Deep Learning for Time-Series Analysis

John Gamboa

University of Kaiserslautern
Kaiserslautern, Germany

Abstract. In many real-world application, e.g., speech recognition or sleep stage classification, data are captured over the course of time, constituting a Time-Series. Time-Series often contain temporal dependencies that cause two otherwise identical points of time to belong to different classes or predict different behavior. This characteristic generally increases the difficulty of analysing them. Existing techniques often depended on hand-crafted features that were expensive to create and required expert knowledge of the field. With the advent of Deep Learning new models of unsupervised learning of features for Time-series analysis and forecast have been developed. Such new developments are the topic of this paper: a review of the main Deep Learning techniques is presented, and some applications on Time-Series analysis are summaried. The results make it clear that Deep Learning has a lot to contribute to the field.

Keywords: Artificial Neural Networks, Deep Learning, Time-Series

1 Introduction

Artificial Neural Networks (ANN), since their origin in 1943 [46], have been used to solve a large range of problems as diverse as robotic processing [32], object recognition [56], speech and handwriting recognition [19], and even real time sign-language translation [2]. Despite the intuition that deeper architectures would yield better results than the then more commonly used shallow ones, empirical tests with deep networks had found similar or even worse results when compared to networks with only one or two layers [57] (for more details, see [6]). Additionally, training was found to be difficult and often inefficient [6]. Längkvist [34] argues that this scenario started to change with the proposal of *greedy layer-wise unsupervised learning* [22], which allowed for the fast learning of Deep Belief Networks, while also solving the vanishing gradients problem [7]. Latest deep architectures use several modules that are trained separately and stacked together so that the output of the first one is the input of the next one.

From stock market prices to the spread of an epidemic, and from the recording of an audio signal to sleep monitoring, it is common for real world data to be registered taking into account some notion of time. When collected together, the measurements compose what is known as a *Time-Series*. For different fields, suitable applications vary depending on the nature and purpose of the data:

while doctors can be interested in searching for anomalies in the sleep patterns of a patient, economists may be more interested in forecasting the next prices some stocks of interest will assume. These kinds of problems are addressed in the literature by a range of different approches (for a recent review of the main techniques applied to perform tasks such as Classification, Segmentation, Anomaly Detection and Prediction, see [14]).

This paper reviews some of the recently presented approaches to performing tasks related to Time-Series using Deep Learning architectures. It is important, therefore, to have a formal definition of Time-Series. Malhotra et al. [44] defined Time-Series as a vector $X = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}\}$, where each element $\mathbf{x}^{(t)} \in R^m$ pertaining to X is an array of m values such that $\{x_1^{(t)}, x_2^{(t)}, \ldots, x_m^{(t)}\}$. Each one of the m values correspond to the input variables measured in the time-series.

The rest of this paper is structured as follows: Section 1.1 introduces basic types of Neural Network (NN) modules that are often used to build deep neural structures. Section 2 describes how the present paper relates to other works in the literature. Sections 3, 4 and 5 describe some approaches using Deep Learning to perform Modeling, Classification and Anomaly Detection in Time-Series data, respectively. Finally, Section 6 concludes the paper.

1.1 Artificial Neural Network

This section explains the basic concepts related to ANN. The types of networks described here are by no means the only kinds of ANN architectures found in the literature. The reader is referred to [49] for a thorough description of architectural alternatives such as Restricted Boltzmann Machines (RBM), Hopfield Networks and Auto-Encoders, as well as for a detailed explanation of the Backpropagation algorithm. Additionally, we refer the reader to [19] for applications of RNN as well as more details on the implementation of a LSTM, and to [54] for details on CNN.

An ANN is basically a network of computing units linked by directed connections. Each computing unit performs some calculation and outputs a value that is then spread through all its outgoing connections as input into other units. Connections normally have weights that correspond to how strong two units are linked. Typically, the computation performed by a unit is separated into two stages: the *aggregation* and the *activation* functions. Applying the aggregation function commonly corresponds to calculating the sum of the inputs received by the unit through all its incoming connections. The resulting value is then fed into the activation function. It commonly varies in different network architectures, although popular choices are the logistic sigmoid ($\sigma(x) = \frac{1}{1+e^{-x}}$) and the hyperbolic tangent ($tanh(x) = \frac{2}{1+e^{-2x}} - 1$) functions. Recently, rectified linear units employing a ramp function ($R(x) = \max(0, x)$) have become increasingly popular.

