NOVA IMS Information Management School

MGI

Mestrado em Gestão de Informação
Master Program in Information Management

Unsupervised feature extraction with Autoencoder
for the representation of Parkinson's disease patients

Veronica Kazak

Dissertation presented as partial requirement for obtaining the Master's degree in Information Management

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

UNSUPERVISED FEATURE EXTRACTION WITH AUTOENCODER

FOR THE REPRESENTATION OF PARKINSON'S DISEASE PATIENTS

by

Veronica Kazak

Proposal for Dissertation presented as partial requirement for obtaining the Master's degree in Information Management, with a specialization in Knowledge Management and Business Intelligence

Advisor: Professor Roberto Henriques, PhD

Co-Advisor: Professor Mauro Castelli, PhD

September 2018

Abstract

Data representation is one of the fundamental concepts in machine learning. An appropriate representation is found by discovering a structure and automatic detection of patterns in data. In many domains, representation or feature learning is a critical step in improving the performance of machine learning algorithms due to the multidimensionality of data that feeds the model. Some tasks may have different perspectives and approaches depending on how data is represented. In recent years, deep artificial neural networks have provided better solutions to several pattern recognition problems and classification tasks. Deep architectures have also shown their effectiveness in capturing latent features for data representation.

In this document, autoencoders will be examined to obtain the representation of Parkinson's disease patients and compared with conventional representation learning algorithms. The results will show whether the proposed method of feature selection leads to the desired accuracy for predicting the severity of Parkinson’s disease.

Keywords:

Autoencoder, Representation Learning, Feature Extraction, Unsupervised Learning, Deep Learning.

Acknowledgements

Developing this master's thesis initially seemed extremely challenging due to requirements for conducting research in the field of deep learning but it has been made possible by the support of some people to whom I would like to express my sincere gratitude.

First and foremost, I would like to thank my two advisors Prof. Dr. Roberto Henriques and Prof. Dr. Mauro Castelli. You both provided me with a perfect balance of guidance and freedom.

Prof. Dr. Roberto Henriques became my guide in the world of data science – first, planting the seed of curiosity in the area and then, encouraging my research. Thank you for helping me to have carried out research and for all your valuable advice in developing the document.

Prof. Dr. Mauro Castelli played a key role in determining the area of research suggesting the source of data on Parkinson's disease patients. Thank you for your availability and support as well as for solving doubts in training neural networks.

I would also like to thank Prof. Dr. Pedro Cabral for teaching me the standards and guidelines for scientific research and writing. Thanks to Professor, I managed to organize the process as from the earliest stage and comply with deadlines.

I would also like to express my special thanks to Eran Lottem, the neuroscientist from Champalimaud Foundation who inspired me for studying deep learning. Being a brain scientist, he triggered my interest in artificial neural networks that simulate a real brain. Our talks about brain activity strengthened further my decision to do this research.

I also want to thank my friends and fellow colleagues for your moral support and contributing to keep my mental energy.

Last but not least, truly heartfelt and great thanks to my family and my loved ones Laurent Filipe Iarincovschi, Ivan Iarincovschi and Svetlana Kalinina for your blessings, for the amazing support and always being on my side giving the strength to continue and conclude this work.

Sincere thanks to all,

Veronica Kazak.

Contents

CHAPTER 1 INTRODUCTION 1

1.1 MOTIVATION 1

1.2 BACKGROUND 1

1.2.1 Theoretical framework 1

1.2.2 Autoencoders in medical data research 3

1.3 STUDY RELEVANCE 4

1.4 OBJECTIVES 5

CHAPTER 2 NEURAL NETWORKS: DEFINITIONS AND BASICS 6

2.1 INTRODUCTION TO ARTIFICIAL NEURAL NETWORKS 6

2.1.1 Single perceptron 6

2.1.2 Multilayer perceptron 7

2.1.3 Activation functions 9

2.1.3.1 Sigmoid activation function 9

2.1.3.2 Hyperbolic tangent activation function 10

2.1.3.3 Rectified Linear Unit (ReLU) activation function 10

2.1.3.4 Softmax activation function 11

2.2 TRAINING ARTIFICIAL NEURAL NETWORKS 12

2.2.1 Backpropagation 12

CHAPTER 3 AUTOENCODER FRAMEWORK 16

3.1 OVERVIEW 16

3.2 AUTOENCODER ALGORITHM 17

3.3 PARAMETRIZATION OF AUTOENCODER 17

3.3.1 Code size 17

3.3.1.1 Undercomplete Autoencoders 18

3.3.1.2 Overcomplete Autoencoders 18

3.3.2 Number of layers 18

3.3.2.1 Restricted Boltzmann machine 18

3.3.2.2 Deep belief network 20

3.3.2.3 Stacked autoencoder 20

3.3.3 Loss function 21

3.3.3.1 Mean squared error loss function 22

3.3.3.2 Cross entropy loss function 22

3.3.4 Optimizer 22

3.3.5 Regularization 23

3.3.5.1 Sparse autoencoder 23

3.3.5.2 Denoising autoencoder 25
3.3.5.3 Contractive Autoencoder 27
3.4 OTHER AUTOENCODERS 28
3.4.1 Variational Autoencoder 28
3.4.2 Convolutional autoencoder 29
3.4.3 Sequence-to-sequence autoencoder 31
3.5 AUTOENCODERS SUMMARIZED 32

CHAPTER 4 METHODOLOGY 34

4.1 RESEARCH METHODOLOGY MAIN STEPS 34
4.2 DATASET 35
4.3 TECHNICAL RESEARCH TOOLS 36
4.4 INITIAL DATA COMPREHENSION 36
4.5 EXTRACTING FEATURES WITH PRINCIPAL COMPONENT ANALYSIS 37
4.6 EXTRACTING FEATURES WITH AUTOENCODER 40
4.6.1 Training Vanilla autoencoder 40
4.6.2 Visualizing features obtained by training Vanilla autoencoder 41
4.6.3 Training Stacked autoencoder and feature visualization 42
4.7 EVALUATION METHOD 44
4.7.1 Linear Regression 44
4.7.2 Support vector regression 45
4.7.3 Neural Network model 46
4.7.4 Validation method 46

CHAPTER 5 RESULTS AND DISCUSSION 48

5.1 RAW DATA OUTCOME 48
5.2 PCA OUTCOME 49
5.3 AUTOENCODER OUTCOME 51
5.4 STACKED AUTOENCODER OUTCOME 53
5.5 COMPARATIVE ANALYSIS OF RESULTS FROM DIFFERENT INPUTS 55

CHAPTER 6 CONCLUSIONS AND FUTURE WORK 57

BIBLIOGRAPHY 59

APPENDIX A 69

APPENDIX B 70

APPENDIX C 71

APPENDIX D 72

APPENDIX F 74

List of Figures

Figure 1.1 Distribution of published papers that use deep learning in subareas of health informatics 4
Figure 2.1 Illustration of the mathematical model of a biological neuron – single perceptron 7
Figure 2.2 Multilayer perceptron 8
Figure 2.3 Sigmoid activation function 10
Figure 2.4 Hyperbolic tangent activation function 10
Figure 2.5 ReLU activation function 11
Figure 2.6 The output of the Softmax function for 4 classes 11
Figure 2.7 The gradient of the loss function 13
Figure 3.1 Autoencoder architecture 16
Figure 3.2 Boltzmann machine 19
Figure 3.3 Restricted Boltzmann machine 19
Figure 3.4 Training a Stacked Autoencoder 21
Figure 3.5 Sparse Autoencoder 24
Figure 3.6 Denoising autoencoder 26
Figure 3.7 Denoising effect 26
Figure 3.8 Variational autoencoder 29
Figure 3.9 Convolution and pooling in CNN 30
Figure 3.10 Convolutional autoencoder 31
Figure 3.11 The unfolded recurrent neural network 32
Figure 4.1 The core block diagram of the proposed methodology 34
Figure 4.2 Correlation matrix of Parkinson's patient dataset presented with the heatmap 37
Figure 4.3 Visualization of the cumulative proportion of total variance explained by PCA 38
Figure 4.4 Scatter plot of the three-dimensional representation produced by taking the first three principal components obtained from different sets of input attributes with the color specification of total_UPDRS scores 39
Figure 4.5 Training autoencoders of different architectures 41
Figure 4.6 Scatter plots of three-dimensional codes produced by training autoencoders based on different sets of input attributes with the color specification of total_UPDRS scores 42
Figure 4.7 Scatter plots of three-dimensional codes produced by training the 19-15-10-5-3 stacked autoencoder with the color specification of total_UPDRS scores 43
Figure 4.8 Neural network architectures used in supervised learning 46
Figure 5.1 Training process of the 20-5-NN model with different number of epochs fed with raw data 49
Figure 5.2 Training of 20-5-NN models fed with different number of principal components 51

Figure 5.3 Training of the 20-5-NN model fed with different number of features obtained by training the autoencoder 53
Figure 5.4 Training stacked autoencoders of different architectures 55

Figure Appendix D-1 Scatter plot of total_UPDRS vs motor_UPDRS 72
Figure Appendix D-2 Scatter plot of the three-dimensional representation produced by taking the first three principal components obtained from different sets of input attributes with the color specification of motor_UPDRS scores 72
Figure Appendix E-1 Visualization of training autoencoders with different hyperparameters and number of epochs 73
Figure Appendix F-1 Scatter plots of the three-dimensional codes produced by training autoencoders of different architectures and trained with different hyperparameters 74

List of Tables

Table 3.1 Variants of the gradient descent (GD) based on the amount of data 23
Table 3.2 Extensions of the gradient descent 23
Table 3.3 Literature review on Autoencoders covered in this thesis 32
Table 4.1 Variable description of Parkinson's telemonitoring dataset 35
Table 5.1 Model performance and computation time using patient data represented by original descriptors 48
Table 5.2 Model performance and computation time using patient data represented by features obtained with PCA 49
Table 5.3 Model performance and computation time using patient data represented by features obtained with autoencoders of different architectures 51
Table 5.4 Model performance and computation time using patient data represented by features obtained with stacked autoencoders 53
Table 5.5 Total UPDRS scores prediction results using patient data represented by 19 original descriptors and pre-processed by principal component analysis, vanilla and stacked autoencoders 55

Table Appendix A-1 Descriptive statistics of Parkinson's patient dataset 69
Table Appendix B-1 Correlation coefficients between variables in Parkinson's patient dataset 70
Table Appendix B-2 Top-10 the most correlated variables 70
Table Appendix C-1 Proportion of the variance explained by principal components obtained via transformation of 16 vocal attributes 71
Table Appendix C-2 Proportion of the variance explained by principal components obtained via transformation of 19 original attributes 71

Chapter 1
Introduction

1.1 Motivation

This thesis is aimed at exploring capabilities of different types of autoencoders in extracting useful features for the representation of Parkinson's disease patients and comparing its performance in supervised learning with the representation obtained using principal component analysis (PCA) technique and with the raw data.

