Science and Technology (VAST)*. 100–101. https://doi.org/10.1109/VAST.2018.8802440

[43] Vasili Ramanishka, Abir Das, Jianming Zhang, and Kate Saenko. 2017. Top-down visual saliency guided by captions. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* 2017-January (2017), 3135–3144. https://doi.org/10.1109/CVPR.2017.334

[44] Mahdi Soltanolkotabi, Ehsan Elhamifar, and Emmanuel J. Candès. 2013. Robust Subspace Clustering. *CoRR* abs/1301.2603 (2013). http://arxiv.org/abs/1301.2603

[45] Daniel Soutner and Luděk Müller. 2013. Application of LSTM neural networks in language modelling. In *International Conference on Text, Speech and Dialogue*. Springer, 105–112.

[46] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. 2019. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation* (2019).

[47] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. 1997. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems* 39, 1 (1997), 43–62.

[48] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[49] Tensorflow Playground. 2017. A Neural Network Playground. https://playground.tensorflow.org/.

[50] Dan Tsafrir, Ilan Tsafrir, Liat Ein-Dor, Or Zuk, Daniel A Notterman, and Eytan Domany. 2005. Sorting points into neighborhoods (SPIN): data analysis and visualization by ordering distance matrices. *Bioinformatics* 21, 10 (2005), 2301–2308.

[51] Joel Vaughan, Agus Sudjianto, Erind Brahimi, Jie Chen, and Vijayan N Nair. 2018. Explainable neural networks based on additive index models. *arXiv preprint arXiv:1806.01933* (2018).

[52] X. Wang, A. Wirth, and L. Wang. 2007. Structure-Based Statistical Features and Multivariate Time Series Clustering. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. 351–360. https://doi.org/10.1109/ICDM.2007.103

[53] Peter R. Winters. 1960. Forecasting Sales by Exponentially Weighted Moving Averages. *Management Science* (1960). https://doi.org/10.1287/mnsc.6.3.324

[54] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems* 2, 1-3 (1987), 37–52.

[55] Yahoo Finance. 2019. S&P 500 (GSPC). https://finance.yahoo.com/quote/%5EGSPC.

[56] Chao-Lung Yang, Chen-yi Yang, Zhi-xuan Chen, and Nai-wei Lo. 2019. Multivariate Time Series Data Transformation for Convolutional Neural Network. In *2019 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 188–192. https://doi.org/10.1109/SII.2019.8700425

[57] Jianfeng Zhang, Yan Zhu, Xiaoping Zhang, Ming Ye, and Jinzhong Yang. 2018. Developing a Long Short-Term Memory (LSTM) based model for predicting water table depth in agricultural areas. *Journal of hydrology* 561 (2018), 918–929.