

# Artificial Neural Networks in Feature Extraction: A Review

Joaquim Leitão\*

\*jpleitao@dei.uc.pt

CISUC, Department of Informatics Engineering, Univesity of Coimbra, Portugal

**Abstract**—Data representation plays a crucial role in the overall performance of machine learning algorithms, since successfully implementing learning algorithms that work in some (typically high-dimensional) input spaces is far from being an easy task. Therefore, machine learning algorithms must learn and apply proper techniques to reduce the dimensionality of their input spaces. Dimensionality reduction is composed by two major classes of methods: On the one side there are Feature Extraction methods, which propose a new set of features by performing transformations on a given feature set. Feature Extraction algorithms can further be classified as either linear or nonlinear, depending on the nature of the performed transformations. On the other side, Feature Selection methods (ideally) select the best subset of features a provided feature set.

The current document intends to survey the literature on the application of artificial neural networks in feature extraction tasks, providing a summarised overview of this research topic.

**Index Terms**—Feature Extraction, Artificial Neural Networks.

## 1 INTRODUCTION

**M**ACHINE Learning (ML) can shortly be defined as a sub-field of computer science which allows a computer to learn a given task without being explicitly programmed to do such task. Since computers are not directly told how to solve a given task, ML approaches aim to learn the task to be solved by collecting and analysing high volumes of data on this task. Most popular ML tasks involve performing some sort of classification or categorisation of collected data, or making predictions based on historical collected information.

In such tasks choosing the right representation for the collected information is of extreme importance and highly conditions the computer's ability to learn the desired task. Indeed, *real-life* data tends to be very high-dimensional, where most learning algorithms tend to perform poorly.

As a result, proper dimensionality reduction techniques must be adopted in order to reduce the dimensionality of the input spaces. Dimensionality reduction is composed by two major classes of methods: *Feature Extraction* (FE) and *Feature Selection* (FS). While FE methods propose a new set of features by performing transformations on a given feature set, FS methods (ideally) select the best subset of features a provided feature set.

The current document intends to cover FE techniques proposed in the literature, providing a literature review on this topic. Furthermore, only the application of Artificial Neural Networks in FE is intended to be covered. Therefore, additional referenced techniques applied for this purpose (such as Principal Components Analysis) will not be covered.

The remainder of this document is organised as follows: Section 2 introduces the need to perform FE in Intelligent Systems. Section 3 presents the most popular and referenced techniques exploring the use of Artificial Neural Networks for FE purposes. Finally, section 4 concludes the document.

## 2 BACKGROUND

The current section addresses the importance of Feature Extraction in Intelligent Systems. In the literature this concept is often presented in the sequel of the well-known curse of dimensionality phenomenon, as a collection of methods that allow a reduction in the problem's dimensionality.

### 2.1 Curse of Dimensionality

The curse of dimensionality refers to how certain learning algorithms may perform poorly when confronted with high-dimensional data. In many Pattern Recognition (PR) problems, low-dimension feature spaces may not allow for a good separation of the data, leading to low-accuracy classifications. By considering more features, a feature space where an hyperplane that perfectly separates the (training) data can be determined with a resulting increase in classification accuracy.

Nonetheless it has been demonstrated that, as dimensionality increases, the amount of training data needed to accurately generalise the produced classifier grows exponentially. In addition, a small *training samples-to-features* ratio may also degrade performance [1].

A practical implication of the curse of dimensionality is that a system designer should select a small number of features when confronted with a limited training dataset. A general accepted practice is to keep the number of training samples at least ten times higher than the dimensionality:  $n/d > 10$  [1].

### 2.2 Dimensionality Reduction

In light of the discussion conducted in 2.1, a common interest in the vast majority of MPR applications is to keep the number of features as small as possible.

Feature Extraction and Feature Selection techniques are the most cited and recognised methods for achieving dimensionality reduction in PR problems. Feature Extraction algorithms propose a new set of features by performing transformations on a given feature set. On their turn, Feature Selection algorithms (ideally) select the best subset of features a provided feature set. As pointed out by Jain *et al.* [1], feature extraction is typically applied before feature selection<sup>1</sup>.

The main issues in dimensionality reduction, namely in feature extraction and selection, are related with the choice of criterion function and the appropriate dimension for the reduced feature space.