Like PCA, autoencoders can provide a solution for learning a lower dimensional latent representation but unlike PCA, they learn non-linear transformations. Having shown better performance in training since 2006 when it was first applied to MNIST dataset (Hinton & Salakhutdinov, 2006), it seems appealing to apply this technique, especially when a problem to be solved is characterized by complex interrelationships between variables.

1.2 Background

1.2.1 Theoretical framework

An autoencoder is an artificial neural network which is trying to reconstruct the input in the output while being trained (Goodfellow, Bengio, & Courville, 2016). It belongs to the class of unsupervised learning algorithms (Baldi, 2012), so it finds patterns in a dataset by learning the internal structure and features of data.

Initially, autoencoders were primarily viewed as a tool for dimensionality reduction and feature learning (Goodfellow et al., 2016). With the emergence and development of contemporary deep learning, autoencoders are considered as an effective way in pre-training neural networks (Hinton & Osindero, 2006; Bengio et al., 2007).

The idea of autoencoders was first associated with the resurgence of interest in back-propagation algorithm (Rumelhart, Hinton, & Williams, 1986). In 1988, again as a part of the

historical background of neural networks, the term "auto-association" was used in studying multilayer perceptron (Bourlard & Kamp, 1988).

A 1989 paper contributed to further study of autoencoders by examining the auto-associative case in detail and demonstrating how, using the backpropagation algorithm, learning occurs without a priori knowledge of data structure (Baldi & Hornik, 1989). This paper also showed that when an activation function is linear, the auto-associative case is a special case of PCA (Baldi & Hornik, 1989).

In 1994, Hinton and Zemel's paper introduced a new way of training autoencoders to study non-linear representations (Hinton & Zemel, 1994). Here, the term autoencoder was used instead of auto-association, although the terminology associated with the concept has continued to evolve over time.

The next milestone dates back to 2006 when several papers on the subject were published. The paper (Hinton & Osindero, 2006) marked the beginning of contemporary deep learning era (Wang & Raj, 2017) and introduced a deep belief network with layer-wise pretraining method. The work of (Hinton & Salakhutdinov, 2006) explored how a multi-layered neural network could be pre-trained one layer at a time by fine-tuning deep autoencoder using backpropagation. Here, feature learning for dimensionality reduction purpose was conducted by both, autoencoder and PCA, resulting in outperformance of autoencoders (Hinton & Salakhutdinov, 2006). This paper was one of the inspirational sources for the thesis.

The idea of layer-wise pretraining has been exploited in several studies where further explanation of the pre-training mechanism was provided with some proposed refinements (Bengio et al., 2007; Bengio, 2009; Vincent et al., 2010) but the stem idea came from the original work on deep belief networks. The main principle is that to obtain a low-dimensional representation, each layer is trained at a time by optimizing a local unsupervised criterion to produce a useful higher-level representation based on the input from the below layer with the expectation to improve the generalized representation. As a result, several autoencoders are stacked together. This architecture is often referred as stacked autoencoder.

Using autoencoders as data compressors is considered a classical approach (Rifai et al., 2011). Modern architectures of autoencoders is mostly associated with so-called overcomplete architectures with the code size larger than the input. Such autoencoders require the

introduction of special regulators to avoid useless reconstruction (Goodfellow et al., 2016). Regularized autoencoders will be discussed in the third chapter.

1.2.2 Autoencoders in medical data research

The subsection 1.2.1 describes the important milestones that form the body knowledge about autoencoders. The mentioned research papers provide a contextual approach to the scientific understanding of the concept. The subject of interest covered by this thesis is applying the autoencoder algorithm for Parkinson's disease patient data and comparing its feature extraction capabilities with other methods.

Since then as deep machine learning techniques have shown their effectiveness in both, supervised and unsupervised tasks, they proved to be well suitable to medical big data to extract useful knowledge from it (Litjens et al., 2017). There are several interesting papers related to the use of autoencoders in the field of medicine. One of the greatest inspirations for the present work was a study conducted by a research team at Mount Sinai Hospital in New York City in 2015. Based on the electronic health records (test results, doctor visits, and so on), a patient´s representation was derived using the stacked autoencoder architecture and called "Deep Patient" (Miotto et al., 2016). Without any expert instruction, Deep Patient discovered hidden patterns in the hospital data that can predict the patient's future with respect to the development of certain diseases. The results obtained using Deep Patient representation were better than when using raw patient's data or representation obtained using PCA (Miotto et al., 2016).

A similar approach for feature extraction with subsequent use in supervised learning is described in the paper (Zafar et al., 2016). The proposed method, called SAFS, used the stacked autoencoder for a higher-level representation of the most vulnerable demographic subgroup of patients to predict risk factors for hypertension. The results showed that SAFS approach and representation learning outperform the models with unrepresented input. This paper demonstrated how deep architectures can be applied to a specific precision medicine problem.

Of particular interest is a recent paper where a stacked autoencoder and softmax classifier were used to diagnose Parkinson's disease (Caliskan et al., 2017). As in previous studies, the framework involved unsupervised learning to extract representation features that served as an input for supervised learning while a softmax classifier was built for prediction purposes.

This paper showed how the data dimension can be reduced with an autoencoder, and thereby improve the predictive power of a classifier.

1.3 Study relevance

Deep learning is an active research area. Many interesting papers are constantly appearing in journals. In healthcare, they are primarily pertinent to medical image analysis (Litjens et al., 2017). Nevertheless, the flow of medical data is much more diverse and not limited to images. Figure 1.1 shows an increasing number of publications on deep learning in healthcare sub-areas.

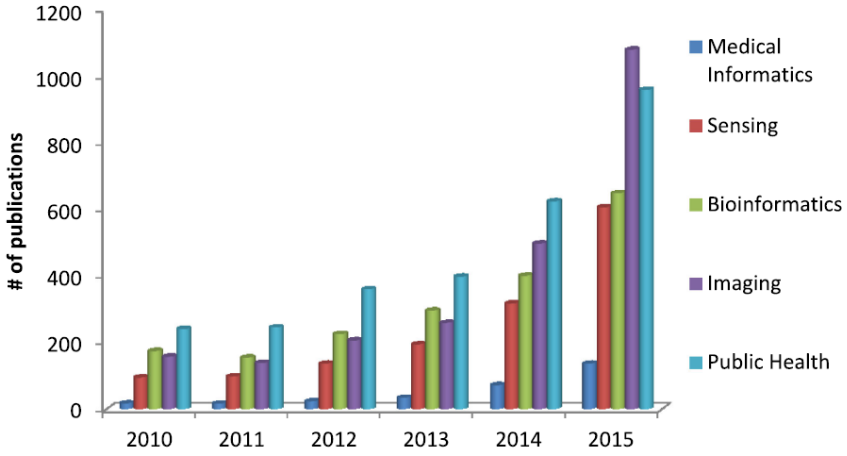


Figure 1.1 Distribution of published papers that use deep learning in subareas of health informatics
Source: (Ravi et al., 2017)

In papers that provide a review of deep learning in medicine, autoencoders are referred as one of the commonly used deep learning architectures along with other deep learning techniques (Ravi et al., 2017; Litjens et al., 2017). The main idea of their use is to capture useful properties for the representation, whether it is an image or a numeric vector of an observation.

Parkinson's disease is a degenerative neurological disorder, the second most common after Alzheimer's disease. About 5 million people suffer from Parkinson's disease worldwide (Gibrat et al., 2009). Some motor symptoms can drastically affect the quality of the patient's daily life (Bryant et al., 2015). Diagnosis of the disease in the early stages can significantly improve the plan for necessary treatment, and thus, maintain the quality of life (Challa et al., 2016).

A great interest in detecting markers for the diagnosis of the disease is related to the fact that this can be done in the preclinical stage (Postuma & Montplaisir, 2009). Parkinson's disease is

known for causing disturbances of varying degrees in the production of vocal sounds (Asgari & Shafran, 2010). Voice recording and speech tests can be an effective non-invasive tool for diagnosis (Tsanas et al., 2010).

Machine learning algorithms have been already used to extract useful information from patients' speech and predicting the severity of Parkinson's disease (Nilashi et al. , 2016; Caliskan et al., 2017). Nevertheless, the improvement of algorithms themselves, including those related to deep learning, provides the ground for continuing studies of patients' speech data to further improve the overall prognostics and develop applications for home-based assessment or telemonitoring Parkinson's disease outside of hospital settings (Asgari & Shafran, 2010).

Parkinson's patient voice data is measurable but some characteristics are indecipherable for the human ear, especially on early stages of the disease (Rouzbahani & Daliri, 2011). Identifying and measuring speech patterns in patients and comparing them with healthy people is thereby becoming one of the central tasks in the study of voice data. In this regard, applying deep techniques, in particular, autoencoders seems to be adequate and promising since these methods have already proved effective in capturing silent features in complex data (Ravi et al., 2017).

1.4 Objectives

The main goal of this study is to explore whether it is possible to improve the Parkinson's patient representation by extracting useful features from a range of biomedical voice measurements using an autoencoder. To achieve the main goal, the following specific objectives were defined:

1. Systematize the knowledge base of autoencoders to provide the background and establish a sense of structure that guides the research.
2. Apply unsupervised learning techniques to train neural networks for the Parkinson's patient representation.
3. Provide sensitivity analysis when using different types of autoencoders, PCA, and raw data.
4. Compare the performance of representations obtained by the autoencoder, PCA and raw data in the supervised learning.

Chapter 2
Neural networks: Definitions and basics

The aim of this chapter is to provide an essential background of artificial neural networks for a better intuitive understanding of concepts and further exploration of autoencoders.

2.1 Introduction to artificial neural networks

Inspired by biological mechanisms for the processing of natural signals in the human brain, artificial neural networks (ANN) were the subject of research since the mid-20th century (Deng et al., 2014). The principal unit of the artificial neural network is a node by analogy with the biological brain neuron. Connections between neurons in ANN also simulate interconnections between neurons in the brain (Gibson & Patterson, 2017). Neurons are exchanging signals between one another forming a dense and complex network.

The term "neural network" appeared in the middle of XX century. In 1943, McCulloch and Pitts introduced a simplified model of a neuron in their paper (McCulloch & Pitts, 1943) which is considered the beginning of the computational theory of brain activity (Piccinini, 2004). Later, in the 50's, Frank Rosenblatt developed a neural network model based on mathematical algorithms (Rosenblatt, 1958). Like its biological prototype, an artificial neuron can learn by adjusting parameters that describe synaptic conductivity. This model has laid the foundation for studies of neural networks as biological processes in the brain, and use of neural networks as an artificial intelligence method for solving various applied problems (Wang & Raj, 2017).

2.1.1 Single perceptron

The simplest form of a neural network was given the name of a perceptron back in Rosenblatt's work. The perceptron was originally conceived as a simplified mathematical model of neuronal processes in the brain: "The perceptron is designed to illustrate some of the fundamental properties of intelligent systems in general, without becoming too deeply enmeshed in the

special, and frequently unknown, conditions which hold for particular biological organisms" (Rosenblatt, 1958).