The input of the network is given in a set of input computing units which compose an *input layer*. Conversely, the output of the network are the values output by the units composing the *output layer*. All other units are called *hidden* and are often also organized in layers (see Figure 1b for an example network).

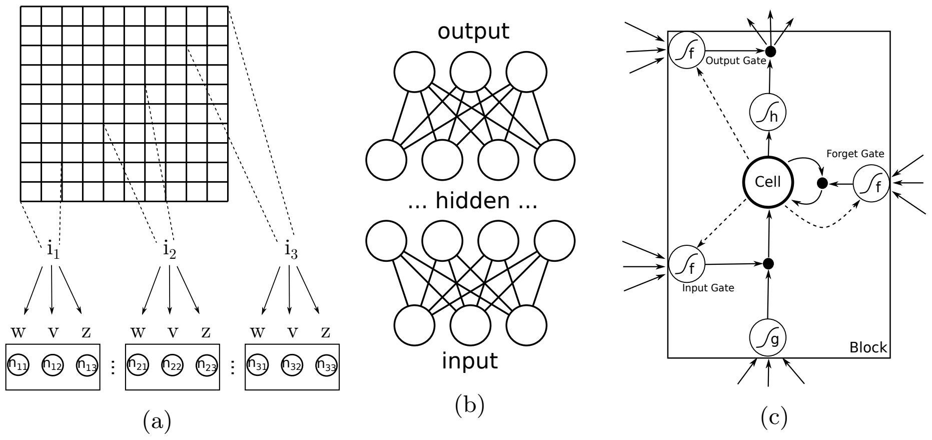

Fig. 1: (a) The *convolutional layer* of a CNN with three groups (also called "filters"). Each group performs a 2×2 convolution in the image: each neuron in the group is connected to a different region of the image but shares the same weights, producing a new image. In the example, the weight vector w is shared by all neurons n_{j1}, the vector v is shared by all neurons n_{j2}, and z is shared by all neurons n_{j3}. If pooling is applied, it is applied to each one of the three newly produced images; (b) An Artificial Neural Network with one input layer composed of three neurons, two hidden layers composed, each one, of four neurons, and one output layer composed of three neurons. Each node of a layer is connected to all nodes of the next layer; (c) A LSTM block (adapted from [19]).

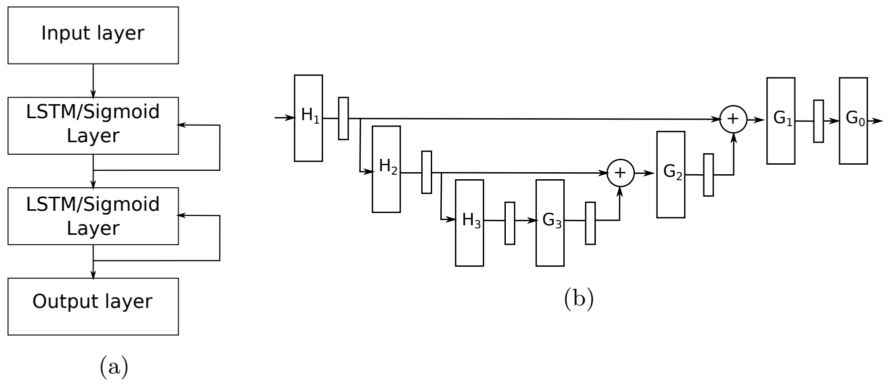

Fig. 2: (a) The proposed "Stacked Architecture" for performing Anomaly Detection (adapted from [44]); (b) The architecture of a UFCNN (adapted from [47]).

The focus of learning algorithms is frequently on deciding what weights would cause the network to output, given some input, the expected values. A popular learning algorithm is the Backpropagation algorithm [51], whereby the gradient of an error function is calculated and the weights are iteratively set so as to minimize the error.