1. The idea is to start by defining a new feature set based on the original one, and then select the features that, hopefully, allow a complete separation of the training data.

### 3 NEURAL NETWORKS IN FEATURE EXTRACTION

Artificial Neural Networks (ANNs) have been extensively applied in the field of machine learning, in both classification and regression tasks. Because of their self-learning and trained characteristics, ANNs have also been successfully applied in FE tasks.

Considering the simplest form of ANNs, feedforward networks, FE can be obtained in the output of each hidden layer: the application of the neurons' activation function to the layer's input produces an output that can be interpreted as a set of new features [1]. Depending on the neurons' activation function this new set of features can be a linear or nonlinear combinations of the features in the input feature set (usually is nonlinear).

In the remainder of the current section the most popular and referenced ANN architectures applied in FE tasks are intended to be covered. Such architectures include *Feedforward* (namely *Stacked Autoencoders*), *Convolutional*, *Recurrent*, *Radial Basis*, *Restricted Boltzmann Machines* and *Self-Organising Maps* [1]–[6].

#### 3.1 Feedforward

A feedforward ANN is one of the simplest examples of a neural network. The network is composed of several layers of neurons, where neurons in each layer are connected to neurons in the next layer: that is, connections between the units do not form a cycle. In the beginning of the current section feedforward networks were used as a motivation for the application of ANNs in FE tasks.

In its simplest forms, FE can be achieved through single or multi-layer feedforward ANNs by training the network to reconstruct the input signal in a conventional way, using a backpropagation algorithm (or other popular alternatives). This form of ANN training is considered unsupervised, as only unlabelled data is being used.

In an opposite view, a multi-layer feedforward network may be trained to perform a classification task. As multiple layers are being considered, the output of the last hidden layer (which will provide the input to the output layer) can be considered as the extracted features from the provided input sample. Networks trained with this approach have been proven to suffer from well-known *vanishing gradient* problems [7].

In both scenarios the FE process is learned automatically in the early layers of the final network. In recent years, namely with advances in the field of deep learning, approaches of the first type tend to be more popular and widely used.

In this sense, training of feedforward networks for FE purposes usually comprises two steps: *Pre-training* and *Fine tuning*.

During pre-training a series of hidden layers are trained independently in an unsupervised way: Groups of 3-layer feed-forward nets (one for each hidden layer of the final network) are trained to reconstruct their respective inputs at the output layer. The idea is to make each network learn an encoder and a decoder<sup>2</sup>. In this process, the first network is trained with the input data; the second network is trained with the output of the first network's encoder, and so on. This process is illustrated in figure 1.

When pre-training is complete, the encoders are stacked together. Due to this last step, ANNs developed in this way are often referred to as *Stacked autoencoders*.

After the described pre-training, a fine tuning step is commonly carried out by considering an additional layer (usually

2. The encoder is learned from the input to the hidden layer, while the decoder is learned from the hidden to the output layer.

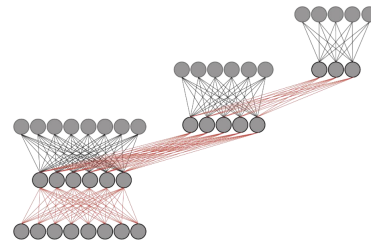


Figure 1. Stacking autoencoders. Taken from [8].

with a softmax activation function) and training the network with backpropagation (or other algorithm), to lightly adjust the weights of the pre-trained layers.

At this moment it is important to address one point in the discussion: based on what has been presented, autoencoders may simply learn the identity function for both the encoder and the decoder, which is not desired. To prevent this, certain activation functions can be forced, producing *regularized autoencoders* such as sparse autoencoders [9], denoising autoencoders [10] and contractive autoencoders [11].

Feedforward networks, namely *Stacked autoencoders*, can be applied for Feature Extraction purposes in a series of applications, including image processing and classification and speech recognition [9], [12], just to name a few.

#### 3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are ANNs designed to process data that comes in the form of multiple arrays, such as an image or a sequence of 1D signals. CNNs are based on four key ideas that take advantage of the properties of natural signals: local connections, shared weights, pooling and multi-layer usage [13]. Similarly to the feedforward case, CNNs can be used to automatically extract features and perform classification tasks.