Figure 2.1 shows a perceptron and zooms in detail how data is processed by neurons. The perceptron is considered a feed-forward network. The term forward is used to indicate the direction of the information flow in neural networks – from the input to the output layer without cycles. The perceptron takes the input data $\boldsymbol{x_i}$ and proceeds it to the neuron (cell body). The incoming signal is multiplied by a weight value $\boldsymbol{w_i}$ that is assigned to each input $\boldsymbol{x_i}$. The neuron processes weighted inputs by summing up to one value. At this stage, a bias or offset term, referred to as $\boldsymbol{b}$ or $\boldsymbol{w0}$ in machine learning models, is added. The bias has a value of 1 and ensures that different functions are computable by shifting them to get a better approximation. Finally, an output signal is produced by performing a mathematical operation on the above result using the activation function $\boldsymbol{f(x)}$.

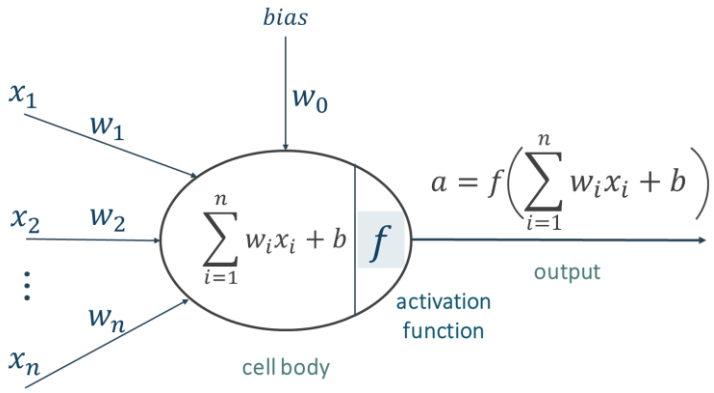


Figure 2.1 Illustration of the mathematical model of a biological neuron – single perceptron
Source: adapted from (Karpathy, 2015)

2.1.2 Multilayer perceptron

In 1969, Minsky and Papert published the book "Perceptrons" where they expressed their scepticism about the perceptron, in particular, of its incapability to learn non-linearly separable boolean XOR function (Minsky & Papert, 1969). This publication is now regarded in literature as the beginning of so-called "AI winter", the period that followed a massive wave of interest to neural networks. Minsky and Papert didn't merely show limitations of the perceptron but argued that a solution to XOR problem can be found by training networks with multiple layers, although a practical study of such networks happened a little later.

One or more neurons in the hidden layer connected only with the input and output form a single-layer neural network. Increasing the number of neurons in a single layer results in a shallow neural network architecture. By stacking one-layer neural network upon another and feeding the successor layers with the output from previous layers one can get a deeper, fully connected neural network which is also known as a multilayer perceptron (Figure 2.2).

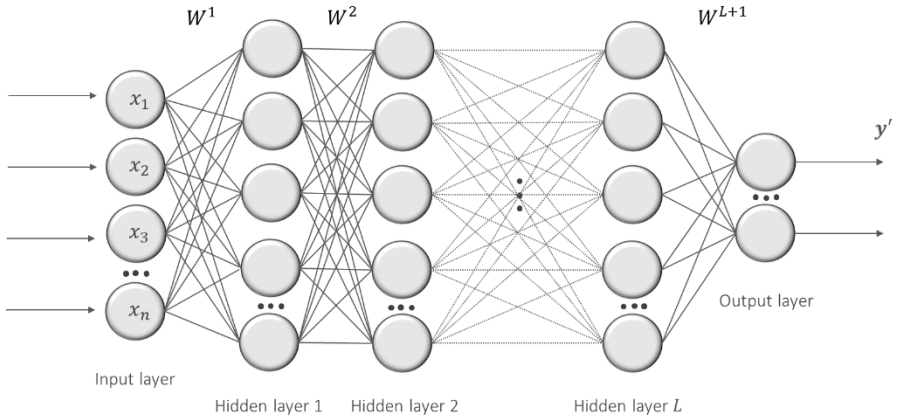


Figure 2.2 Multilayer perceptron

A multilayer perceptron uses the feed-forward mechanism in information processing. The idea of feed-forward network is to approximate some function $\boldsymbol{f(x)}$ and map the input represented by a vector $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{n}}$ to a target $\boldsymbol{y}$. The network defines the mapping $\boldsymbol{y' = f(x, w, b)}$ and learns the values of parameters $\boldsymbol{w}$ and $\boldsymbol{b}$ resulting in the best function approximation. As it was examined in subsection 2.2.1 each neuron's output will be computed by activating the weighted sum of the previous layer input. Equation (2.5) represents this mathematically:

$$a_j = f(\sum_{i=1}^{n} w_{ij}x_i + b_j), \tag{2.1}$$

where $\boldsymbol{f}$ is an activation function; $\boldsymbol{w_{ij}}$ – weight for the connection of input $\boldsymbol{x_i}$ to a neuron $\boldsymbol{j}$; $\boldsymbol{b_j}$ – bias term of the corresponding neuron.

The presence of several hidden layers requires the introduction of an additional nomenclature (for the convenience of designation in vectorized form):

$$z = Wx + b \tag{2.2}$$

$$a = f(z), \tag{2.3}$$

where $\boldsymbol{W} \in \mathbb{R}^{\boldsymbol{mxn}}$, $\boldsymbol{b} \in \mathbb{R}^{\boldsymbol{m}}$, $\boldsymbol{m}$ is a number of units in a layer, and $\boldsymbol{f}$ is applied stepwise, describing how functions are composed together becoming an input for the subsequent layer:

$$f(z) = f([z_1, z_2, \dots z_m]) = [f(z_1);\ f(z_2); \dots f(z_m)] \tag{2.4}$$

For subsequent stacked layers, a superscript will be used to denote each layer and parameterize each layer connection. So, the equation takes the following form:

$$a_k^l = f(\sum_j w_{kj}^l a_j^{l-1} + b_k^l), \tag{2.5}$$

where each weight has two indices: $\boldsymbol{j}$ indicates an emitting node and $\boldsymbol{k}$ – a receiving node.

The mathematical explanation of the feed-forward process in a multilayer perceptorn make it clear that increasing the number of layers makes sense only when using non-linear activation functions, otherwise, several layers of linear perceptrons can be collapsed into one due to the linearity of the entire circuit of computations (Ng, Katanforoosh, & Mourri, 2017b).

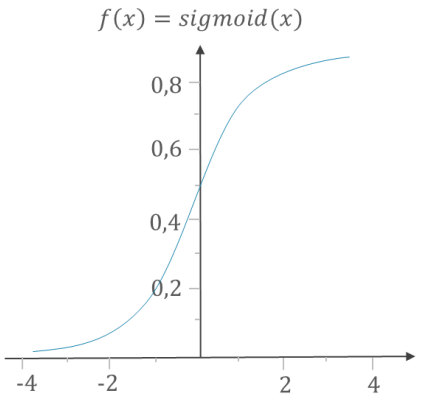
2.1.3 Activation functions

In the original perceptron, the activation function was a simple threshold operator. Linear functions have also been exploited as activation functions in neural networks (Baldi & Hornik, 1989). In modern networks, however, activation functions are most often referred to as non-linear functions. Adding nonlinearity to the network allows it to learn more complex functional mappings from data and avoid constraints associated with linear functions.

There are several commonly used activation functions in neural networks. The following subsections discuss how exactly each of the functions proceeds the linear combination of weighted inputs to produce non-linear decision boundaries in hyperplanes.

2.1.3.1 Sigmoid activation function

The sigmoid has a mathematical form as shown by equation (2.1). It maps a real number into the interval between 0 and 1 (Figure 2.3).

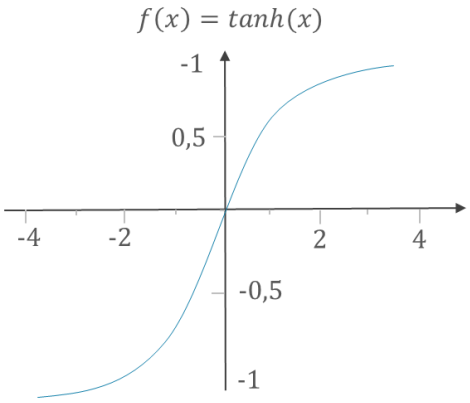


$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

Figure 2.3 Sigmoid activation function

2.1.3.2 Hyperbolic tangent activation function

The hyperbolic tangent or tanh nonlinearity is shown by equation (2.2). It squashes a real-valued number to the range between -1 and 1. Unlike the sigmoid, its output is zero-centred (Figure2.4).



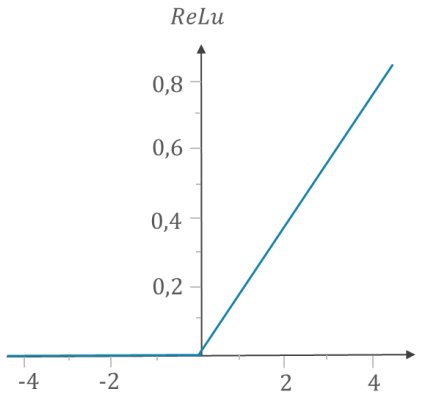
$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.7}$$

Figure 2.4 Hyperbolic tangent activation function

2.1.3.3 Rectified Linear Unit (ReLU) activation function

This function became popular after showing its effectiveness in training some types of neural networks (Nair & Hinton, 2010), as well as being faster for training larger networks (Krizhevsky, Hinton, & Sutskever, 2012). The ReLU returns zero if the value of the argument is negative, and raw output otherwise (2.3). In other words, the activation is generated at zero. This effect is shown on Figure 2.5.

The rectifying neurons have been explored in the paper (Glorot, Bengio, & Bordes, 2011). These neurons, according to the study's authors, are more biologically authentic and perform equally or better in comparison to sigmoid and hyperbolic tangent networks.



$$ReLu = max(x, 0) \tag{2.8}$$

Figure 2.5 ReLU activation function

2.1.3.4 Softmax activation function

The softmax function is usually applied at the output level of a neural network when dealing with multinomial logistic regression where the dependent variable has more than two levels (albeit it can also be used for a binary classifier). The softmax function squashes outputs of each unit to the range [0, 1] so, that the total sum of outputs is equal to 1. The result of softmax is equivalent to the categorical probability distribution (Hinton, Srivastava, & Swersky, 2012). The mathematical form of the softmax function for $\boldsymbol{k}$ classes is represented by equation (2.4).

$$softmax(x_j) = \frac{e^{xj}}{\sum_{k=0}^{k} e^{xk}}, \quad for\ j = 1 \dots K, \tag{2.9}$$

The output of the softmax function might look like a matrix of probabilities (Figure 2.6). For example, let the target have 4 labels (assuming indexing from zero) and let's take 4 training samples. The resulting prediction matrix will have the probabilities of belonging to a class. The indicator target matrix assigns a class for each sample.