Convolutional Neural Network (CNN) A network that is too big and with layers that are fully connected can become infeasible to train. Also trained with the Backpropagation algorithm, CNNs [36] are common for image processing tasks and reduce the number of parameters to be learned by limiting the number of connections of the neurons in the hidden layer to only some of the input neurons (i.e., a local area of the input image). A hidden layer (in this case, also called a *convolutional layer* – see Figure 1a) is composed by several groups of neurons. The weights of all neurons in a group are shared. Each group is generally composed by as many neurons as needed to cover the entire image. This way, it is as if each group of neurons in the hidden layer calculated a convolution of the image with their weights, resulting in a "processed" version of the image. We call this convolution a *feature*.

Commonly, *pooling* is applied to the resulting filtered images. The tecnique allows for achieving some translation invariance of the learned features. The groups (containing a newly processed version of the input image) are divided in chunks (e.g., 2×2) and their maximum value is taken. This results in yet another version of the input image, now smaller than the original size (in the example, 1/4 of the size of the group).

These steps can be repeatedly applied as many times as desired: a new convolutional layer can be applied on the pooled layer, followed by another pooling layer, and so forth. Finally, when the layers become small enough, it is common to have fully connected layers before the output layer.

Tiled Convolutional Neural Network The usage of shared weights in a group allow for the translation invariance of the learned features. However, "it prevents the pooling units from capturing more complex invariances, such as scale and rotation invariance" [48]. To solve this problem, Tiled CNNs allow for a group to be divided into subgroups called *tiles*, each of which can have separate weights. A parameter k defines how many tiles each group has: neurons that are exactly k steps away from each other share the same weights.

Fully Convolutional Networks (FCN) While the pooling operation performed by CNNs makes sense for object recognition tasks, because it has the advantage of achieving some robustness to small shifts of the learned features, it is not suited for tasks like Semantic Segmentation, where the goal is to segment the pixels of the image according to the objects that they refer to. FCNs [42] allow for the input and output layers to have the same dimensions by introducing "a decoder stage that is consisted of upsampling, convolution, and rectified linear units layers, to the CNN architecture" [47].

Recurrent Neural Network (RNN) When the network has loops, it is called a RNN. It is possible to adapt the Backpropagation algorithm to train a recurrent network, by "unfolding" the network through time and constraining some of the connections to always hold the same weights [51].

Long Short-Term Memory (LSTM) One problem that arises from the unfolding of an RNN is that the gradient of some of the weights starts to become too small or too large if the network is unfolded for too many time steps. This is called the *vanishing gradients* problem [7]. A type of network architecture that solves this problem is the LSTM [23]. In a typical implementation, the hidden layer is replaced by a complex block (see Figure 1c) of computing units composed by gates that trap the error in the block, forming a so-called "error carrousel".

2 Literature Review

Independently of Deep Learning, analysis of Time-Series data have been a popular subject of interest in other fields such as Economics, Engineering and Medicine. Traditional techniques on manipulating such data can be found in [21], and the application of traditional ANN techniques on this kind of data is described in [5].

Most work using ANN to manipulate Time-Series data focuses on modeling and forecasting. As an early attempt on using ANN for such tasks, [9] modelled flour prices over the range of 8 years. Still in the 90's, [29] delineated eight steps on "designing a neural network forecast model using economic time series data". More recent approaches include usage of Elman RNNs to predict chaotic Time-Series [10], employing ANN ensemble methods for forecasting Internet traffic [12], using simple Multilayer Perceptrons for modeling the amount of littering in the North Sea [53], and implementing in FPGA a prediction algorithm using Echo State Networks for "exploiting the inherent parallelism of these systems" [3].

Hybrid approaches to Time-Series analysis utilizing ANN are not uncommon. [31] presents a model for Time-Series forecasting using ANN and ARIMA models, and [15] applies the same kinds of models to water quality time series prediction. In still other examples of the same ideas, [30] compares the performance of ARIMA models and ANNs to make short-term predictions on photovoltaic power generators, while [38] compares both models with the performance of Multivariate Adaptive Regression Splines. [13] performs Time-Series forecasting by using a hybrid fuzzy model: while the Fuzzy C-means method is utilized for fuzzification, ANN are employed for defuzzification. Finally, [24] forecasts the speed of the wind using a hybrid of Support Vector Machines, Ensemble Empirical Mode Decomposition and Partial Autocorrelation Function.

Despite being relatively new, the field of Deep Learning has attracted a lot of interest in the past few years. A very thorough review of the entire history of developments that led the field to its current state can be found in [52], while a higher focus on the novelties from the last decade is given in [4]. We proceed to a review of the applications of Deep Learning to Time-Series data.

