Deep neural networks for time series prediction with applications in ultra-short-term wind forecasting

Mladen Đalto,
University of Zagreb, Faculty of Electrical Engineering and Computing,
E-mail: mladen.dalto@fer.hr.

Abstract—**The aim of this paper is to present deep neural network architectures and algorithms and explore their use in time series prediction. Existing and novel input variable selection algorithms and deep neural networks are applied for ultra-short-term wind prediction. Since gradient-based optimization starting from random initialization often appears to get stuck in poor solutions, recent research effort aimed at training methods for such deep networks is summarized. Shallow and deep neural networks coupled with two input variable selection algorithms are compared on a ultra-short-term wind prediction task. Initial results show that deep neural networks outperform shallow ones. Depth adds additional computational cost and input variable selection use reduces it.**

Keywords—Deep Neural Networks, Input Variable Selection, ultra-short-term, wind forecasting

1. Introduction

Artificial neural networks with backpropagation learning algorithm are widely used in solving various classification and prediction problems. Until recently the depth of a practical neural network (NN) was mostly limited to one hidden layer. Deep Neural Networks (DNN) have many layers of nonlinear functions, unlike shallow networks which have one hidden layer. Many layers allow them to compactly represent highly non-linear and highly-varying functions. Because neural network learning is in general not convex (and hence hard), many problems arise in their training. This paper explores Input Variable Selection (IVS), the novel methods based on intermediate representation (feature) learning and their potential for time series prediction with applications in ultra-short-term wind forecasting. Recently deep architectures have reported state-of-the-art performance across fields such as object recognition [25], speech recognition [2], natural language processing [15], physiological affect modelling [29], etc. Depth is motivated both biologically and by circuit complexity theory which suggests that deep architectures can be much more efficient (sometimes exponentially) than shallow ones in terms of computational elements and parameters required to represent some functions. Unfortunately, training deep architectures is a difficult task and classical methods that have proved effective when applied to shallow architectures are not as efficient when adapted to deep architectures. A number of papers highlight that performance of neural networks is heavily dependent on the input representation, or layers of intermediate representations of the data [7], [5]. In this work unsupervised pretraining procedures are presented. This work is restricted to presenting the deep neural network architectures such as Auto-Encoders (AE) [36], Denoising Auto-Encoders (DAE) [40], Stacked Auto-Encoders (SAE) [9], Convolutional Neural Networks (CNN) [27]; some aspects of their training and other time series prediction specifics. Input data in earlier mentioned state-of-the-art fields is already temporally (1-D time series) correlated, spatially correlated (images) or can be transformed to have such properties, e.g., sound, by using time-frequency analysis.
The multivariate time series data set often does not exhibit such relationships among variables as they describe different phenomena. Selecting input variables and grouping them in a useful way is another topic of interest highly related to Input Variable Selection (IVS).
Section 2 describes some problems in neural network learning and some recent approaches. In Section 3 an introduction to DNN is given. IVS algorithms are described in Section 4. Section 5 gives an overview of neural network time series prediction and existing applications of deep learning in the field. Section 6 presents some results obtained with IVS and DNN. This work is concluded with Section 7.

2. Problems in neural network training and some recent approaches

Problems in neural network training are caused by many factors:

- **Non convex problem**:
  The problem of finding optimal parameters of a Neural Network (NN) is in general not convex, implying there is no guarantee of global solution.
- **Curse of dimensionality** phenomena arise when analysing and organizing data in high-dimensional spaces that do not occur in low-dimensional settings. When the dimensionality increases, the volume of the space increases so fast that the available data becomes sparse. This sparsity is problematic for any method, such as NN, that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality.
- **Vanishing gradient** is identified in [23] and affects many-layered feedforward networks, but also recurrent neural networks which can be considered as DNN [34]. Conventional Backpropagation (BP), however, suffers from a too long learning time because error signals ”flowing backwards” tend to vanish. As errors propagate from layer to layer, they shrink exponentially with the number of layers.
- **Hyper-parameter values selection** becomes harder with deep neural networks. Hyper-parameters associated with the neural network structure and the training algorithm need to be selected. When using gradient based optimization, e.g. Stohastic Gradient Descent (SGD) methods and feedforward neural networks, some of the hyper-parameters are: initial learning rate, learning rate schedule, mini-batch size in SGD, number of training iterations, number of hidden units, weight decay, sparsity of activation, neuron non-linearity, weights initialization scaling coefficient, preprocessing etc. There is no clear theoretical solution for values of any of these hyper-parameters, although they are topics of some

research papers [6], [11].

A theoretical algorithm in which hidden neurons are inserted one at a time, with no upper bound, can be characterized as a convex neural network optimization problem [10]. Algorithms for approximate or exact solutions to convex neural network problems exist. With approximate algorithm global optimum can not be guaranteed and exact algorithm is NP hard. When using a special training criteria (hinge loss), a bound on the size of the hidden layer is obtained. It is bounded by the training set size, which is not of practical use, although the authors claim that much of hidden layers can be removed because optimization with L1 regularization forces the weights to zero. With a finite number of non-zero weights this becomes an ordinary neural network.

In the case of highly varying functions, learning algorithms entirely based on local generalization are severely impacted by the curse of dimensionality [8]. Deep architectures address this issue with the use of distributed representations and weight sharing, and as such may constitute a tractable alternative.

Vanishing gradient problem was initially recognized for feed-forward and recurrent neural networks [34], and recently for deep architectures. This optimization difficulty has motivated the exploration of second-order optimization methods for deep architectures and recurrent networks [28]. Deep rectifier networks [20] seem to have no vanishing gradient effects due to using non-sigmoid activation function.

Guidelines for hyper-parameter selection are discussed in [6]. A genetic algorithm approach seems particularly interesting.