The architecture of a CNN comprises several stages: In the first few two types of layers are alternated: *convolutional* and *pooling* - In the literature it is common to refer to a pair of convolutional and pooling as one hidden layer of the CNN, or a *slice*. Following these sequences of alternating convolutional and pooling layers, a fully connected layer is considered in order to produce the network's output. Figure 2 presents a conceptual example of a CNN.

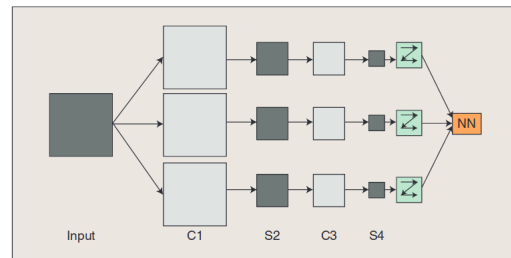


Figure 2. Conceptual example of a Convolutional Neural Network. Taken from [14].

In short, units in convolutional layers are organised in feature maps, where for each map a subset of units from one hidden layer serves as input to the next hidden layer. Like in other ANN architectures, each input sample has an associated weight. The same weight vector must be applied to the neurons in a single depth slice.

The logic behind the convolutional layer is to perform, for each map in a given hidden layer, a convolution of the neurons' weights with the input volume. The result of this local weighted sum is then passed through a non-linear activation function, such as a  $ReLU^3$ .

Because of the mentioned weight sharing principle, it is common to refer to the sets of weights as a *filter* or *kernel* that is applied to the input. For this reason, the convolutional layer is often described as consisting in the application of several filters to the input (resulting from the different feature maps considered). It can also be said that the role of this layer is to detect local conjunctions of features [13].

In a different view, units in the pooling layers enforce a form of non-linear down-sampling, by seeking to merge semantically similar features into one [13]. Several non-linear functions have been proposed to this purpose, a typical pooling function is the *max pooling*, which computes the maximum of a local set of units in one (or few) feature map.

From a feature extraction and dimensionality reduction point of view, the described sequence of convolutional and pooling layers is responsible for reducing the dimensionality of the provided input set. In some scenarios a normalisation layer may be considered; however, as pointed out in [15], this practice has fallen out of favor because its contribution has been shown to be minimal.

In the vast majority of the scenarios, CNNs are applied in classification problems, and for that reason additional fully-connected layers are considered where an high-level reasoning is performed. Regarding the network's training, the application of a backpropagation gradient through a CNN is similar to regular deep networks, allowing all the weights in all the layers and filters to be trained.

Convolutional Neural Networks have been extensively applied for Feature Extraction purposes, with the most sounding applications in fields such as image [16], [17], video [18] and speech processing and recognition [19], [20], natural language processing [21] and analysis of other sequences of 1D signals. CNNs were also applied to develop an artificial intelligence for the board game Go [22], [23]

### 3.3 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are ANNs designed to process sequential information. Unlike in Feedforward networks, where each layer can only be connected to the next one, forming a directed acyclic graph, connections in a RNN can form directed cycles. That is, in a RNN one neuron layer can be connected to a previous layer, or itself. Figure 3 presents a RNN with one hidden layer.

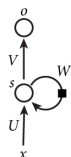


Figure 3. Illustration of a Recurrent Neural Network. Taken from [24].

Two distinct types of recurrence can be considered in RNN: If the outputs of a given layer are connected to the inputs of the same layer, then *internal recurrence* is being considered; however, if the output layer is connected to the input layer, *external recurrence*

is being considered. Current RNN architectures tend to consider *internal recurrence* rather than *external recurrence*.

RNNs can be seen as structures with *memory*, allowing them to capture information about what previous computations. This is the main reason why RNNs are so popular and effective when dealing with sequential data, such as time series. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps.

Similarly to Feedforward networks, several activation functions can be considered in the neurons of a RNN, of which sigmoid and softmax activation functions are examples.

Studying the literature in the topic of RNNs several architectures for RNNs can be found. The most popular architectures proposed in the literature include *Elman* [25], [26] and *Jordan* [27] networks. Because of space restrictions, a detailed description of the different structures will not be presented. The reader is directed to the cited works for a more detailed coverage.