Original target (label) *Probabilities predicted* *Indicator target matrix*

$$\begin{bmatrix}0\\1\\2\\3\end{bmatrix} \Rightarrow \begin{bmatrix}0.6 & 0.1 & 0.2 & 0.1\\0.1 & 0.7 & 0.1 & 0.1\\0.2 & 0.1 & 0.5 & 0.2\\0.1 & 0.1 & 0.1 & 0.7\end{bmatrix} \Rightarrow \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

Figure 2.6 The output of the Softmax function for 4 classes

2.2 Training artificial neural networks

The learning algorithm used in the Rosenblatt's perceptron for the binary classification was based on the Perceptron convergence theorem. The idea was to find the upper and lower bounds along the length of the weight vector by updating the weights accordingly (Haykin, 2008). An important variation of the perceptron training algorithm was introduced by (Widrow & Hoff, 1960) who developed a model, called ADALINE, based on the least-mean-square (LMS) algorithm. The main difference from the perceptron is the way the output of a neural system is used in the learning rule, known as Delta rule. These two were the first training algorithms specific to single-layer neural networks. Minsky and Pitts's work showed that these techniques do not work for multilayer networks but didn't provide a solution to the problem of how to adjust weights in hidden layers that are not connected to the output. The issue of training multilayer networks remained open until the idea of propagating the error in the opposite direction of the forward process with the explanation of reverse calculations became the subject of close examination in several works since the early 1970s.

2.2.1 Backpropagation

A valuable contribution to overcoming restrictions of previous algorithms for training multilayer networks was first made by Paul Werbos, who proposed a backpropagation training technique published in his PhD thesis (Werbos, 1974). This work remained almost unknown in the scientific community until the method was again explored by David Parker who published his findings in the technical report (Parker, 1985). Finally, in 1986, a well-known paper on backpropagation was published by Geoff Hinton, David Rumelhart and Ronald Williams (Rumelhart et al., 1986). The idea conceived by people in the past was concisely stated and led to a clear framework which enhanced its popularization (Widrow & Lehr, 1990). The paper of 1986 showed that backpropagation worked much faster than previous approaches to learning. It has been eventually considered the starting point of the contemporary Deep learning theory (Wang & Raj, 2017; Beam, 2017).

Backpropagation belongs to the iterative optimization algorithms that progressively improve the results with respect to the objective function. The objective function ($\boldsymbol{J}$), also called loss or cost function, is a difference between estimated ($\boldsymbol{y}'$) and true values ($\boldsymbol{y}$) of the target. In

other words, it shows how much different our mappings from actual values are. The generic relationship can be denoted as follows:

$$J = y - y' \tag{2.10}$$

In applied tasks, different loss functions are used depending on the problem at hand. The most used ones are cross-entropy and mean squared error or mean absolute error (Goodfellow et al., 2016).

Changing the parameters towards the optimal solution minimizing the cost function is basically what the training process is:

$$J = min(f(x, \theta)), \tag{2.11}$$

where $\boldsymbol{\theta}$ is parameters $\boldsymbol{w}$ and $\boldsymbol{b}$.

To find an optimal combination of parameters by checking all possible combinations is a very exhaustive process. The point is to figure out which way is downhill to understand whether to increase or decrease the value of weights to minimize the cost function. Knowing the direction in which to move, rather than applying countless combinations of weights, leads to a faster optimization. This numerical estimation is called a *gradient descent*. Graphically, this is reflected by the slope of the loss function at a certain point corresponding to a specific weight value $\boldsymbol{w_i}$ (Figure 2.6).

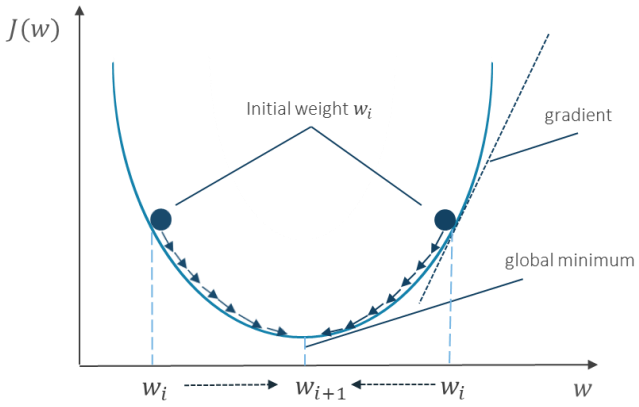


Figure 2.7 The gradient of the loss function
Source: adapted from (Raschka, 2015)

This way, the change in the loss function at each point with respect to the parameter change is nothing other than a derivative of the function with respect to weights. Parameters will be

adjusted incrementally to a step size value $\boldsymbol{\alpha}$ which is called learning rate, by increasing (2.12) or decreasing (2.13) in a direction of the gradient.

$$w_{i+1} \leftarrow w_i + \alpha \frac{\partial J}{\partial w_i} \quad (2.12) \qquad w_{i+1} \leftarrow w_i - \alpha \frac{\partial J}{\partial w_i} \quad (2.13)$$

Since the derivative of the function is a product of derivatives of several terms, the concept of a total derivative is applied. The process becomes apparent when applying the chain rule to calculate the derivative of each component. First, the derivative of the cost function with respect to the parameters of the last hidden layer are calculated. The gradients for the output layer directly affect the cost function. Using the chain rule, the derivative of the cost function with respect to the weights in the last layer is a result of three components. The first component is a derivative of the cost function with respect to the output. Its calculation depends on the exact formula of the chosen cost function. The second component is a derivative of the activation function which will be assumed as sigmoid $\boldsymbol{\sigma}$ for ease of presentation. The third component is a derivative of linear function and it corresponds to the activation from the previous layer. This is reflected by equation 2.14:

$$\frac{\partial J}{\partial w_{jk}^L} = \frac{\partial J(y', f\left(z_j^L\left(w_{jk}^L\right)\right))}{\partial w_{jk}^L} = \frac{\partial J}{\partial y'} * \frac{\partial a_j^L}{\partial z_j^L} * \frac{\partial z_j^L}{\partial w_{jk}^L} \quad (2.14)$$

where $\boldsymbol{L}$ is the last layer in the sequence $l = 1, 2, \ldots L$.

According to common conventions, the symbol $\boldsymbol{\delta^l}$ is used to denote the error associated with the layer $\boldsymbol{l}$. This is the value that network parameters will be adjusted to. The partial derivative of the error function with respect to weights in the final layer has a following form:

$$\frac{\partial J}{\partial w_{jk}^L} = \delta^L * \frac{\partial a_j^L}{\partial z_j^L} * \frac{\partial z_j^L}{\partial w_{jk}^L} = \delta^L a_j^{L-1} \quad (2.15)$$

For the hidden layers, the chain rule for multivariate functions is applied again:

$$\frac{\partial J}{\partial w_{jk}^l} = \frac{\partial J}{\partial a_j^l} * \frac{\partial a_j^l}{\partial z_j^l} * \frac{\partial z_j^l}{\partial w_{jk}^l} = \frac{\partial J}{\partial a_j^l} \sigma'\left(z_j^l\right) a_k^{l-1} \quad (2.16)$$

The derivative of the cost with respect to the activation $\frac{\boldsymbol{\partial J}}{\boldsymbol{\partial a_j^l}}$ is a part of the input to the subsequent layer $\boldsymbol{m}^{l^{th}}$, so its contributions will be summed in the next layer:

$$\frac{\partial J}{\partial a_j^l} = \sum_m \frac{\partial J}{\partial z_m^{l+1}} * \frac{\partial z_m^{l+1}}{\partial w_j^l} = \sum_m \delta_m^{l+1} w_{mj}^{l+1} \tag{2.17}$$

In the propagated sequence all deltas are connected in the recursive formula where the delta of the previous layer is a function of the delta in the subsequent layer:

$$\delta_j^l = (\sum_m \delta_m^{l+1} w_{mj}^{l+1}) \sigma'(z_j^l) \tag{2.18}$$

The generalized formula for updating the weights will take a form:

$$\Delta w_{jk}^l = \alpha \frac{\partial f(x, \theta)}{\partial w_{jk}^l} \tag{2.19}$$

Summarizing the backpropagation algorithm, the following steps can be identified:

1. *Forward path*. Computing the sequence of inputs and outputs in each of the layers by summing weighted inputs and adding nonlinearity.
2. *Backward path*. Obtaining an error term in the final layer and backpropagating it to the hidden layers by reapplying equation (2.18).
3. *Combining all the gradients* in each layer to get the total gradient.
4. *Updating weights* according to the learning rate and total gradient by applying (2.19).

Chapter 3
Autoencoder framework

This chapter provides an overview of autoencoders and their varieties. The goal is to explain the difference between autoencoders based on principles of how they learn the representation of data rather than focusing on the mathematical explanation of parameters introduced for this purpose.

3.1 Overview

The idea behind autoencoders is to reconstruct the input data in the output with the least possible distortion (Baldi, 2012). Formulated this way, it may seem useless but the expectation is that useful properties of the input will be extracted in the course of training.

An autoencoder is respectively composed by three components: an encoder, code, and decoder. The encoder compresses the input and generates the code, the decoder reconstructs the input based on the code (Figure 3.1). The training process is similar to training feedforward neural network via backpropagation (Goodfellow et al., 2016). Autoencoders by their nature are lossy which means that the output will degrade compared to the original input. The goal of training is to minimize the reconstruction error.

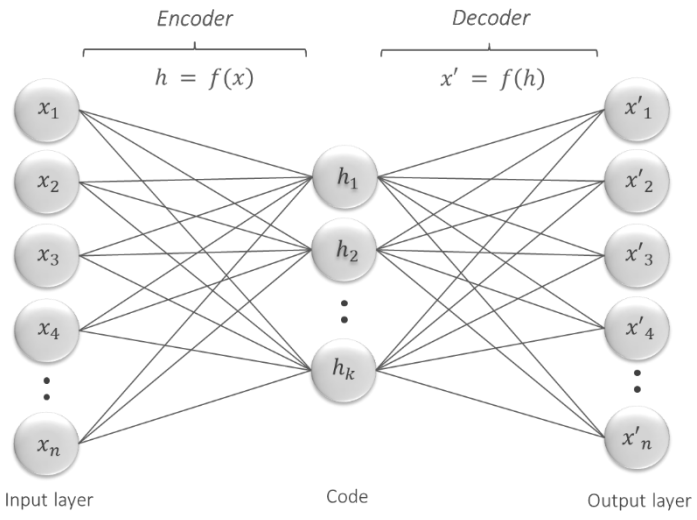


Figure 3.1 Autoencoder architecture

3.2 Autoencoder algorithm

Each input in the autoencoder is represented by a vector $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{n}}$ of a dimension $\boldsymbol{n}$ equal to the input size. The autoencoder takes the input and first maps it to a hidden representation or code through a deterministic mapping, called *the encoder* which can be viewed as a function:

$$h = f_\theta(x) = \sigma(xW + b), \tag{3.1}$$

where $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{b}\}$; $\boldsymbol{W} \in \mathbb{R}^{\boldsymbol{mxn}}$ is a weight matrix; $\boldsymbol{b} \in \mathbb{R}^{\boldsymbol{m}}$ is a bias vector; $\boldsymbol{f_\theta(x)}$ – the encoder; $\boldsymbol{\sigma}$ is an activation function[1].