Classification The task of Classification of any type of data has benefited by the advent of CNNs. Previously existing methods for classification generally relied on the usage of domain specific features normally crafted manually by human experts. Finding the best features was the subject of a lot of research and the performance of the classifier was heavily dependent on their quality. The advantage of CNNs is that they can learn such features by themselves, reducing the need for human experts [34].

An example of the application of such unsupervised feature learning for the classification of audio signals is presented in [37]. In [1], the features learned by the CNN are used as input to a Hidden Markov Model, achieving a drop at the error rate of over 10%. The application of CNNs in these works presuppose the constraint that the Time-Series is composed of only one channel. An architecture that solves this constraint is presented in [60].

In [18] the performance of CNNs is compared with that of LSTM for the classification of Visual and Haptic Data in a robotics setting, and in [28] the signals produced by wearable sensors are transformed into images so that Deep CNNs can be used for classification.

Relevant to Tiled CNNs was the development of Independent Component Analysis (ICA) [27]. Several alternative methods for calculating independent components can be found in the literature (e.g., [17], [55] or [25]). Tiled CNNs are normally trained with a variation of such technique that looses the assumption that each component is statistically independent and tries to find a *topographic order* between them: the Topographic ICA [26].

Forecasting Several different Deep Learning approaches can be found in the literature for performing Forecasting tasks. For example, Deep Belief Networks are used in the work of [33] along with RBM. [58] also compares the performance of Deep Belief Networks with that of Stacked Denoising Autoencoders. This last type of network is also employed by [50] to predict the temperature of an indoor environment. Another application of Time-Series forecasting can be found in [43], which uses Stacked Autoencoders to predict the flow of traffic from a Big Data dataset.

A popular application to the task of Time-Series prediction is on Weather Forecasting. In [41], some preliminary predictions on weather data provided by The Hong Kong Observatory are made through the usage of Stacked Autoencoders. In a follow up work, the authors use similar ideas to perform predictions on Big Data [40]. Instead of Autoencoders, [20] uses Deep Belief Networks for constructing a hybrid model in which the ANN models the joint distribution between the weather predictors variables.

Anomaly Detection Work applying Deep Learning techniques to Anomaly Detection detection of Time-Series data is not very abundant in the literature. It is still difficult to find works such as [16], that uses Stacked Denoising Autoencoders to perform Anomaly Detection of trajectories obtained from low level tracking algorithms.

However, there are many similarities between Anomaly Detection and the previous two tasks. For example, identifying an anomaly could be transformed into a Classification task, as was done in [35]. Alternatively, detecting an anomaly could be considered the same as finding regions in the Time-Series for which the forecasted values are too different from the actual ones.

3 Deep Learning for Time-Series Modeling

In this section the work presented in [47] is reviewed. As discussed above, FCNs are a modification of the CNN architecture that, as required by some Time-Series related problems, allows for the input and output signals to have the same dimensions.

Mittelman [47] argues that the architecture of the FCN resembles the application of a wavelet transform, and that for this reason, it can present strong variations when the input signal is subject to small translations. To solve this problem, and inspired by the undecimeated wavelet transform, which is translation invariant, they propose the Undecimated Fully Convolutional Neural Network (UFCNN), also translation invariant.

The only difference between an FCN and an UFCNN is that the UFCNN removes both the upsampling and pooling operators from the FCN architecture. Instead, the “filters at the l^{th} resolution level are upsampled by a factor of 2^{l-1} along the time dimension”. See Figure 2b for a graphical representation of the proposed architecture.

The performance of the UFCNN is tested in three different experiments. In the first experiment, “2000 training sequences, 50 validation sequences and 50 testing sequences, each of length 5000 time-steps” are automatically generated by a probabilistic algorithm. The values of the Time-Series represent the position of a target object moving inside a bounded square. The performance of the UFCNN in estimating the position of the object at each time-step is compared to that of a FCN, a LSTM, and a RNN, and the UFCNN does perform better in most cases.

In a second experiment, the “MUSE” and “NOTTINGHAM” datasets[1] area used. The goal is to predict the values of the Time-Series in the next time-step. In both cases, the UFCNN outperforms the competing networks: a RNN, a Hessian-Free optimization-RNN [45], and a LSTM.