With respect to the training of this networks, *Backpropagation Through Time* (BPTT) is among the most referenced and known algorithms applied to RNN. BPTT is defined as an adaptation of the popular backpropagation<sup>4</sup> training method applied in feed-forward neural networks. This method proposes an "*unfolding*" of the recurrences in the network, as presented in figure 4.

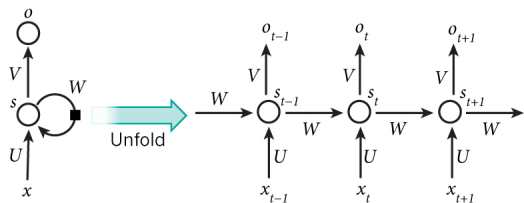


Figure 4. Illustration of a Recurrent Neural Network. Taken from [24].

Applying this technique, a feedforward neural network is obtained, and the backpropagation algorithm can be applied. During the training process the BPTT algorithm progressively unfolds the network, leading to a continuous increase in the size of the network. In theory, BPTT does not define a limit for the unfolding, although approaches such as *Truncated BPTT* have been proposed to achieve this by defining a fixed number of layers in the network.

Other alternatives to these methods can also be identified in the literature, of which *Kalman Filters* [28], *Real-Time Recurrent Learning* (RTRL) [29] and *Long Short-Term Memory* (LSTM) [30].

The main feature of the RTRL algorithm has to do with its mathematical formulation: the derivatives are explicitly computed, and therefore, no unfolding is performed. By its turn, LSTM avoids the well-known vanishing gradient problems and introduces a distinct architecture, where units are implemented in blocks containing gates used to control the flow of information into or out of their memory. Again, due to size restrictions on the current paper, the reader is deferred to the cited literature for a more in-depth coverage of these techniques.

Recurrent Neural Networks have been extensively applied for Feature Extraction purposes, with the most sounding applications in fields such as image classification [31]; time series prediction, namely in EEG signal prediction [32], speech recognition [33], among others. The work of Lee and Tashev [33], focusing on speech recognition, applies RNNs in automatic high-level feature representation, which can be considered as

4. A detailed explanation on how the backpropagation method works can be found in [7].

3. Rectified Linear Unit.

a FE task. Indeed, a transformation of raw input data to a representation that can be effectively exploited in machine learning tasks is carried out by a RNN in this work.

### 3.4 Restricted Boltzmann Machines

*Restricted Boltzmann Machines* (RBM) [34] can be defined as a variant of Boltzmann Machines<sup>5</sup> [35] with the restriction that the neurons must form a bipartite graph.

RBM comprehend two types of neurons, also called units: *visible* and *hidden* units. Neurons in the visible unit are connected to the hidden unit through a set of weights in a fully-connected way. The hidden units are used to model the data distribution:

Briefly, the following idea is adopted: Values in the visible unit are known (they can be the original input data) while values in the hidden units are not known. Therefore, the task for a RBM is to estimate these values, and this is accomplished by learning a probabilistic model to describe these values. In the literature covering this topic, this is the same as stating that the values  $P(h_j|v)$  are going to be learned, where  $h_j$  is related with one neuron in the hidden unit, while  $v$  is related with the values in the visible unit (which are known).

The values of  $P(h_j|v)$  are learned for a given value of  $W$ , which includes the weights of the connections between the two units and the bias of the neurons in the RBM. The training of a RBM consists in updating the weights and the bias of the neurons, as referenced in [36].

Regarding Feature Extraction, the connections from the visible to the hidden units can be seen as an encoder, and the connection in the opposite direction as a decoder [37]. This is illustrated in figure 5. Therefore, if one considers the expected value of the hidden units, this can be viewed as a set of features extracted from the input data [36].

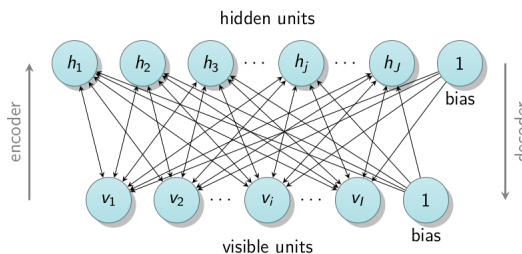


Figure 5. Restricted Boltzmann Machine. Taken from [37].