3. Deep neural network architectures

This paper focuses on architectures that are equivalent to Multi Layer Perceptoron (MLP) NN and CNN, that have a somewhat different structure. The difference between standard MLP and DNN is in the training procedure and criteria. Clasification of deep architectures by Y. LeCun and M.A. Ranzato is given in Figure 1, where architectures falling into neural networks category are noted green. In this Section neural network structures, without the Perceptron,

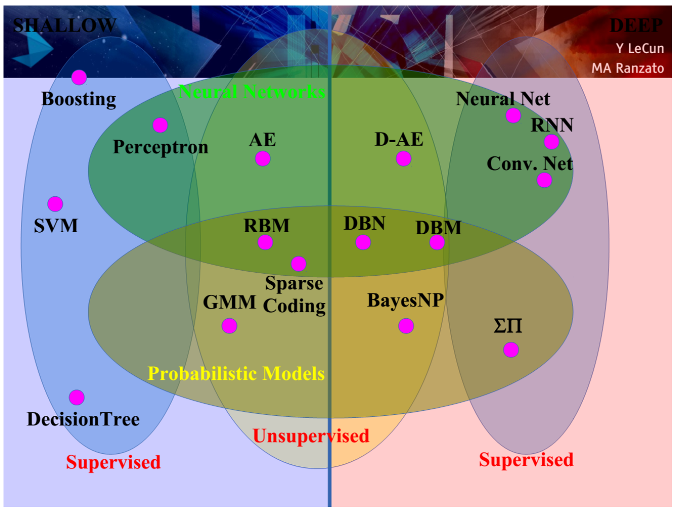

Figure 1: Clasification of deep architectures.

Recurrent Neural Network (RNN) and the probabilistic models, are presented.

A. Autoencoder

An autoencoder takes a preprocessed input vector (i.e. normalized to $[0, M]$ or $[-M, M]$ depending on the task at hand) $\mathbf{x} \in [-1,1]^{n(x)\times 1}$, and first maps it to a hidden representation $\mathbf{y} \in [-1,1]^{n(y)\times 1}$ through a deterministic mapping (encoding):

$$\mathbf{y} = f_e(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \tag{1}$$

parametrized by:

$$\mathbf{\Theta}_1 = \left[vec(\mathbf{W}_1)^T, \quad vec(\mathbf{b}_1)^T\right]^T \in \mathbb{R}^{n(\Theta_1)\times 1}, \tag{2}$$

where $vec(\cdot)$ is a linear transformation which converts the matrix into a column vector, and $n(\cdot)$ denotes the size of the argument.
$\mathbf{W}_1 \in \mathbb{R}^{n(y)\times n(x)}$ is the weight matrix and $\mathbf{b}_1 \in \mathbb{R}^{n(y)\times 1}$ is the bias vector.
The resulting latent representation $\mathbf{y}$ is then mapped back to a "reconstructed" vector $\hat{\mathbf{x}} \in [-1,1]^{n(x)\times 1}$ with a decoder:

$$\hat{\mathbf{x}} = f_d(\mathbf{W}_2\mathbf{y} + \mathbf{b}_2), \tag{3}$$

parametrized by:

$$\mathbf{\Theta}_2 = \left[vec(\mathbf{W}_2)^T, \quad vec(\mathbf{b}_2)^T\right]^T \in \mathbb{R}^{n(\Theta_2)\times 1}. \tag{4}$$

$\mathbf{W}_2 \in \mathbb{R}^{n(x)\times n(y)}$ is the weight matrix and $\mathbf{b}_2 \in \mathbb{R}^{n(x)\times 1}$ is the bias vector of the decoder.
In order to reduce the number of parameters sometimes tied weights ($\mathbf{W}_2 = \mathbf{W}_1^T$) are used. Reducing the number of parameters can be beneficial because optimizing less parameters means we can use less data and possibly exploit sparse datasets. This constraint requires slight modifications to the standard BP derivation. Putting it together results in an MLP:

$$\hat{\mathbf{x}} = g(\mathbf{x}, \mathbf{\Theta}_1, \mathbf{\Theta}_2) = f_d\left(\mathbf{W}_2 f_e\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right) + \mathbf{b}_2\right). \tag{5}$$

Each training sample $\mathbf{x}(i)$ is mapped to corresponding intermediate representation $\mathbf{y}(i)$ and reconstruction $\hat{\mathbf{x}}(i)$. The parameters of this model ($\mathbf{\Theta}_1$ and $\mathbf{\Theta}_2$) are optimized to minimize the average reconstruction error:

$$\begin{aligned}\mathbf{\Theta}_1^*, \mathbf{\Theta}_2^* &= \underset{\mathbf{\Theta}_1,\mathbf{\Theta}_2}{\arg\min} \frac{1}{N}\sum_{i=1}^{N} L\left(\mathbf{x}(i), \hat{\mathbf{x}}(i)\right) \\ &= \underset{\mathbf{\Theta}_1,\mathbf{\Theta}_2}{\arg\min} \frac{1}{N}\sum_{i=1}^{N} L\left(\mathbf{x}(i), g(\mathbf{x}(i), \mathbf{\Theta}_1, \mathbf{\Theta}_2)\right),\end{aligned} \tag{6}$$

where L is the loss function, e.g. Euclidean distance $L\left(\mathbf{x}(i), \hat{\mathbf{x}}(i)\right) = \|\mathbf{x}(i) - \hat{\mathbf{x}}(i)\|^2$, and N is the number of training samples. If $n(\mathbf{y}) \geq n(\mathbf{x})$ autoencoder can learn trivial identity mapping, which is not a interesting representation. A compression encoding ($n(\mathbf{y}) < n(\mathbf{x})$) can be use to avoid learning the identity mapping. Although sometimes useful, it is usually avoided because it can significantly constrain the representations. To avoid learning trivial identity mapping, usually alongside the Euclidean distance, a regularization term $\sum_{k=1}^{n(\Theta_1)} \Theta_{1,k}^2 + \sum_{m=1}^{n(\Theta_2)} \Theta_{2,m}^2$ and a Kullback-Leibler (KL) divergence term are used [33]. KL divergence between a Bernoulli random variable with mean ρ and a Bernoulli random variable with mean $\hat{\rho}_j$ is used as a sparsity criteria:

$$\sum_{j=1}^{n(\mathbf{y})} KL\left(\rho\|\hat{\rho}_j\right) = \sum_{j=1}^{n(\mathbf{y})} \rho\log\frac{\rho}{\hat{\rho}_j} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_j}. \tag{7}$$

KL criteria is used to enforce sparse average activations $\hat{\rho}_j$ of the hidden layer units by setting small average activation (e.g. $\rho = 0.05$). Sparsity is motivated biologically and has been successfully used to learn useful intermediate representations. For logistic function activations $s(t) = \frac{1}{1+e^{-t}}$, average activation is calculated by:

$$\hat{\rho}_j = \frac{1}{N}\sum_{i=1}^{N} y_j(i). \quad (8)$$

When using hyperbolic tangent activation functions ($\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$) $\hat{\rho}_j$ has to be calculated in some other way. Approximate activation (8) measures how close the activations are to 0, which is not a stationary value for the hyperbolic tangent function. This suggests the need for a unified way to penalize activity e.g. enforce sparsity as in i.e. [35]. The training scheme of an AE is illustrated on Figure 2. Practical recommendations [6] are to take $n(\mathbf{y}) > n(\mathbf{x})$.