The hidden representation is mapped back with *the decoder* reproducing the output layer $\boldsymbol{x'} \in \mathbb{R}^{\boldsymbol{n}}$ of the same dimension $\boldsymbol{n}$ as the input. This process can be written as a function:

$$x' = g_{\theta'}(h) = \sigma(hW' + b'), \tag{3.2}$$

where $\boldsymbol{\theta'} = \{\boldsymbol{W'}, \boldsymbol{b'}\}$; $\boldsymbol{g_{\theta'}(h)}$ – the decoder.

One of the approaches to constrain the weight matrix of the decoder $\boldsymbol{W'}$ is to transpose the encoder weight matrix $\boldsymbol{W' = W^T}$. This is referred to as tied weights. The use of tied weights is optional and some experiments with untied weights have yielded similar results (Vincent et al., 2010).

3.3 Parametrization of Autoencoder

To train an autoencoder, several parameters must be pre-set. These are parameters that determine the architecture of an autoencoder (code size, number of layers) and training parameters (loss function, activation function at each layer, optimizer, regularizer).

3.3.1 Code size

The code size is a number of units in the middle layer of an autoencoder or the last layer of the encoder. By the size of the code, autoencoders can be classified into two groupings - undercomplete and overcomplete autoencoders.

[1] The symbol $\boldsymbol{\sigma}$ in this equation is referred to any activation function

3.3.1.1 Undercomplete Autoencoders

Undercomplete autoencoders have a code size smaller than the input size. These autoencoders are designed to capture useful features in the data and reduce its dimensionality by representing the whole population with captured silent features. In this case, the network is encouraged to learn some sort of compression.

3.3.1.2 Overcomplete Autoencoders

The dimensionality reduction argument is built on the assumption that the code dimension is smaller than the input. Nevertheless, even with the opposite statement, an autoencoder can learn useful information about the data structure by imposing some constraints on the activity of the hidden representations (Ng, 2011). When the architecture is properly adjusted by adding the regularization terms, the autoencoder gains other qualities besides its capacity to copy the input to the output. These autoencoders will be considered in section 3.3.6 when different types of regularization are introduced.

3.3.2 Number of layers

Depending on the number of hidden layers, autoencoders can be divided into deep and shallow. A shallow autoencoder has just one hidden layer and it can be viewed as a Restricted Boltzmann machine (Hinton, 2012c), especially when trained with contrastive divergence (Hinton, 2002). The first deep architectures were trained precisely as stacked RMBs. In the following subsections, the evolution from shallow to deep architectures is considered, and the main difference between stochastic and deterministic algorithms is defined.

3.3.2.1 Restricted Boltzmann machine

The Boltzmann machine was introduced in the paper (Ackley, Hinton, & Sejnowski, 1985) with subsequent variations that have surpassed their original version (Goodfellow et al., 2016), and underlie the modern deep learning (Heaton, 2015).

A Boltzmann machine is a kind of fully connected neural networks that can be considered as a stochastic generative variant of the Hopfield network (Hopfield, 1982) in which neurons are both, input and output. The Boltzmann machine consists of two layers – visible $\boldsymbol{v} \in \mathbb{R}^{\boldsymbol{n}}$ and

hidden $\boldsymbol{h} \in \mathbb{R}^{\boldsymbol{k}}$ (Figure 3.2). The inputs are binary vectors and the learning algorithm is based on maximum likelihood (Goodfellow et al., 2016).

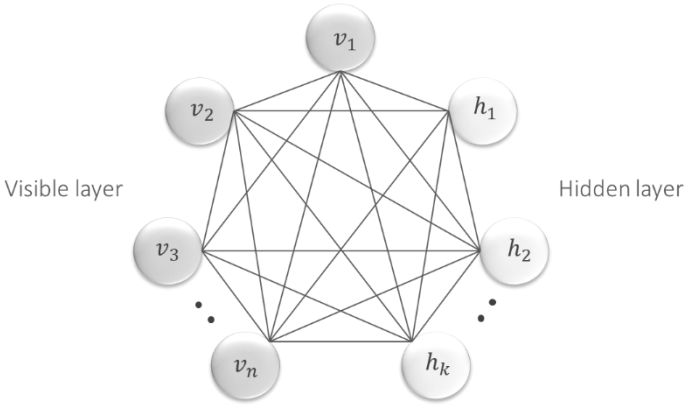


Figure 3.2 Boltzmann machine

One of the varieties of Boltzmann machines is a Restricted Boltzmann machine (RBM) which was introduced by Paul Smolensky under the name harmonium (Smolensky, 1986). The main difference is that there are no connections between neurons in the same layer (Figure 3.3).

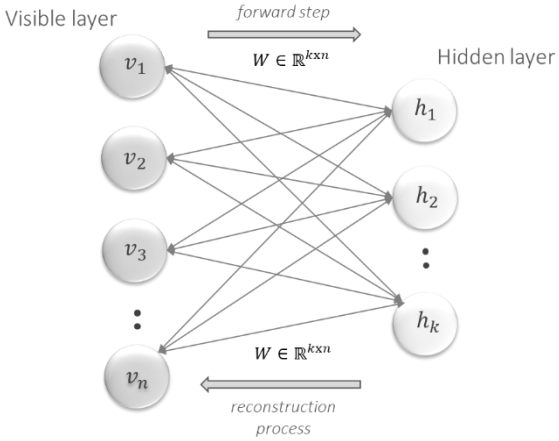


Figure 3.3 Restricted Boltzmann machine

The training algorithm consists of several forward and backward passes in a fashion of training a feedforward network with the difference that an RBM does not produce the output. The hidden layer is used to model the input data, and then, to reconstruct it in the visible layer. This implies a bi-directional relationship between layers (Hinton, 2012b) which is reflected by so-called bipartite graph (Figure 3.3). The fact that RBMs are undirected and neurons are activated probabilistically makes them a special case of Markov networks (Hinton, 2012a).

3.3.2.2 Deep belief network

In 2006, the paper (Hinton & Osindero, 2006) demonstrated how several RBMs can be stacked together and trained in a greedy manner to form a deeper structure, called Deep Belief Network (DBN). When two layers of the RBM are part of a deeper neural network, the output of the hidden layer is passed as the input to the next hidden layer, and from there through as many hidden layers as defined. These feed-forward movements form an architecture that resembles an autoencoder, however there is an essential difference between them. The goal of training a DBN is to reproduce the distribution of input data based on activations in the hidden layer using a stochastic approach, whereas an autoencoder is deterministically learning the representation of the input data. In other words, the hidden layer of a DBN is a probabilistic distribution among the hidden variables and the hidden layer of an autoencoder is a representation of the input data.

The Deep Belief Network is a hybrid involving both, directed and undirected connections. The presence of directions distinguishes DBNs from Deep Boltzmann machines (Salakhutdinov & Hinton, 2009) that are fully undirected.

DBNs are not commonly used in the modern architectures but they played a fundamental role in the history of deep learning (Goodfellow et al., 2016) and considered the predecessors of contemporary generative algorithms (Salakhutdinov, 2015).

3.3.2.3 Stacked autoencoder

The first deep networks were trained in a greedy layer-wise manner. The papers (Hinton & Salakhutdinov, 2006; Bengio et al., 2007) showed how using this method, one can obtain the representation that yields a better performance in the supervised learning. This approach underlies the stacked autoencoder where several codes of a vanilla encoder are combined into a more complex structure (Figure 3.4).

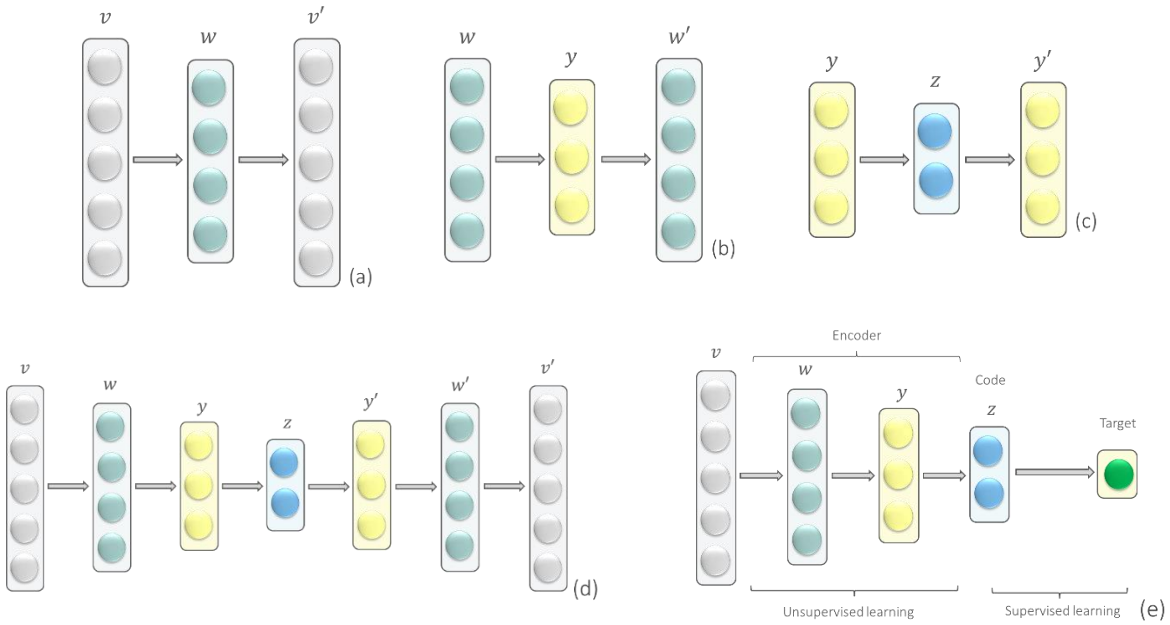


Figure 3.4 Training a Stacked Autoencoder

The algorithm of a stacked autoencoder can be described as follows:

1. Each layer is trained at a time to preserve as much information of input as possible (Figure 3.4 (a), (b), (c)).
2. The restoring layers v', w', y' are discarded in each training procedure.
3. The weights of each hidden layer $\boldsymbol{w}, \boldsymbol{y}, \boldsymbol{z}$ are fixed, and the next network is built based on data coming from the previous hidden layer which now acts as the input layer.
4. All hidden layers are combined forming a deep structure (Figure 3.4 (d)).
5. The fine-tuning of an entire network with respect to the final measure of interest is performed (Figure 3.4 (e)). Here, the transition from unsupervised to supervised learning is taking place. The full neural network processes all layers of the deep encoder as a single entity in such a way that weights are improved in the stacked autoencoder.