Resembling the common practices with feedforward networks, namely with autoencoders, RBM can also be stacked together forming *Belief Networks*, or *Deep Belief Networks* (DBN) [38] if a deep architecture is considered.

RBM and DBN have been applied in several problems of distinct natures such as stock price prediction [36] and predictions from EEG data [39], just to name a few.

### 3.5 Self-Organising Maps

In Self-Organising Maps (SOM), another type of ANN used for Non-Linear Feature Extraction [1], [40], a low-dimension discretized representation of the input space is produced. Such representation is called a map.

A SOM consists of neurons with associated weight vectors (with the same dimension as the input). The neurons are positioned in the map space, usually in a two-dimensional regular

spacing in either a hexagonal or rectangular grid (instead of being organised in layers, like in feedforward ANNs).

During the training of a SOM, inputs are presented to the network and, for each, the weight vector closest to the input vector is identified. Then, the weights of all the neurons in the neighbourhood of the winner neuron (that is, the one whose weight vector was selected) are updated, so that they move towards the input vector. In this way, weight vectors of neighbouring neurons in the grid are likely to represent inputs which are close in the original feature space - a "topology-preserving" map is, thus, formed [41], [42]. An example of an application where SOM was used for FE can be found in [16], although many more are available in the literature.

### 3.6 Radial Basis Function

Radial Basis Function (RBF) neural networks are a type of ANNs characterised by their use of radial basis functions as activation functions in their neurons. Although these networks have been mainly used in classification problems, one can still find applications in the literature where RBF networks were used to perform feature extraction tasks, such as in [43].

Similarly to Self-Organising Maps, Radial Basis Function neural networks explore neighborhood relationships in the data. When applied to FE tasks, RBF networks are used to predict the coordinates of the data points in the transformed feature space. Such predictions are computed during the training of these structures, by adjusting their weights in order to minimise a suitable error measure.

In short, a RBF network is composed of three neuron layers: input, hidden and output. It is common to adopt non-linear activation functions for the neurons in the network's hidden layer, such as a Gaussian function. On the other hand, neurons in the output layer typically implement a simple linear function, performing a weighted sum of their inputs.

According to Lowe and Tipping [43], the training process of a RBF network is "relatively supervised". The main reason for this claim is related with the fact that, for each input sample, the specific representation in the transformed feature space is unknown; however a relative measure of the target separation between the points in the transformed space is provided.

## 4 CONCLUSION

The current document presents a summarised overview of Feature Extraction methods exploring the application of Artificial Neural Networks.

Of the techniques included in this document, the application of stacked autoencoders and Convolutional Neural Networks has regained popularity among the *deep learning* scientific community, mostly as a result of the existent computational capabilities of today's computers and machines; Other techniques, such as Self-Organising Maps and Radial Basis Function networks do not seem to be so popular in this community at the present.

Studying the available literature, additional methods can be found which are not covered in this document. Based on the researched literature the most cited and relevant methods were chosen to be covered in this document, as space and time limitations do not allow for a more extensive coverage of this area of research. Such analysis can be the subject of future work.

## REFERENCES

- [1] A. K. Jain, R. P. W. Duin, and J. Mao, "Statistical pattern recognition: A review," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 1, pp. 4-37, 2000.

<sup>5</sup> Which, by its turn, is a type of stochastic Recurrent Neural Network and Markov Random Field.