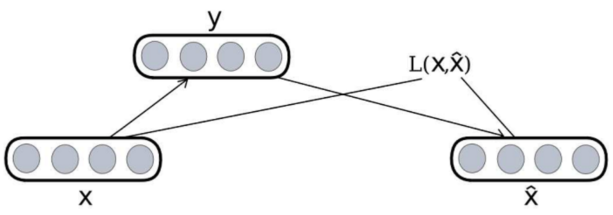

Figure 2: Training scheme of an AE. Image taken from [4].

The case of compressing ("bottleneck") AE ($n(\mathbf{y}) < n(\mathbf{x})$) and unconstrained weights was analysed in [12] with analytical results superior to those obtained by BP. Although this approach does not include regularization terms nor sparsity criteria alongside the Euclidean distance, and has a "bottleneck" hidden representation, it is still a valuable result worth re-analysing.

B. Denoising autoencoder

Denoising autoencoders [40] have the same structure as the autoencoders presented in subsection 3.1. A very notable difference is the input corruption process. The DAE is trained to reconstruct a clean "repaired" input $\mathbf{x}$ form a corrupted, partially destroyed $\tilde{\mathbf{x}}$. This is done by first corrupting the initial input $\mathbf{x}$ to get a partially destroyed version $\tilde{\mathbf{x}}$ by means of a stochastic mapping. Corruption is usually performed on a random proportion of the input data by zeroing (e.g. forcing values to 0) a fixed number of input vector $\mathbf{x}$ components, while the others are left untouched. All the information about the chosen components is thus removed from that particular input pattern, and the autoencoder will be trained to "fill-in" these artificially introduced "blanks". Loss function L in training criteria (6) can be slightly modified to $L(\tilde{\mathbf{x}}(i), \mathbf{x}(i)) = \|\tilde{\mathbf{x}}(i) - \mathbf{x}(i)\|^2$ or a cross-entropy loss function can be used. The training procedure of an SAE is illustrated on Figure 3.

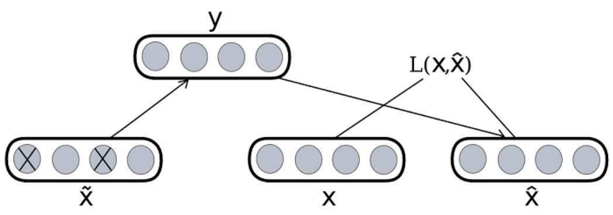

Figure 3: Training scheme of an D-AE. Image taken from [4].

Note that in this way, the DAE cannot learn the identity, unlike the basic AE, thus removing the constraint $n(\mathbf{y}) < n(\mathbf{x})$ or the need to specifically regularize to avoid such a trivial solution.
In [40] it is stated that the corruption+denoising training works remarkably well as an initialization step for DNN. This principle can be used to train and stack autoencoders to initialize a deep neural network. Authors also conclude that unsupervised initialization of layers with an explicit denoising criterion helps to capture interesting structure in the input distribution.

C. Stacked autoencoders

Stacked Autoencoder were one of the first methods for building deep architectures [9], along with a probabilistic model: Restricted Boltzmann Machines [22]. SAE is a deep neural network trained following the deep learning scheme: an unsupervised layer-wise pre-training before a fine-tuning supervised stage. Once a basic or denoising AE is trained, its internal representation ($\hat{\mathbf{h}}^{(1)}(\mathbf{x}) = f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$ or $\hat{\mathbf{h}}^{(1)}(\tilde{\mathbf{x}}) = f_1(\mathbf{W}_1\tilde{\mathbf{x}} + \mathbf{b}_1)$) can be used as input for training a second AE etc. In every step, trained parameters $\mathbf{\Theta}_1^*$ are kept and $\mathbf{\Theta}_2^*$ discarded. This is because we only need the first layer to obtain $\mathbf{y}$. Figure 4 illustrates this iterative training procedure. The SAE can then be fine-tuned with respect to a supervised training criterion (adding a predictive output layer on top), using BP to compute gradient on parameters of all layers.

D. Convolutional Neural Networks

CNN is introduced in [27] by Yan LeCun and others. This CNN introduction is mostly taken from [17], an excellent tutorial web page on deep learning; specifically, a CNN chapter based on [27]. CNN are variants of MLPs with a special structure which is inspired from biology. From early work on the cat's visual cortex [24], we know there exists a complex arrangement of cells within the visual cortex. These cells are sensitive to small sub-regions of the input space, called a receptive field, and are tiled in such a way as to cover the entire visual field.
CNN filters, obtained by the convolution of inputs and weights, are local in input space and are thus better suited to exploit the strong, spatially local correlation present in natural images.
CNN exploit spatially local correlation (coherence) by enforcing a local connectivity pattern between neurons of adjacent layers. The input hidden units in the m-th layer are connected to a local subset of units in the $(m-1)$-th layer, which have spatially contiguous receptive fields. This is illustrated in Figure 5. Such architecture

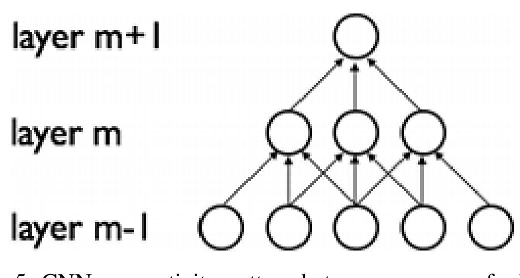

Figure 5: CNN connectivity pattern between neurons of adjacent layers. Image taken from [17].

confines the learnt "filters" to have a spatially local pattern. As shown above, stacking many such layers leads to "filters" which become increasingly "global" (i.e spanning a larger region of the input space). The unit in hidden layer $m+1$ can encode a non-linear feature of larger width.
In CNN, each sparse filter h_i is additionally replicated across the entire visual field. These "replicated" units form feature maps, which