Finally, the third experiment uses a trading dataset[2], where the goal is, given only information about the past, to predict the set of actions that would “maximize the profit”. In a comparison to a RNN, the UFCNN again yielded the best results.

[1] available at http://www-etud.iro.umontreal.ca/ boulanni/icml2012
[2] available at http://www.circulumvite.com/home/trading-competition

4 Deep Learning for Time-Series Classification

Wang and Oates [59] presented an approach for Time-Series Classification using CNN-like networks. In order to benefit from the high accuracy that CNNs have achieved in the past few years on image classification tasks, the authors propose the idea of transforming a Time-Series into an image.

Two approaches are presented. The first one generates a Gramian Angular Field (GAF), while the second generates a Markov Transition Field (MTF). In both cases, the generation of the images increases the size of the Time-Series, making the images potentially prohibitively large. The authors therefore propose strategies to reduce their size without loosing too much information. Finally, the two types of images are combined in a two-channel image that is then used to produce better results than those achieved when using each image separately. In the next sections, GAF and MTF are described.

In the equations below, we suppose that $m = 1$. The Time-Series is therefore composed by only real-valued observations, such that referring to $x^{(i)} \in X$ is the same as referring to $\mathbf{x}^{(i)} \in X$.

4.1 Gramian Angular Field

The first step on generating a GAF is to rescale the entire Time-Series into values between $[-1, 1]$. In the equation 1, $max(X)$ and $min(X)$ represent the maximum and minimum real-values present in the Time-Series X:

$$\tilde{x}^{(i)} = \frac{(x^{(i)} - max(X)) + (x^{(i)} - max(X))}{max(X) - min(X)} \tag{1}$$

The next step is to recode the newly created Time-Series $\tilde{X}$ into polar coordinates. The angle is encoded by $x^{(i)}$ and the radius is encoded by the the time stamp i.

Notice that, because the values $x^{(i)}$ were rescaled, no information is lost by the usage of $arccos(\tilde{x}^{(i)})$ in 2.

$$\begin{cases} \phi = arccos(\tilde{x}^{(i)}), & -1 \leq \tilde{x}^{(i)} \leq 1, \tilde{x}^{(i)} \in \tilde{X} \\ r = \frac{i}{N}, & i \in \mathbb{N} \end{cases} \tag{2}$$

Finally, the GAF is defined as follows:

$$G = \begin{bmatrix} cos(\phi_1 + \phi_1) & \cdots & cos(\phi_1 + \phi_n) \\ cos(\phi_2 + \phi_1) & \cdots & cos(\phi_2 + \phi_n) \\ \vdots & \ddots & \vdots \\ cos(\phi_n + \phi_1) & \cdots & cos(\phi_n + \phi_n) \end{bmatrix} \tag{3}$$

Here, some information is lost by the fact that ϕ no more belongs to the interval $[0, \pi]$. When trying to recover the Time-Series from the image, there may be some errors introduced.

4.2 Markov Transition Field

The creation of the Markov Transition Field is based on the ideas proposed in [8] for the definition of the so-called Markov Transition Matrix (MTM).

For a Time-Series X, the first step is defining Q quantile bins. Each $x^{(i)}$ is then assigned to the corresponding bin q_j. The Markov Transition Matrix is the matrix W composed by elements w_{ij} such that $\sum_j w_{ij} = 1$ and w_{ij} corresponds to the normalized "frequency with which a point in the quantile q_j is followed by a point in the quantile q_i." This is a $Q \times Q$ matrix.

The MTF is the $n \times n$ matrix M. Each pixel of M contains a value from W. The value in the pixel ij is the likelihood (as calculated when constructing W) of going from the bin in which the pixel i is to the bin in which the pixel j is:

$$M = \begin{bmatrix} w_{ij|x_1 \in q_i, x_1 \in q_j} & \cdots & w_{ij|x_1 \in q_i, x_n \in q_j} \\ w_{ij|x_2 \in q_i, x_1 \in q_j} & \cdots & w_{ij|x_2 \in q_i, x_n \in q_j} \\ \vdots & \ddots & \vdots \\ w_{ij|x_n \in q_i, x_1 \in q_j} & \cdots & w_{ij|x_n \in q_i, x_n \in q_j} \end{bmatrix} \tag{4}$$