- [2] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *International Conference on Artificial Neural Networks*, pp. 52–59, Springer, 2011.
- [3] G. Kvascev, M. G. Kvascev, and Z. Djurovic, "Radial basis function network based feature extraction for improvement of the procedure of sourcing neolithic ceramics," in *Neural Network Applications in Electrical Engineering (NEUREL), 2012 11th Symposium on*, pp. 95–100, IEEE, 2012.
- [4] O. Fabius and J. R. van Amersfoort, "Variational recurrent auto-encoders," *arXiv preprint arXiv:1412.6581*, 2014.
- [5] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.
- [6] E. Marchi, F. Vesperini, S. Squartini, and B. Schuller, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *Computational Intelligence and Neuroscience*, vol. 2017, 2017.
- [7] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.
- [8] R. Barata, "Stacked autoencoders." Lecture. Advanced Topics in Cognitive Models. University of Coimbra, 2017.
- [9] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.
- [10] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 265–272, 2011.
- [11] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 833–840, 2011.
- [12] X. Lu, Y. Tsao, S. Matsuda, and C. Hori, "Speech enhancement based on deep denoising autoencoder," in *Interspeech*, pp. 436–440, 2013.
- [13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [14] I. Arel, D. C. Rose, and T. P. Karnowski, "Deep machine learning-a new frontier in artificial intelligence research [research frontier]," *IEEE Computational Intelligence Magazine*, vol. 5, no. 4, pp. 13–18, 2010.
- [15] "Cs231n convolutional neural networks for visual recognition." <http://cs231n.github.io/convolutional-networks/>. Accessed: 2017-04-10.
- [16] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997.
- [17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.
- [18] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.
- [19] Y. LeCun, Y. Bengio, et al., "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.
- [20] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, and G. Penn, "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.
- [21] B. Hu, Z. Lu, H. Li, and Q. Chen, "Convolutional neural network architectures for matching natural language sentences," in *Advances in neural information processing systems*, pp. 2042–2050, 2014.
- [22] C. Clark and A. Storkey, "Teaching deep convolutional neural networks to play go," *arXiv preprint arXiv:1412.3409*, 2014.
- [23] C. J. Maddison, A. Huang, I. Sutskever, and D. Silver, "Move evaluation in go using deep convolutional neural networks," *arXiv preprint arXiv:1412.6564*, 2014.
- [24] "Recurrent neural networks tutorial, part 1 – introduction to rnns." <http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/>. Accessed: 2017-04-11.
- [25] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.
- [26] J. L. Elman, "Distributed representations, simple recurrent networks, and grammatical structure," *Machine learning*, vol. 7, no. 2–3, pp. 195–225, 1991.
- [27] M. I. Jordan, "Serial order: A parallel distributed processing approach," *Advances in psychology*, vol. 121, pp. 471–495, 1997.
- [28] R. E. Kalman et al., "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.
- [29] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.
- [30] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [31] A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," in *Advances in neural information processing systems*, pp. 545–552, 2009.
- [32] S. C. Prasad and P. Prasad, "Deep recurrent neural networks for time series prediction," *arXiv preprint arXiv:1407.5949*, 2014.
- [33] J. Lee and I. Tashev, "High-level feature representation using recurrent neural network for speech emotion recognition," in *Interspeech 2015, ISCA - International Speech Communication Association*, September 2015.
- [34] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (D. E. Rumelhart and J. L. McClelland, eds.), vol. 1, ch. 6, pp. 1944–281, MIT Press, 1986.
- [35] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.
- [36] X. Cai, S. Hu, and X. Lin, "Feature extraction using restricted boltzmann machine for stock price prediction," in *Computer Science and Automation Engineering (CSAE), 2012 IEEE International Conference on*, vol. 3, pp. 80–83, IEEE, 2012.
- [37] B. Ribeiro, "Deep learning." Lecture. Advanced Topics in Cognitive Models. University of Coimbra, 2017.
- [38] G. E. Hinton, "Deep belief networks," *Scholarpedia*, vol. 4, no. 5, p. 5947, 2009.
- [39] M. Hajinoroozi, T.-P. Jung, C.-T. Lin, and Y. Huang, "Feature extraction with deep belief networks for driver's cognitive states prediction from eeg data," in *Signal and Information Processing (ChinaSIP), 2015 IEEE China Summit and International Conference on*, pp. 812–815, IEEE, 2015.
- [40] Z. M. Hira and D. F. Gillies, "A review of feature selection and feature extraction methods applied on microarray data," *Advances in bioinformatics*, vol. 2015, 2015.
- [41] T. Kohonen, "Self-organising maps," *Springer Series in Information Science*, vol. 30, Berlin, 1995.
- [42] T. Villmann, R. Der, M. Herrmann, and T. M. Martinetz, "Topology preservation in self-organizing feature maps: exact definition and measurement," *IEEE Transactions on Neural Networks*, vol. 8, no. 2, pp. 256–266, 1997.
- [43] D. Lowe and M. E. Tipping, "Neuroscale: Novel topographic feature extraction using rbf networks," *Advances in Neural Information Processing Systems*, pp. 543–549, 1997.