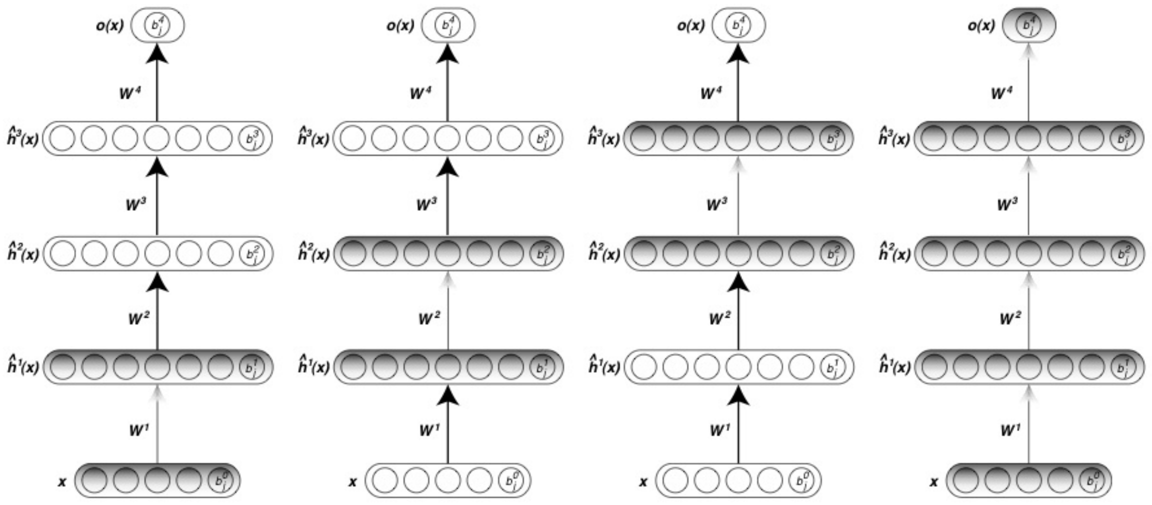

Figure 4: Training scheme of an SAE. Image taken from [26].

share the same parametrization, i.e. the same weight vector and the same bias (as illustrated in Figure 6). Weights of the same color

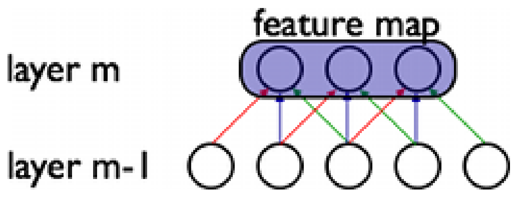

Figure 6: CNN weight sharing associated to a particular feature map. Image taken from [17].

are shared, i.e. are constrained to be identical. Gradient descent can still be used to learn such shared parameters and requires only a small change to the original algorithm. The gradient of a shared weight is simply the sum of the gradients of the parameters being shared. Replicating units in this way allows for features to be detected regardless of their position in the visual field. Additionally, weight sharing offers a very efficient way to do this, since it greatly reduces the number of free parameters to learn. By controlling model capacity, CNN tend to achieve better generalization on vision problems.

Conceptually, a feature map is obtained by convolving the input image with a linear filter, adding a bias term and then applying a non-linear function. If we denote the k-th feature map at a given layer as h^k, whose filters are determined by the weights W^k and bias b_k, then the feature map h^k is obtained as follows (for hyperbolic tangent non-linearities):

$$\mathbf{h}_{ij}^k = \tanh((\mathbf{W}^k * \mathbf{x})_{ij} + \mathbf{b}_k), \tag{9}$$

where convolution for 1-D signals is defined by:

$$o[n] = f[n]*g[n] = \sum_{u=-\infty}^{\infty} f[u]g[u-n] = \sum_{u=-\infty}^{\infty} f[n-u]g[u], \tag{10}$$

and for 2-D signals:

$$\begin{aligned} o[m,n] &= f[m,n] * g[m,n] \\ &= \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} f[u,v]g[u-m,v-n]. \end{aligned} \tag{11}$$

To form a richer representation of the data, hidden layers are composed of a set of multiple feature maps $\{h^{(k)}, k = 0, ..., K\}$. Weights W of this layer can be parametrized as a 4D tensor (destination feature map index, source feature map index, source vertical position index, source horizontal position index) and the biases b as a vector (one element per destination feature map index). Another important

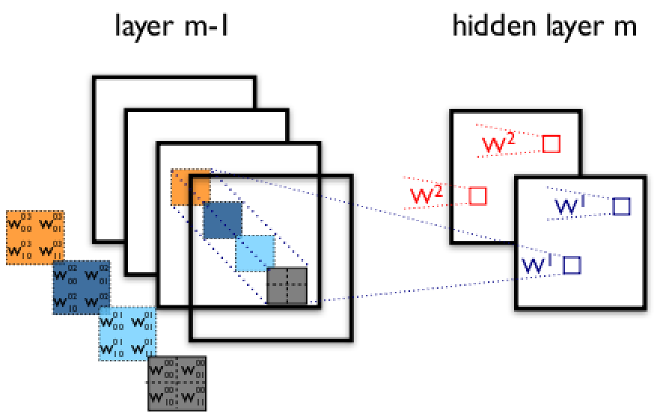

Figure 7: Example of the convolutional layer. Image taken from [17].

concept of CNN is that of max-pooling, which is a form of non-linear down-sampling. Max-pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region $N(l,n)$, outputs the maximum value:

$$h_{ln}^{m+1} = \max_{i,j \in N(l,n)} h_{ij}^m, \tag{12}$$

where integers l and n denote the rectangle. Max-pooling is useful because it reduces the computational complexity for upper layers and provides a form of translation invariance for image inputs. Moreover, weight sharing helps in reducing over-fitting due to the reduced number of trainable parameters.

CNN's are hierarchical models that alternate convolutional and pooling layers in order to process large input spaces in which a spatial or temporal relation among the inputs exists. On top of a CNN an MLP can be added to combine the features and perform prediction or classification. Figure 8 shows an example of the entire CNN architecture.