4.3 Performing Classification with the Generated Images

The authors use Tiled CNNs to perform classifications using the images. In the reported experiments, both methods are assessed separately in 12 "hard" datasets "on which the classification error rate is above 0.1 with the state-of-the-art SAX-BoP approach" [39], which are *50Words*, *Adiac*, *Beef*, *Coffee*, *ECG200*, *Face (all)*, *Lightning-2*, *Lightning-7*, *OliveOil*, *OSU Leaf*, *Swedish Leaf* and *Yoga* [11]. The authors then suggest the usage of both methods as "colors" of the images. The performance of the resulting classifier is competitive against many of the state-of-the-art classifiers, which are also reported by the authors.

5 Deep Learning for Time-Series Anomaly Detection

Anomaly Detection can be easily transformed into a task where the goal is to model the Time-Series and, given this model, find regions where the predicted values are too different from the actual ones (or, in other words, where the probability of the observed region is too low). This is the idea implemented by the paper reviewed in this section [44].

Through the learned model, not all m input variables need to be predicted. The learned model predicts, at any given time-step, l vectors with d input variables, where $1 \leq d \leq m$.

The modeling of the Time-Series is done through the application of a Stacked LSTM architecture. The network has m input neurons (one for each input variable) and $d \times l$ output neurons (one neuron for each one of the d predicted variables of the l vectors that are predicted at a time-step). The hidden layers are composed by LSTM units that are "fully connected through recurrent connections". Additionally, any hidden unit is fully connected to all units in the hidden layer above it. Figure 2a sketches the proposed architecture.

For each one of the predictions and each one of the d predicted variables, a prediction error is calculated. The errors are then used to fit a multivariate Guassian distribution, and a probability $p^{(t)}$ is assigned to each observation. Any observation whose probability $p^{(t)} < \tau$ is treated as an anomaly.

The approach was tested in four real-world datasets. One of them, called *Multi-sensor engine data* is not publicly available. The other three datasets (*Electrocardiograms (ECGs)*, *Space Suttle Marotta valve time series*, and *Power demand dataset*) are available for download[3]. The results demonstrated a significant improvement in capturing long-term dependencies when compared to simpler RNN-based implementations.

6 Conclusion

When applying Deep Learning, one seeks to stack several independent neural network layers that, working together, produce better results than the already existing shallow structures. In this paper, we have reviewed some of these modules, as well the recent work that has been done by using them, found in the literature. Additionally, we have discussed some of the main tasks normally performed when manipulating Time-Series data using deep neural network structures.

Finally, a more specific focus was given on one work performing each one of these tasks. Employing Deep Learning to Time-Series analysis has yielded results in these cases that are better than the previously existing techniques, which is an evidence that this is a promising field for improvement.

Acknowledgments. I would like to thank Ahmed Sheraz and Mohsin Munir for their guidance and contribution to this paper.

References

1. Abdel-Hamid, O., Mohamed, A.r., Jiang, H., Penn, G.: Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In: Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on. pp. 4277–4280. IEEE (2012)
2. Akmeliawati, R., Ooi, M.P.L., Kuang, Y.C.: Real-time malaysian sign language translation using colour segmentation and neural network. In: Instrumentation and Measurement Technology Conference Proceedings, 2007. IMTC 2007. IEEE. pp. 1–6. IEEE (2007)
3. Alomar, M.L., Canals, V., Perez-Mora, N., Martínez-Moll, V., Rosselló, J.L.: Fpga-based stochastic echo state networks for time-series forecasting. Computational Intelligence and Neuroscience 501, 537267 (2015)
4. Arel, I., Rose, D.C., Karnowski, T.P.: Deep machine learning-a new frontier in artificial intelligence research [research frontier]. Computational Intelligence Magazine, IEEE 5(4), 13–18 (2010)