3.3.3 Loss function

The objective function evaluates the effectiveness of training neural networks. It returns a score that indicates how well a network performs. In the case of autoencoder, it measures how good the reconstruction is, reason why the term *loss function* is more adequate. Here, the commonly used loss functions will be considered, although the choice of functions is a lot richer.

3.3.3.1 Mean squared error loss function

Typically used for real-valued inputs, the function $\boldsymbol{L}$ represents the sum of squared Euclidian distances between the input vector $\boldsymbol{x}$ and reconstructed vector $\boldsymbol{x'}$, as shown by equation (3.3).

$$L(x, g(f(x))) = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x'}_i - x_i)^2, \tag{3.3}$$

where $\boldsymbol{g(h)}$ is the decoder output, $\boldsymbol{h = f(x)}$ is the encoder output or code.

When the input is real values $[-\infty;\ +\infty]$ it is recommended to use a linear activation function in the last reconstruction layer (Larochelle, 2013).

3.3.3.2 Cross entropy loss function

Typically used for binary inputs, the loss function represents the sum of Bernoulli cross-entropies between two distributions:

$$L(x; g(f(x))) = -\frac{1}{n}\sum_{i=1}^{n}(x_i log(x'_i) + (1 - x_i) log(1 - x'_i)) \tag{3.4}$$

This function is also applicable when inputs belong to a range $[0; 1]$ (Larochelle, 2013).

In literature, one can be confronted with the term *negative log likelihood*. The cross entropy and negative log likelihood have different origins, from information theory and statistical modeling, respectively, while mathematically they are exactly the same (Gibson & Patterson, 2017).

3.3.4 Optimizer

These are the algorithms that try to find optimal values of mathematical functions used in training a neural network. The gradient descent algorithm is one of the optimization algorithms. It was considered in section 2.2.

There are different *variants* of the gradient descent algorithm with respect to the amount of data taken per update to compute the gradient, and *extensions* of the gradient descent based on different approaches to adapting the learning rate and selecting hyperparameters. We do not go into detail here and just summarize some of them (Table 3.1, Table 3.2).

Table 3.1 Variants of the gradient descent (GD) based on the amount of data
Source: (Ruder, 2016)

Method	Amount of data per update	Update speed	Memory usage	Online learning
Stochastic GD	one sample	High	Low	Yes
Batch GD	entire dataset	Slow	High	No
Mini-batch GD	around 50-250 samples	Medium	Medium	Yes

Table 3.2 Extensions of the gradient descent
Sources: (Portilla & Mosconi, 2017; Sokolov et al., 2017)

Method	Description
Gradient descent	Basic unmodified GD
Momentum	Smooth updates. Tends to move in the same directions as on previous steps
Nesterov Momentum	Update slows down before changing direction, making a small corrective jump based on gradient
AdaGrad	Separate learning rates for each dimension. Adaptive correction using squared gradient. Suits for sparse data
RMSprop	Learning rate adapts to latest gradient steps. EWMA[2] applied to squared gradient AdaGrad
Adam	Combines Momentum and individual learning rates. Use EMWA on 1st and 2nd moments

3.3.5 Regularization

A strategy aimed at improving the performance of algorithms not only on training but also on new data is known as regularization. There are various regularization methods that generally based on modifying the basic algorithm by adding some constraints. The following subsections discuss regularizers applicable to autoencoders.

3.3.5.1 Sparse autoencoder

The concept of sparsity was introduced in computational neuroscience (Olshausen & Field, 1997), and subsequently arose in different contexts: unsupervised learning of sparse features represented by the linear code (Ranzato et al., 2006); as a variant of DBNs in the context of convolutional networks (Ranzato, 2007; Lee & Ng, 2008; Mairal et al., 2009).

The overcomplete architecture of a sparse autoencoder allows a larger number of hidden units in the code, but this requires that for the given input, most of hidden neurons result in very

[2] Exponentially Weighted Moving Average

little activation (Ng, 2011). For each hidden node, the average activation value should be close to zero (in the case of sigmoid or to -1 when activation function is tanh). The neuron will be considered active if the output is close to 1, and otherwise – inactive (Ng, 2011).

What is the goal of having hidden units with most zeros? The idea is to make a neuron activated only for a small fraction of the training examples. Since the samples have different characteristics, the activation of neurons should not be held in the same fashion in all neurons and must be coordinated. The goal is to obtain a latent representation with many zeros and only a few non-zero elements that will represent the most protruding features (Figure 3.5).

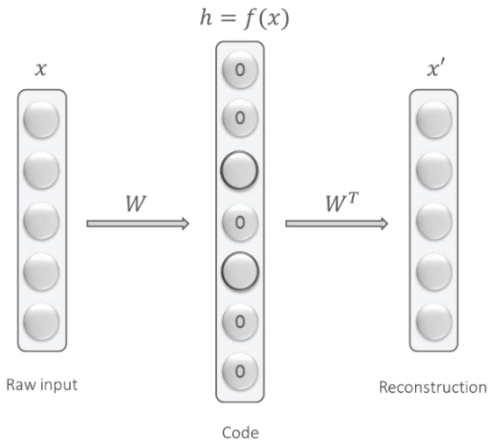


Figure 3.5 Sparse Autoencoder

The training of a sparse autoencoder involves a penalty term on the code layer to reflect deviation from a desired sparsity. The penalty is added to the loss function modifying the error:

$$L\left(x; g\left(f(x)\right)\right) + \Omega(h), \tag{3.5}$$

where $\boldsymbol{g(h)}$ is the decoder output, $\boldsymbol{h = f(x)}$ is the encoder output or code, $\boldsymbol{\Omega(h)}$ is a sparsity penalty which is a log function (3.6; 3.7).

$$\Omega(h) = \sum_{j=1}^{s} KL(p||p'_j), \tag{3.6}$$

where $\boldsymbol{p}$ is a sparsity parameter, typically a small value close to zero; $\boldsymbol{p'_j}$ is the average activation of hidden unit $\boldsymbol{j}$, that is forced to approximate to $\boldsymbol{p}$; $\boldsymbol{s}$ is the number of neurons in the hidden layer; $\boldsymbol{KL}$ is Kullback-Leibler divergence (3.7) between a Bernoulli random variable with mean $\boldsymbol{p}$ and a Bernoulli random variable with mean $\boldsymbol{p'}$ (Ng, 2011).

$$KL(p||\hat{p}_j) = plog\frac{p}{p'_j} + (1-p)log\frac{1-p}{1-p'_j} \tag{3.7}$$

The effects of introducing a sparse component into the network were explored and visualized using the MNIST dataset in the paper (Makhzani & Frey, 2014). In the hidden layer, only $\boldsymbol{k}$ highest activities were kept. The results showed that with a higher level of sparsity, the network tended to capture local fragments of digits, and with decreasing values of $\boldsymbol{k}$, it was forced to learn a more complete representation of each digit.

A method of creating a sparse representation by achieving true zero in the code layer was introduced in the paper (Glorot et al., 2011) where the ReLU activation function was used for this purpose. Researchers highlighted several reasons why a sparse representation might be appealing, thereby enriching the conceptual part about the need of sparsity – among others, information disentangling and efficient variable-size representation (Glorot et al., 2011).

3.3.5.2 Denoising autoencoder

The idea of training a network by adding noise was approached in the works (Gallinari et al., 1987) and (Lecun, 1987). Later, the denoising task was applied to more complex convolutional and recurrent networks (Seung, 1998; Jain & Seung, 2008) with continuing study of this parameterization on autoencoders (Alain & Bengio, 2014).

Denoising autoencoders are trained to map a noisy input to an output. The noise in the input layer is added to force the autoencoder to learn the most robust features and distinguish them from noise (Vincent & Larochelle, 2008). The input in this case is a corrupted version $\tilde{\boldsymbol{x}} \in \mathbb{R}^{\boldsymbol{n}}$ of the original input $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{n}}$ (Goodfellow et al., 2016). The denoising autoencoder does not simply copy the input to its output but cleans data from noise and then, reproduces the input from its corrupted version (Figure 3.6).

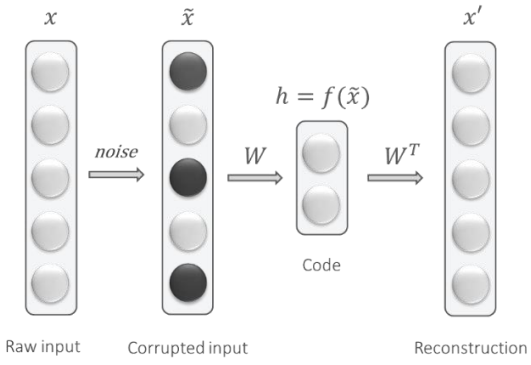


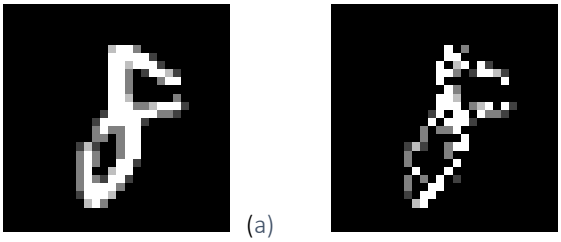
Figure 3.6 Denoising autoencoder

The loss function minimizes error not from the original but from the corrupted input and takes the following form:

$$L(x, g(f(\tilde{x}))) \tag{3.8}$$

where $\boldsymbol{g}(\boldsymbol{f}(\tilde{\boldsymbol{x}}))$ is the decoder output, $\boldsymbol{f}(\tilde{\boldsymbol{x}})$ is the encoder output encoded from the corrupted input $\tilde{\boldsymbol{x}}$.

Figure 3.7 visualizes the idea of a denoising autoencoder: the noisy version reveals some pixels missing but it is still clearly visible unequivocally interpretable digit's shape.



(a) (b)

Figure 3.7 Denoising effect
(a) original version of the digit from MNIST dataset; (b) corrupted version of the same digit.

3.3.5.2.1 Dropout regularizer

One of the varieties of denoising is a dropout method. In the section 3.3.5.2, there have been considered denoising autoencoders where the noise is injected only into the input layer. The additional introduction of noise into hidden layers turns to be a powerful regularization tool for networks and considered as one of the ways to prevent a network from overfitting (Srivastava et al., 2014).

3.3.5.2.2 Noise

There are different forms of noise to implement corruption. Here, they will not be discussed in detail but some generally applicable techniques (Vincent et al., 2010) are outlined in the following subsections.

Gaussian noise

One of the common statistical noise choices which, by its nature, relies on the fact that values the noise can take are normally distributed. It's described in several works applied to autoencoders (Vincent, 2011; Rifai et al., 2011) and compared with other types of noise added to neural networks (Poole, Sohl-dickstein, & Ganguli, 2014).