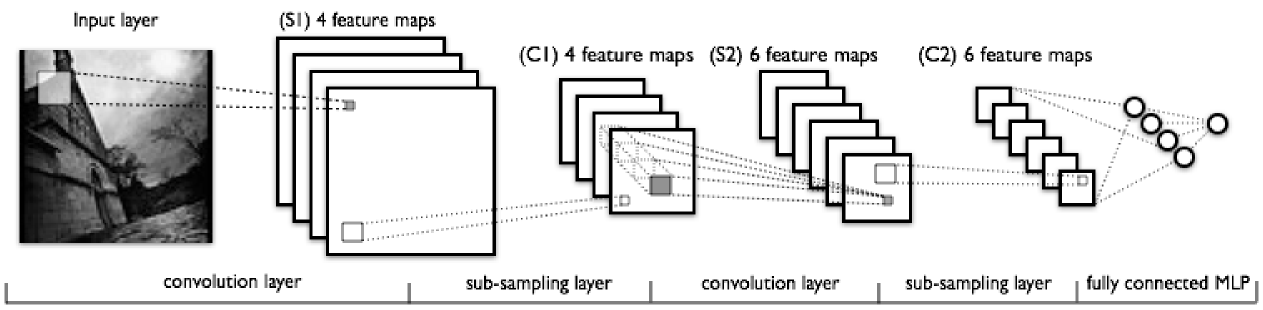

Figure 8: Example of a CNN. Image taken from [17].

4. Input Variable Selection

IVS is very important in identification of optimal structure for statistical models. Task of choosing input variables is common in the development of all statistical models and greatly depends on discovering connections between available input-output data for detection of suitable inputs (so-called predictors) for output prediction. In the case of NN, and other similar data based models, there is no assumption of a specific input-output mapping structure, so a general structure is assumed. In that case several problems occur:

- large number of input variables,
- correlation amongst available variables,
- uninformative variables.

Many IVS procedures do not give good results because linear relationships are assumed or because of redundancy in the input candidate set. Correlation measures comparison [14] motivates experimental work presented in Subsection 4.2.

A. Partial mutual information

Recently Mutual information (MI) is being used as a more suitable measure for IVS. Mutual information is defined in [16] as:

$$I(X,Y) = \int_Y \int_X p(x,y) \log\left(\frac{p(x,y)}{p(x)\,p(y)}\right) dx\,dy, \qquad (13)$$

where $p(x,y)$ represents joint density estimation and $p(x)$ and $p(y)$ marginal distributions of X and Y. In practical context probability distributions in (13) are rarely known. MI advantage is in the fact that it does not make assumptions about the structure of the analysed variables and is invariant to transformations and noise [30]. Naive application of MI cannot detect redundancy between candidate input variables and lacks a stopping criterion. Partial mutual information (PMI) is a nonlinear IVS algorithm that can completely or partially overcome these problems. PMI between the output Y and candidate C_j with preselected predictors $\mathbf{S}$ is defined as:

$$\begin{aligned} PMI(Y, C_j|\mathbf{S}) =& H(Y,\mathbf{S}) + H(\mathbf{S}, C_j) \\ &- H(\mathbf{S}) - H(Y,\mathbf{S},C_j), \end{aligned} \qquad (14)$$

where Y, C_j are random scalars, $\mathbf{S}$ a random vector and $H(\cdot)$ represents Shannon entropy:

$$H(X) = -\sum_i p(x_i) \log p(x_i). \qquad (15)$$

A detailed formulation of entropy for multiple arguments is provided in [18]. In a special case, when no predictors are preselected, PMI becomes $PMI(Y, C_j|\emptyset) = I(Y, C_j)$. Two variants of PMI were implemented and tested: one based on KDE from [30] and another based on kNN from [18]. In further work KDE based PMI is used. Consider a system with output Y, and with S and C_j as predictor candidates. If $I(Y;S) \geq I(Y;C_j)\,\forall j$, predictor S is selected as a first candidate because it introduces more information required to predict Y. Now we perform a second step of the algorithm with preselected S. The information contribution of any candidate C_j is represented by $I(C_j;Y|S)$.

B. Novel association measures and potential use for IVS

Correlation measures comparison [14] and [37] compare Pearson correlation, Spearman's Measure, Hoeffding's D, Brownian distance correlation and Maximal Information Coefficient. Brownian distance correlation [38] is highlighted as the one with more statistical power then others. Distance correlation was combined with the algorithm for PMI [31], but instead of MI, distance correlation is used. Distance correlation can be used to measure dependence between random vectors of arbitrary dimension, which is a great advantage. Preliminary results for the ultra-short wind data described in [3] are shown on Figure 9. Test example uses 2000 samples for IVS between 1512 candidates and 36 outputs.

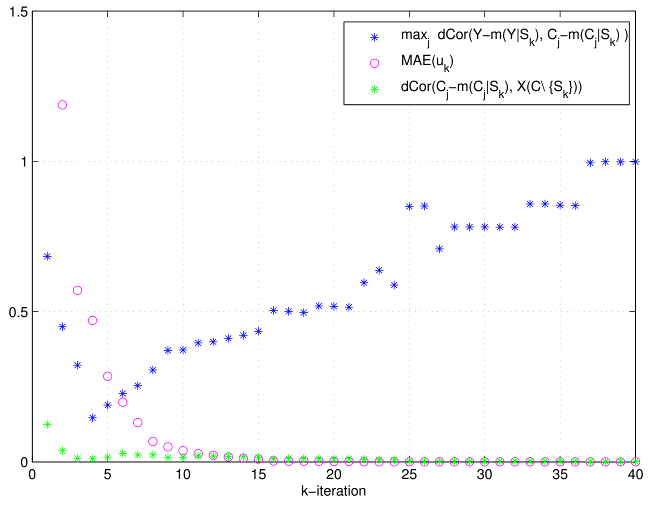

Figure 9: Example of distance correlation based IVS and stopping indicators with 40 iterations.

Current algorithm lacks a proper stopping criteria so it was limited to 40 iterations. This algorithm can be used in a scenario where inputs have different dimensions (e.g. images and signals) and arbitrary dimensioned outputs.
Another promising dependence measure is the HHG test [21] which has yet to be tested in the IVS setting.

5. Neural network time series prediction

Time series data is either measured real-world data or man-made data. Either way it has a temporal component. Analysis of such data has been the subject of active research for decades and is considered a challenging problem in data mining. If a time-series model, such as linear model, is assumed parameter estimation is performed. But almost all interesting time-series is nonlinear and linear models do not have the capacity to accurately model such complex data. Nonlinar systems are usually hard to model mathematically. Even if such models exist we still have to deal with noisy data, lack of (nonlinear) observability, etc.