[3] available at http://www.cs.ucr.edu/ eamonn/discords/

5. Azoff, E.M.: Neural network time series forecasting of financial markets. John Wiley & Sons, Inc. (1994)
6. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., et al.: Greedy layer-wise training of deep networks. Advances in neural information processing systems 19, 153 (2007)
7. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. Neural Networks, IEEE Transactions on 5(2), 157–166 (1994)
8. Campanharo, A., Sirer, M.I., Malmgren, R.D., Ramos, F.M., Amaral, L.A.: Duality between time series and networks. PloS one 6(8), e23378 (2011)
9. Chakraborty, K., Mehrotra, K., Mohan, C.K., Ranka, S.: Forecasting the behavior of multivariate time series using neural networks. Neural networks 5(6), 961–970 (1992)
10. Chandra, R., Zhang, M.: Cooperative coevolution of elman recurrent neural networks for chaotic time series prediction. Neurocomputing 86, 116–123 (2012)
11. Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G.: The ucr time series classification archive (July 2015), `www.cs.ucr.edu/~eamonn/time_series_data/`
12. Cortez, P., Rio, M., Rocha, M., Sousa, P.: Multi-scale internet traffic forecasting using neural networks and time series methods. Expert Systems 29(2), 143–155 (2012)
13. Egrioglu, E., Aladag, C.H., Yolcu, U.: Fuzzy time series forecasting with a novel hybrid approach combining fuzzy c-means and neural networks. Expert Systems with Applications 40(3), 854–857 (2013)
14. Esling, P., Agon, C.: Time-series data mining. ACM Computing Surveys (CSUR) 45(1), 12 (2012)
15. Faruk, D.Ö.: A hybrid neural network and arima model for water quality time series prediction. Engineering Applications of Artificial Intelligence 23(4), 586–594 (2010)
16. Feng, W., Han, C.: A novel approach for trajectory feature representation and anomalous trajectory detection. In: Information Fusion (Fusion), 2015 18th International Conference on. pp. 1093–1099. IEEE (2015)
17. Gao, P., Woo, W., Dlay, S.: Non-linear independent component analysis using series reversion and weierstrass network. IEE Proceedings-Vision, Image and Signal Processing 153(2), 115–131 (2006)
18. Gao, Y., Hendricks, L.A., Kuchenbecker, K.J., Darrell, T.: Deep learning for tactile understanding from visual and haptic data. arXiv preprint arXiv:1511.06065 (2015)
19. Graves, A., et al.: Supervised sequence labelling with recurrent neural networks, vol. 385. Springer (2012)
20. Grover, A., Kapoor, A., Horvitz, E.: A deep hybrid model for weather forecasting. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 379–386. ACM (2015)
21. Hamilton, J.D.: Time series analysis, vol. 2. Princeton university press Princeton (1994)
22. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural computation 18(7), 1527–1554 (2006)
23. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9(8), 1735–1780 (1997)
24. Hu, J., Wang, J., Zeng, G.: A hybrid forecasting approach applied to wind speed time series. Renewable Energy 60, 185–194 (2013)
25. Hyvärinen, A.: Fast and robust fixed-point algorithms for independent component analysis. Neural Networks, IEEE Transactions on 10(3), 626–634 (1999)