Masking noise

Another way to add noise is to randomly set some of the inputs to zero and left others as they are (Vincent et al., 2010). Consequently, the autoencoder will try to predict artificially assigned "blanks" from non-missing values. This type of corruption is called masking because assigning zero to inputs means that they are completely ignored (masked) in the computation of subsequent layers (Vincent, et al., 2010).

Salt-and-pepper noise

Mainly associated with images, this type of noise reveals itself in randomly occurring black and white pixels on the image. Mathematically, a random fraction of inputs is set to their minimum and maximum value. Applying to images, this noise replaces pixels with black or white pixels, regardless of other pixels and their original color (Gonzalez et al., 2007).

3.3.5.3 Contractive Autoencoder

The contractive autoencoder was introduced in the paper (Rifai et al., 2011). This type of autoencoders implies the introduction of a regularization term that forces a network to learn representation features that are robust towards small changes around the input (Rifai et al., 2011). This is achieved by adding a penalty term to the loss function (3.9) that encourages the derivatives of encoding function to be as small as possible, whereas a denoising autoencoder does the same for the reconstruction function (Goodfellow et al., 2016). The penalty term differs from the penalty used in the sparse autoencoder and corresponds to the Frobenius norm of the Jacobian matrix (3.10) of the encoder activations with respect to the input (Rifai et al., 2011).

$$L\left(x; g\left(f(x)\right)\right) + \Omega(h), \tag{3.9}$$

where $\boldsymbol{g(h)}$ is the decoder output, $\boldsymbol{h = f(x)}$ is the encoder output, $\boldsymbol{\Omega(h)}$ is a penalty term which is the squared Frobenius norm (sum of squared elements):

$$\Omega(h) = \lambda \left\| \frac{\partial f(x)}{\partial x} \right\|_F^2 \tag{3.10}$$

where $\boldsymbol{\lambda}$ is a hyper-parameter that controls the strength of the regularization.

Since a contractive autoencoder is trained to resist disturbances to its input, it is forced to map the neighbourhood of the input points to the contractive neighbourhood in the hidden layer, making the mapping not too sensitive and giving rise to a name of this regularized autoencoder. The paper (Alain & Bengio, 2014) showed that a denoising autoencoder with very small Gaussian corruption and squared error loss is a kind of contractive autoencoder.

3.4 Other Autoencoders

In this section, a brief overview of other autoencoders will be provided to preserve the logic of the classification according to different criteria. The core of these autoencoders is neural networks that are fundamentally different in terms of how information is processed between layers. These are the major classes of neural networks each of which merits a special chapter. It is appropriate mentioning them here since they can be used in the autoencoder architecture.

3.4.1 Variational Autoencoder

This type of autoencoders belongs to a specific class of algorithms - generative models of data (Goodfellow et al., 2016). In a very simplified form, these are the models that are trained to generate plausible looking fake samples that resemble training data. They are called autoencoders because they have constituent parts of a traditional autoencoder – encoder and decoder – however, mathematically do not have much in common with it (Doersch, 2016).

A variational autoencoder (VA) is a generative model that learns a latent variable model of its input by adding constraints on the encoded representation. It was introduced as a variational Bayesian approach that involves the optimization of an approximation (Kingma & Welling, 2013) and can be trained with gradient-based methods (Rezende et al., 2014).

The encoder part of the variational autoencoder takes a sample represented by a vector $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{n}}$, passes through the encoding network and produces a probability distribution in the latent space. This distribution is Gaussian (Kingma & Welling, 2013). In practice, this is reflected in an additional variational layer consisting of a mean vector $\boldsymbol{\mu} \in \mathbb{R}^{\boldsymbol{m}}$ and standard deviation vector $\boldsymbol{\sigma} \in \mathbb{R}^{\boldsymbol{m}}$ of the same dimension as the latent vector. For each data sample, a point in the latent space is sampled from the distribution and it is represented by a vector $\boldsymbol{z} \in \mathbb{R}^{\boldsymbol{m}}$ which is fed into the decoder. As a result of the decoding process, a new sample is obtained, and it resembles the input sample (Figure 3.8).

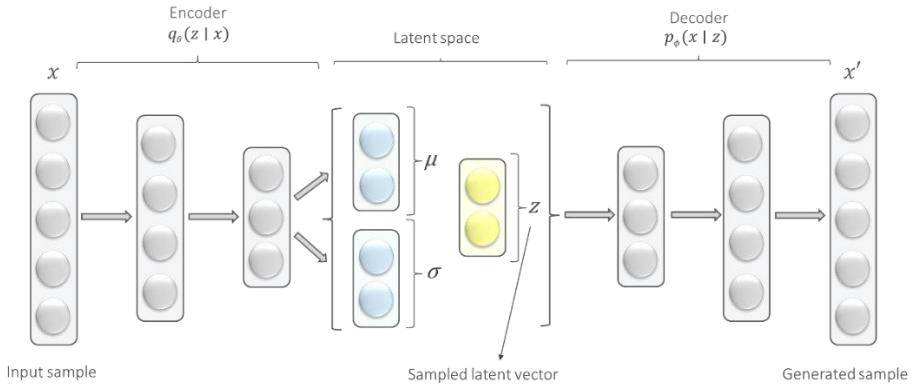


Figure 3.8 Variational autoencoder

The generative model of the variational autoencoder is a probabilistic model and is not considered here in detail, but it is worth noting a key difference from the vanilla autoencoder that the variational autoencoder is a stochastic model where a probabilistic encoder $\boldsymbol{q_\theta(z \mid x)}$ approximates the true posterior distribution $\boldsymbol{p(z|x)}$, and a generative decoder $\boldsymbol{p_\phi(x \mid z)}$ does not rely on any particular input $\boldsymbol{x}$. The idea is to ensure that the decoder can decode any input, and for this, one needs to define the distribution of inputs that the decoder should expect (Doersch, 2016). In literature, the variational autoencoder algorithm is referred to as variational approximation to inference (Rezende et al., 2014) which literally means prediction of latent representations given new data by approximating a posterior distribution over the latent units given the input data. Generative models are in active research and used in a wide range of interesting applications.

3.4.2 Convolutional autoencoder

This is an autoencoder based on convolutional neural network (CNN). Convolutional architectures assume that inputs are data that has a grid-structured topology (Goodfellow et al., 2016) such as images. This allows to take advantage of the graphical object structure and encode its specific properties. CNNs were introduced by (Fukushima, 1980) under a different name, and became the basis for modern convolutional networks (LeCun et al., 1989; LeCun et al., 1998).

A CNN consists of convolutional and subsampling layers optionally followed by a fully connected layer (Figure 3.9). The convolutional process is implemented to create a feature (activity) map from the input by extracting local receptive fields across the entire image. After activation in hidden neurons, the pooling step is applied resulting in dimensionality reduction

of the feature map by condensing outputs of small regions of neurons into a single output. This is considered as one layer of a CNN (Ng et al., 2017).

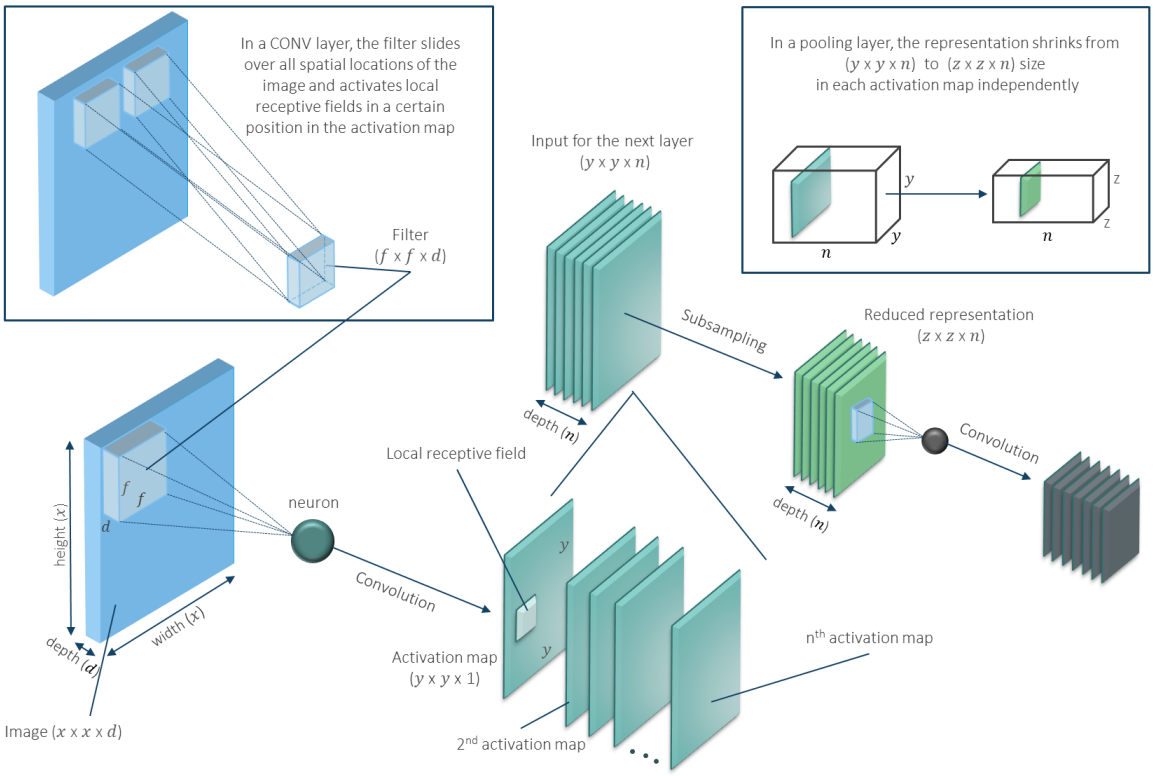


Figure 3.9 Convolution and pooling in CNN

Each subsequent layer increases a complexity of the learned feature map. When high-level features are detected a fully connected layer is attached to the network. In supervised learning, the fully connected layer accesses the output of the previous layer and defines properties that are more related to a particular class of objects (Krizhevsky et al., 2012).

In unsupervised learning, the middle subsampling layer is connected to the decoder and proceeds with the convolutional process again to reconstruct the input image (Figure 3.10). Since training a network with the image input from scratch requires huge computational power, a convolutional autoencoder was viewed as a technique of initializing weights in a neural network (Erhan, Courville, & Vincent, 2010). It is also widely used for its conventional purpose of feature extraction (Masci et al., 2011; Du et al., 2017).