A simple feedforward architecture is used for common Nonlinear Auto Regressive eXogenous (NARX) models which can describe any nonlinear process under some mild conditions. Using a purely feedforward network we ignore some of the temporal structure in time series data and gain in training and use simplicity. Input regression vector of a NARX model can be formed from past inputs, past model outputs and past system outputs. Input regression vector increases the number of inputs compared to initial multivariate inputs. This is necessary in order to capture dependencies between past and future system states. Inputs formed in such a way may include unnecessary past variables. For this and other reasons IVS is used to select a proper subset of input regressor vector elements.

A. Deep neural networks time series examples

There is still little literature on deep neural networks and time-series prediction. Few papers on time-series prediction or classification with deep neural networks stand out. CNN's are used for speech recognition [1], congestive heart faliure detection [41], psychological models of affect [29], etc. In [41] multi-channels deep convolutional neural networks were used. This architecture extracts fetures by using convolutional neural networks on each input (1-D time-series) separately, and feeds them to a MLP NN for classification as shown in Figure 10.

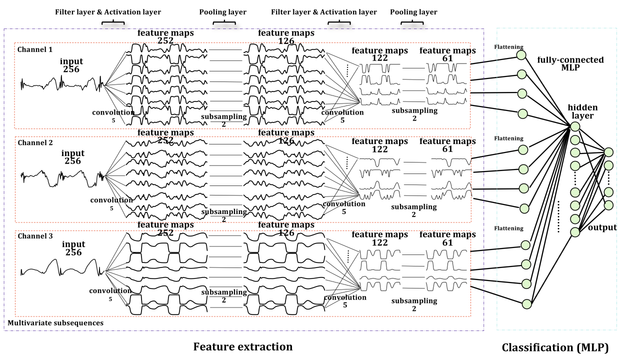

Figure 10: Multi-channel convolutional neural network. Image taken from [41].

In this way CNN's are used on temporaly coherent signals instead of naively using the whole regressor vector for single-channel input. CNN receptive fields are not feed with temporally incoherent inputs, and hence can perform meaningful feature extraction.
Energy load forecasting in [13] was performed using a deep feed-forward neural network, large datasets and man-made features. Best results were obtained by using 3 layers, whilst SAE pretraining gave no significant performance gain.

Another successful application of CNN's is for speech recognition. Audio signals are transformed to time-frequency domain and feed into multi-layered CNN as shown in Figure 11. Convolution-pooling pairs are stacked up to obtain higher level features, on top of which the standard fully connected layer is added, to combine the features of different bands.

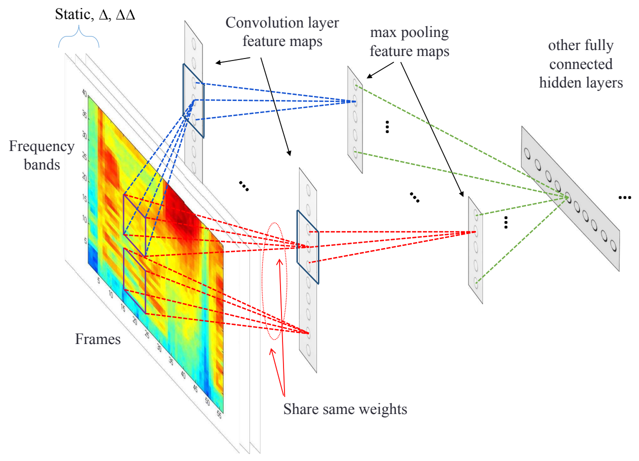

Figure 11: CNN structure used for speech recognition. Image taken from [1].

Time series data is often unlabelled and inconsistent. If a single element of the input and output vector is corrupted usually the entire vector is marked as unuseful and excluded form learning data. If we use a regression vectors the flawed sample propagates in d samples, where d denotes number of historic samples in a regressor vector. As the number of inputs grows so does the probability of a flawed sample. The input data also suffers from outliers, high redundancy, noise and other undesirable effects.
DAE provide a method for using such unlabelled, corrupt and missing data to fix it and/or use it for neural network preinitialization. If a certain proportion of data is known to be uncorrupted it can be corrupted in a way that simulates undesirable effects in data. Such data can then be used for unsupervised training e.g. to train a DAE for data reconstruction or for intermediate representation.

6. Ultra-short term wind prediction

MLP NN structure was chosen for this work. To denote dimension of some vector $\mathbf{v}$, $n(\cdot)$ notation is used: $n(\mathbf{v})$, or $n(l)$ when denoting the dimension of a layer, where $0 \le l \le L$. Mathematical description of MLP NN with L layers can be formulated as follows:

$$\mathbf{y}_0 = \mathbf{x}. \tag{16}$$
$$\mathbf{x}_l = \left[\mathbf{y}_{l-1}^T, 1\right]^T, \quad 1 \le l \le L, \tag{17}$$
$$\mathbf{v}_l = \mathbf{W}_l \cdot \mathbf{x}_l, \quad 1 \le l \le L, \tag{18}$$
$$\mathbf{y}_l = \psi_l(\mathbf{v}_l), \quad 1 \le l \le L, \tag{19}$$

where:
$\mathbf{x} = \left[x_1, x_2, ..., x_{n(x)}\right]^T \in \mathbb{R}^{n(x)\times 1}$ - input vector;
$\mathbf{y}_0 = \left[y_{0,1}, y_{0,2}, ..., y_{0,n(0)}\right]^T \in \mathbb{R}^{n(0)\times 1}$ - layer 0 output vector;
$\mathbf{x}_l = \left[x_{l,1}, x_{l,2}, ..., x_{l,n(l-1)}, x_{l,n(l-1)+1}\right]^T \in \mathbb{R}^{n(l-1)+1\times 1}$ - input vector of layer l;
$\mathbf{v}_l = \left[v_{l,1}, v_{l,2}, ..., v_{l,n(l)}\right]^T \in \mathbb{R}^{n(l)\times 1}$ - synaptic operation output vector of layer l;
$\mathbf{y}_l = \left[y_{l,1}, y_{l,2}, ..., y_{l,n(l)}\right]^T \in \mathbb{R}^{n(l)\times 1}$ - output vector of layer l;

$\mathbf{W}_l = \left[\mathbf{W}_l' \; \mathbf{b}_l\right] \in \mathbb{R}^{n(l)\times n(l-1)+1}$ - synaptic weight coefficients matrix of layer l;
$\boldsymbol{\psi}_l(\mathbf{v}_l) = \left[\psi_{l,1}(v_{l,1}), \psi_{l,2}(v_{l,2}), ..., \psi_{l,n(l)}(v_{l,n(l)})\right]^T \in \mathbb{R}^{n(l)\times 1}$ - element-wise activation functions vector of layer l ($\psi_l = \psi_{l,1} = \psi_{l,2} = \cdots = \psi_{l,n(l)}$).