26. Hyvärinen, A., Hoyer, P., Inki, M.: Topographic independent component analysis. Neural computation 13(7), 1527–1558 (2001)
27. Hyvärinen, A., Oja, E.: Independent component analysis: algorithms and applications. Neural networks 13(4), 411–430 (2000)
28. Jiang, W., Yin, Z.: Human activity recognition using wearable sensors by deep convolutional neural networks. In: Proceedings of the 23rd Annual ACM Conference on Multimedia Conference. pp. 1307–1310. ACM (2015)
29. Kaastra, I., Boyd, M.: Designing a neural network for forecasting financial and economic time series. Neurocomputing 10(3), 215–236 (1996)
30. Kardakos, E.G., Alexiadis, M.C., Vagropoulos, S., Simoglou, C.K., Biskas, P.N., Bakirtzis, A.G., et al.: Application of time series and artificial neural network models in short-term forecasting of pv power generation. In: Power Engineering Conference (UPEC), 2013 48th International Universities'. pp. 1–6. IEEE (2013)
31. Khashei, M., Bijari, M.: A novel hybridization of artificial neural networks and arima models for time series forecasting. Applied Soft Computing 11(2), 2664–2675 (2011)
32. King, S.Y., Hwang, J.N.: Neural network architectures for robotic applications. Robotics and Automation, IEEE Transactions on 5(5), 641–657 (1989)
33. Kuremoto, T., Kimura, S., Kobayashi, K., Obayashi, M.: Time series forecasting using a deep belief network with restricted boltzmann machines. Neurocomputing 137, 47–56 (2014)
34. Längkvist, M.: Modeling time-series with deep networks (2014)
35. Längkvist, M., Karlsson, L., Loutfi, A.: Sleep stage classification using unsupervised feature learning. Advances in Artificial Neural Systems 2012, 5 (2012)
36. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11), 2278–2324 (1998)
37. Lee, H., Pham, P., Largman, Y., Ng, A.Y.: Unsupervised feature learning for audio classification using convolutional deep belief networks. In: Advances in neural information processing systems. pp. 1096–1104 (2009)
38. Lin, C.J., Chen, H.F., Lee, T.S.: Forecasting tourism demand using time series, artificial neural networks and multivariate adaptive regression splines: evidence from taiwan. International Journal of Business Administration 2(2), p14 (2011)
39. Lin, J., Khade, R., Li, Y.: Rotation-invariant similarity in time series using bag-of-patterns representation. Journal of Intelligent Information Systems 39(2), 287–315 (2012)
40. Liu, J.N., Hu, Y., He, Y., Chan, P.W., Lai, L.: Deep neural network modeling for big data weather forecasting. In: Information Granularity, Big Data, and Computational Intelligence, pp. 389–408. Springer (2015)
41. Liu, J.N., Hu, Y., You, J.J., Chan, P.W.: Deep neural network based feature representation for weather forecasting. In: Proceedings on the International Conference on Artificial Intelligence (ICAI). p. 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp) (2014)
42. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. arXiv preprint arXiv:1411.4038 (2014)
43. Lv, Y., Duan, Y., Kang, W., Li, Z., Wang, F.Y.: Traffic flow prediction with big data: a deep learning approach. Intelligent Transportation Systems, IEEE Transactions on 16(2), 865–873 (2015)
44. Malhotra, P., Vig, L., Shroff, G., Agarwal, P.: Long short term memory networks for anomaly detection in time series. In: European Symposium on Artificial Neural Networks. vol. 23

45. Martens, J., Sutskever, I.: Learning recurrent neural networks with hessian-free optimization. In: Proceedings of the 28th International Conference on Machine Learning (ICML-11). pp. 1033–1040 (2011)
46. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics 5(4), 115–133 (1943)
47. Mittelman, R.: Time-series modeling with undecimated fully convolutional neural networks. arXiv preprint arXiv:1508.00317 (2015)
48. Ngiam, J., Chen, Z., Chia, D., Koh, P.W., Le, Q.V., Ng, A.Y.: Tiled convolutional neural networks. In: Advances in Neural Information Processing Systems. pp. 1279–1287 (2010)
49. Rojas, R.: Neural networks: a systematic introduction. Springer Science & Business Media (2013)
50. Romeu, P., Zamora-Martínez, F., Botella-Rocamora, P., Pardo, J.: Time-series forecasting of indoor temperature using pre-trained deep neural networks. In: Artificial Neural Networks and Machine Learning–ICANN 2013, pp. 451–458. Springer (2013)
51. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Cognitive modeling 5, 3 (1988)
52. Schmidhuber, J.: Deep learning in neural networks: An overview. Neural Networks 61, 85–117 (2015)
53. Schulz, M., Matthies, M.: Artificial neural networks for modeling time series of beach litter in the southern north sea. Marine environmental research 98, 14–20 (2014)
54. Stutz, D.: Understanding convolutional neural networks (2014)
55. Suzuki, K., Kiryu, T., Nakada, T.: Fast and precise independent component analysis for high field fmri time series tailored using prior information on spatiotemporal structure. Human brain mapping 15(1), 54–66 (2002)
56. Szegedy, C., Toshev, A., Erhan, D.: Deep neural networks for object detection. In: Advances in Neural Information Processing Systems. pp. 2553–2561 (2013)
57. Tesauro, G.: Practical issues in temporal difference learning. Springer (1992)
58. Turner, J.T.: Time series analysis using deep feed forward neural networks. Ph.D. thesis, University of Maryland, Baltimore County (2014)
59. Wang, Z., Oates, T.: Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. In: Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence (2015)
60. Zheng, Y., Liu, Q., Chen, E., Ge, Y., Zhao, J.L.: Time series classification using multi-channels deep convolutional neural networks. In: Web-Age Information Management, pp. 298–310. Springer (2014)