Here, for wind prediction $L = 4$ layers are used with only hyperbolic tangent functions $\psi_l(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ as activation functions. All hidden layers have the same dimension, i.e. $n(1) = n(2) = n(3) = 250$ neurons. DNN were initialized using normalized initialization [19], which considerably contributes to finding good solutions in DNN, especially when used with $\tanh(x)$ activation functions. A Scaled Conjugate Gradient algorithm [32] for fast supervised learning is used to minimize Mean Squared Error (MSE) learning criteria with regularization. To avoid overfitting early stopping based on a validation dataset was used.
The used data consists of physical quantities measured with sampling time of $T_s = 10$ min. Original measurements of outputs Y represent wind speed and direction. Because of seemingly abrupt changes in wind direction measurements $\theta(i)$ when crossing 360° or 0° we use orthogonal wind components $v_{North}(i) = ||v(i)|| \cdot \cos\left(\frac{\pi\theta(i)}{180}\right)$ and $v_{East}(i) = ||v(i)|| \cdot \sin\left(\frac{\pi\theta(i)}{180}\right)$. By doing so smooth functions are used for mapping direction and wind speed change.
Three years (2010-2012) of data were obtained with wind measured on 10 m relative hight from the ground in Split, Croatia, and used for neural network training (75%), validation (15%) and testing (15%). Some of the measured data is unnecessary (redundant, not useful, unknown,...) and some needs to be artificiality derived (e.g. turbulence, gustiness) to increase the chance of successful neural network training. Input data consists of 1512 man-made and measured features which are described in [3], with additional numeric weather prediction products of ALADIN model [39] for the nearest point. Inputs represent past values of wind speed, direction, temperature, atmospheric pressure etc. All data was normalized to $[-0.9, 0.9]$.
With the use of PMI and distance based IVS $n(0) = 146$ neural network inputs were obtained. This is a drastic reduction in the number of inputs compared to naive use of 1512 available inputs. It also helps reduce the number of NN parameters. Distance based IVS was limited to the same number of inputs. Neural network has $n(4) = 36$ outputs which represent 3 hour horizon predictions with 10 min resolution, for North and East direction.

A. Neural network results

DNN described in Section 6 was trained with inputs obtained by distance based IVS and PMI. In addition, a shallow neural network was trained on the same inputs, with 936 neurons in the hidden layer to match the overall number of parameters in DNN. Results are presented in Figure 12. Although the results favour deep NN over shallow ones they lack statistical evidence obtained by training many NN. This is mainly because DNN training lasts around 10 hours. Best results obtained with DNN and distance based IVS are compared to persistent estimation on Figure 13. Results clearly show the advantages of using NN instead of persistent forecast, even on near horizon, where persistent estimation is hard to beat.

7. Conclusion

Many recent papers on deep learning architectures and training techniques provide evidence of advantages over classic shallow archi-

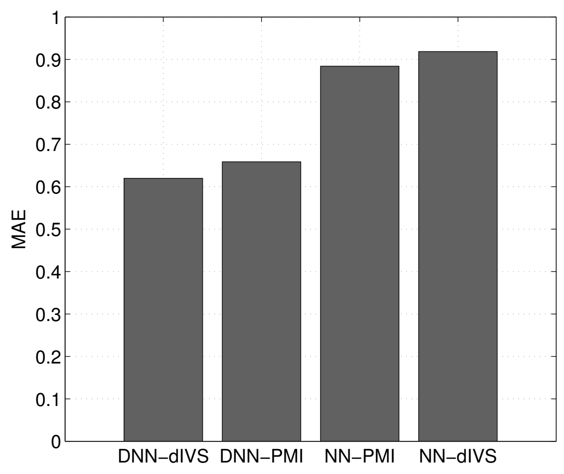

Figure 12: MAE of deep and shallow NN with IVS.

tectures. This is partially because they provide the solutions to some of the training problems mentioned in this paper. This work gives a short analysis of DNN and their key properties for time series prediction. Existing PMI and novel distance based IVS algorithm is also presented and used in ultra-short-term wind prediction for reducing input size and overall number of parameters. In this paper additional evidence to support deep NN over shallow ones is provided for ultra-short-term wind forecast. DNN results obtained in this work rely on the simple normalized initialization and represent a baseline for future work on more advanced architectures.

References

[1] O Abdel-Hamid, Li Deng, and Dong Yu. Exploring convolutional neural network structures and optimization techniques for speech recognition. *INTERSPEECH*, (August):3366–3370, 2013.

[2] O Abdel-Hamid and A Mohamed. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. *Acoustics, Speech, and Signal Processing*, 2012.

[3] Mladen Đalto, Mario Vašak, Mato Baotić, Jadranko Matuško, and Kristian Horvath. Neural-network-based ultra-short-term wind forecasting. *bib.irb.hr*, (0):8, 2014.

[4] Ludovic Arnold, Sébastien Rebecchi, Sylvain Chevallier, and H Paugam-Moisy. An Introduction to Deep Learning. *ESANN*, 2011.

[5] Yoshua Bengio. Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009.

[6] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *Neural Networks: Tricks of the Trade*, 2012.

[7] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *Pattern Analysis and Applications*, (1993):1–30, 2013.

[8] Yoshua Bengio, Olivier Delalleau, and NL Roux. The curse of highly variable functions for local kernel machines. *Advances in neural …*, 2005.

[9] Yoshua Bengio and Pascal Lamblin. Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems*, (1), 2006.

[10] Yoshua Bengio and N.L. Roux. Convex neural networks. *Advances in Neural Information Processing Systems*, 2005.

[11] J. S. Bergstra and R. Bardenet. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, pages 1-9, 2011.

[12] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 294:291–294, 1988.

[13] Enzo Busseti, Ian Osband, and Scott Wong. Deep learning for time series modeling. 2012.

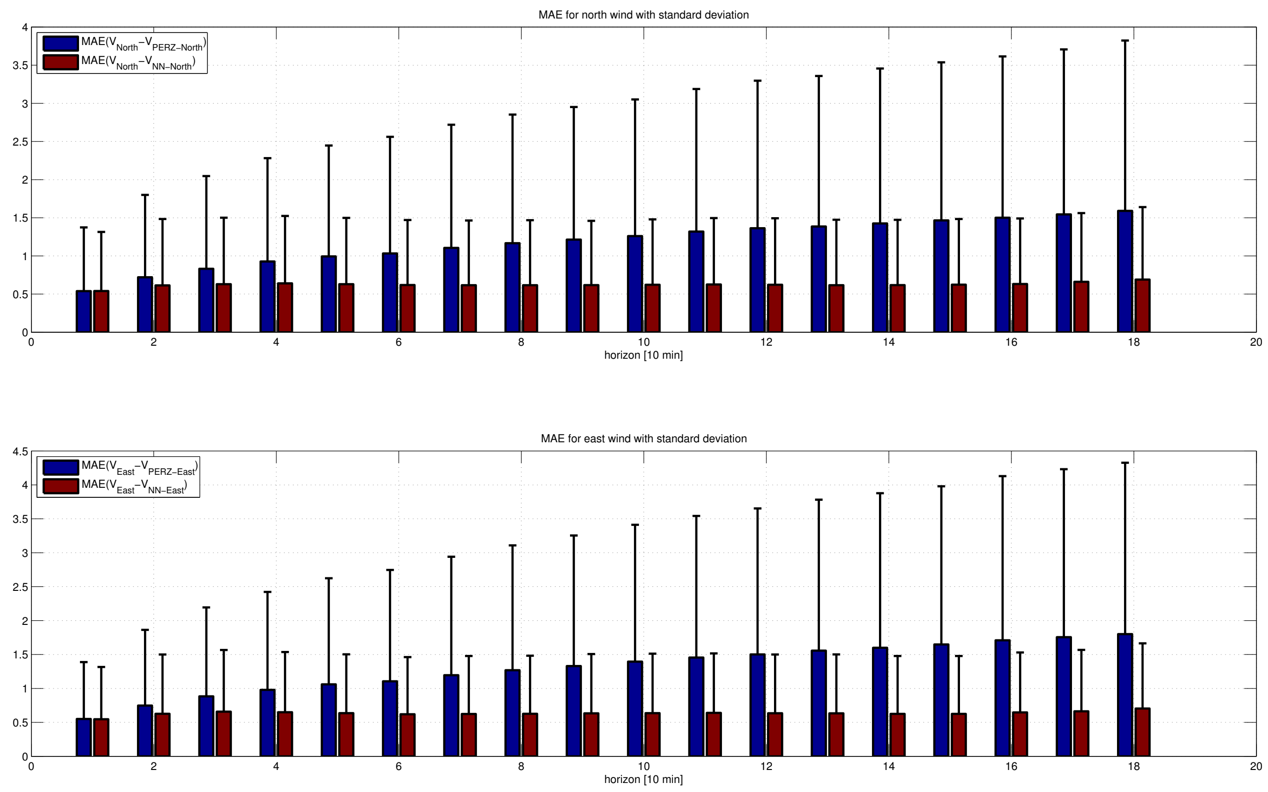

Figure 13: DNN MAE with standard deviation on 3 hour horizon for Split location.

[14] M. Clark. A COMPARISON OF CORRELATION MEASURES. *nd.edu.*

[15] R. Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. *Proceedings of the 25th international conference on Machine learning*, 2008.

[16] Thomas M. Cover and Joy A. Thomas. Elements of information theory. January 1991.

[17] Team Theano Development. DeepLearning 0.1 documentation, 2008.

[18] S. Frenzel and B. Pompe. Partial mutual information for coupling analysis of multivariate time series. *Physical review letters*, 99(20):204101, November 2007.

[19] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, 9:249–256, 2010.

[20] Xavier Glorot, Antoine Bordes, and Y Bengio. Deep sparse rectifier networks. *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, 15:315–323, 2011.

[21] R. Heller, Y. Heller, and M. Gorfine. A consistent multivariate test of association based on ranks of distances. *Biometrika*, 100(2):503–510, December 2012.

[22] G Hinton, Simon Osindero, and YW Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 2006.

[23] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, pages 1–10, 1998.

[24] DH Hubel and TN Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, pages 215–243, 1968.

[25] Alex Krizhevsky, I Sutskever, and GE Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, pages 1–9, 2012.

[26] Hugo Larochelle and Yoshua Bengio. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 1:1–40, 2009.

[27] Y LeCun and L Bottou. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[28] James Martens. Deep learning via Hessian-free optimization. *The 27th International Conference on Machine Learning (ICML 2010)*, 2010.

[29] HP Martinez. Learning deep physiological models of affect. *Computational Intelligence Magazine*, (April):20–33, 2013.

[30] Robert May, Graeme Dandy, and Holger Maier. Review of input variable selection methods for artificial neural networks. *Artificial Neural Networks - Methodological Advances and Biomedical Applications*, 2011.

[31] Robert J. May, Holger R. Maier, Graeme C. Dandy, and T.M.K. Gayani Fernando. Non-linear variable selection for artificial neural networks using partial mutual information. *Environmental Modelling & Software*, 23(10-11):1312–1326, October 2008.

[32] MF Mø ller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6:525–533, 1993.

[33] Andrew Ng. Cs294a lecture notes: Sparse autoencoder. pages 1–19, 2011.

[34] Razvan Pascanu, Tomas Mikolov, and Y Bengio. On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*, (2), 2012.

[35] Salah Rifai and P Vincent. Contractive auto-encoders: Explicit invariance during feature extraction. *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, (1), 2011.

[36] DE Rumelhart, GE Hinton, and RJ Williams. Learning internal representations by error propagation. 1985.

[37] Noah Simon and Robert Tibshirani. Comment on "Detecting Novel Associations In Large Data Sets" by Reshef Et Al, Science Dec 16, 2011. page 3, January 2014.

[38] GJ Székely and ML Rizzo. Brownian distance covariance. *The annals of applied statistics*, 3(4):1266–1269, January 2009.

[39] Martina Tudor and S Ivatek-Šahdan. Forecasting weather in Croatia

using ALADIN numerical weather prediction model. *Climate Change and Regional/Local Responses*, 2013.

[40] Pascal Vincent and Hugo Larochelle. Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th international conference on Machine learning*, 2008.

[41] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and JL Zhao. Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks. *Web-Age Information Management*, pages 298–310, 2014.