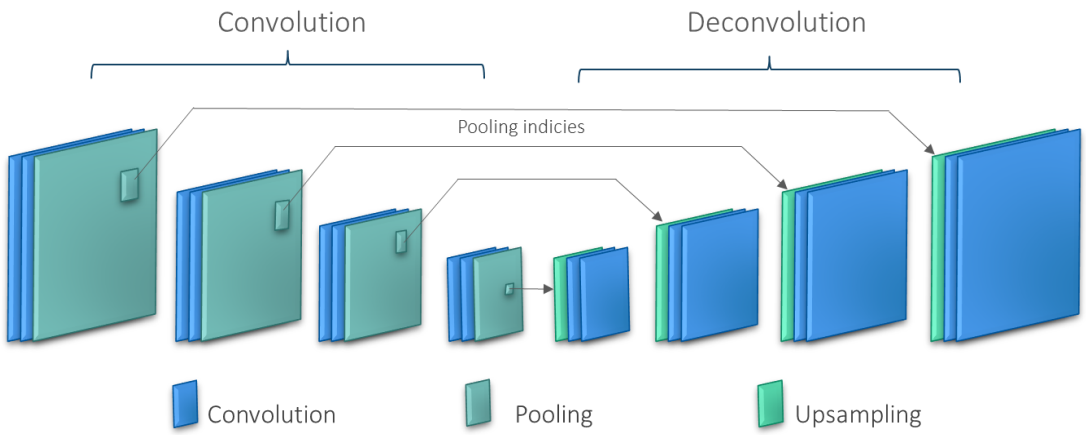


Figure 3.10 Convolutional autoencoder
Source: adapted from (Badrinarayanan, Kendall, & Cipolla, 2017)

3.4.3 Sequence-to-sequence autoencoder

Designed to be trained with sequential data, this autoencoder is based on another specialization of neural networks which is appropriate to handle sequences - recurrent neural network (RNN). The idea behind RNNs is to use sequential information where connections between elements form a directed circle (Jordan, 1986). There are several variants of RNNs can be found in literature: Jordan networks (Jordan, 1986), Elman networks (Elman, 1990), Bidirectional RNN (Schuster & Paliwal, 1997), Long-Short Term Memory networks (Hochreiter & Urgen Schmidhuber, 1997), Gated Recurrent Units (Chung, Gulcehre, Cho, & Bengio, 2015).

The specificity of training neural networks fed with sequential data is determined by the nature of this data. A time series can be viewed as a generated sequence of values monitored over time. The unstable character of data affects the output in a way that it depends not only on the fixed input but also on how data behaved in the past. In practice, this is reflected in the loops that allow information to persist (Figure 3.11). A sequential input is processed by applying a recurrent formula (3.11) at every time step.

$$h_t = f_W(h_{t-1}, x_t) \tag{3.11}$$

$$y_t = W_{hy} h_t \tag{3.12}$$

where $\boldsymbol{h_t}$ is a new state, $\boldsymbol{f_W}$ is a function with parameters $\boldsymbol{W}$, $\boldsymbol{h_{t-1}}$ is an old state, $\boldsymbol{x_t}$ is an input vector at some time step, $\boldsymbol{y_t}$ is a sequence of outputs.

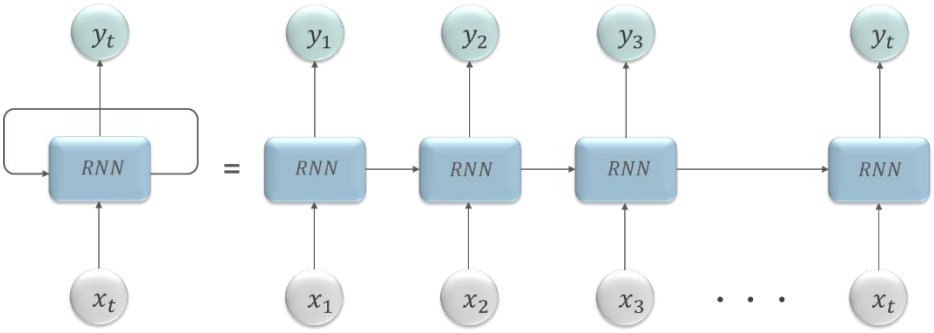


Figure 3.11 The unfolded recurrent neural network
Source: adapted from (Olah, 2015)

The augmentation of an RNN where the output is fed back to the input can suffer from a vanishing gradient problem (Hochreiter et al., 2001) and it's not able to capture long-term dependencies. To solve this problem, an approach of Long-Short Term Memory network (LSTM) has been proposed (Hochreiter & Urgen Schmidhuber, 1997).

RNNs are widely used in natural language processing tasks (Bengio et al., 2003; Socher et al., 2011; Mikolov et al., 2013). A sequence-to-sequence model (Seq2Seq) was originally applied to the problem of machine translation (Kalchbrenner & Blunsom, 2013). In (Cho et al., 2014), a Seq2Seq model was proposed and evaluated on the task of translating from English to French. In this model, based on statistical machine translation, the encoder is trained to obtain a semantically and syntactically meaningful representation of linguistic phrases and the decoder learns a continuous space representation of a phrase that preserves both, the semantic and syntactic structure of the phrase by predicting the next characters of the target sequence given previous characters (Cho et al., 2014). A similar approach was considered in the paper (Sutskever et al., 2014) demonstrated that a deep LSTM outperformed other RNNs.

3.5 Autoencoders summarized

To summarize the literature review about autoencoders, the following table is presented for ease of access:

Table 3.3 Literature review on Autoencoders covered in this thesis

Contribution	Papers
Hopfield network	(Hopfield, 1982)
Boltzmann machine	(Ackley et al., 1985)

Restricted Boltzmann machine	(Smolensky, 1986)
Auto-associative neural networks	(Bourlard & Kamp, 1988), (Baldi & Hornik, 1989)
Autoencoders, Helmholtz machine	(Hinton & Zemel, 1994)
Deep Belief Network	(Hinton & Osindero, 2006)
Stacked RBM for dimensionality reduction	(Hinton & Salakhutdinov, 2006)
Deep Boltzmann machine	(Salakhutdinov & Hinton, 2009)
Stacked autoencoders	(Bengio et al., 2007), (Bengio, 2009), (Vincent et al., 2010)
Sparse autoencoders	(Olshausen & Field, 1997), (Ranzato, 2007; Ranzato et al., 2006), (Lee & Ng, 2008), (Mairal et al., 2009), (Glorot et al., 2011), (Makhzani & Frey, 2014)
Denoising autoencoders	(Gallinari et al., 1987), (Seung, 1998), (Vincent & Larochelle, 2008), (Vincent et al., 2010), (Vincent, 2011) (Alain & Bengio, 2014)
Dropout regularizer	(Srivastava et al., 2014)
Contractive autoencoders	(Rifai et al., 2011)
Variational autoencoders	(Kingma & Welling, 2013), (Rezende et al., 2014)
Convolutional autoencoders	(Erhan et al., 2010), (Masci et al., 2011), (Du et al., 2017)
Sequence-to-sequence autoencoders	(Kalchbrenner & Blunsom, 2013), (Cho et al., 2014), (Sutskever et al., 2014)

Chapter 4
Methodology

This chapter presents major stages of the research process and describes how experiments were conducted. It provides information about the dataset, initial data comprehension via exploratory analysis of its variables, feature extraction and their visualization, training procedure details and finally, evaluation methods.

4.1 Research methodology main steps

The present study is framed in the phases illustrated on Figure 4.1.

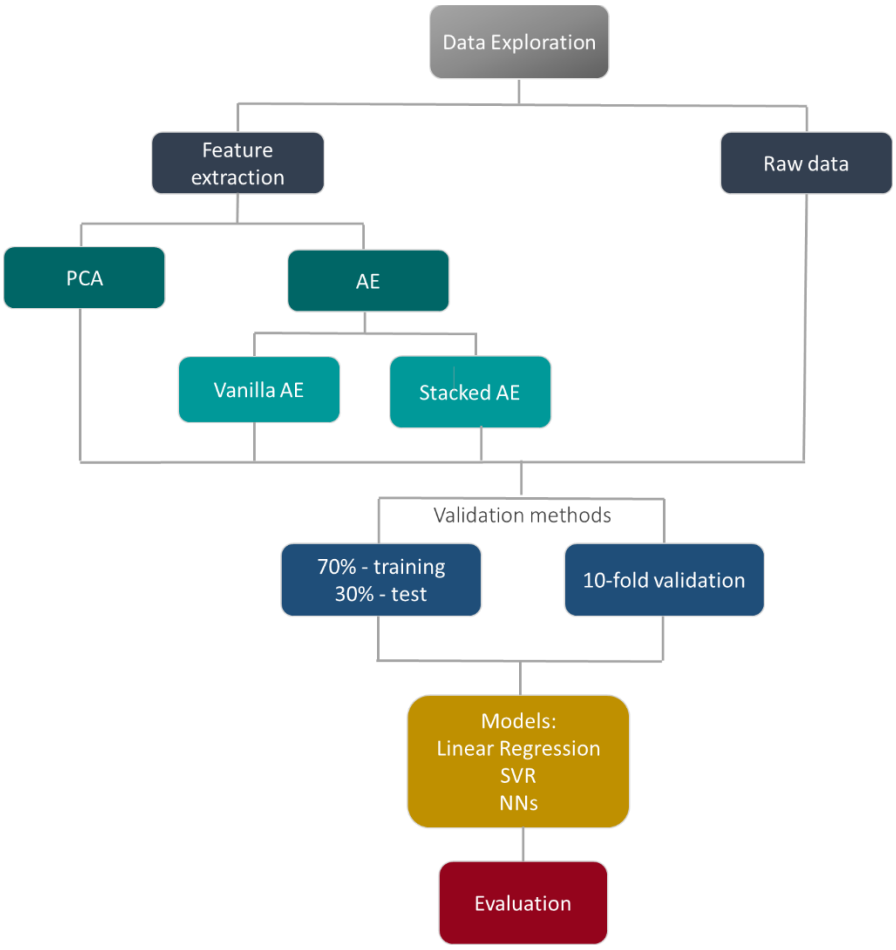


Figure 4.1 The core block diagram of the proposed methodology

4.2 Dataset

The dataset used for experiments contains 5875 voice recordings from 42 patients diagnosed with early-stage Parkinson's disease - 28 men and 14 women (about 200 recordings per patient). Each observation is described with following variables: patient number, age, gender, time interval from baseline recruitment date, 16 vocal attributes based on biomedical voice measures and Unified Parkinson's Disease Rating Scale (UPDRS) scores, used for tracking symptom progression (Table 4.1). This scale is the most widely used clinical rating scale for the Parkinson's disease (Goetz et al., 2003).

The dataset is available on the UCI Machine learning repository website. The main goal of data is to predict UPDRS scores. There are two UPDRS – total which refers to the full range of the metric, and motor which refers to the motor section of UPDRS. Their ranges are 0–176 and 0–108, respectively, from healthy status to total disability or severe motor impairment (Eskidere et al., 2012). Since the patients with early-stage of the disease were involved in the trial the maximum values of UPDRS are in the middle of scales – 54.992 and 39.511 for total and motor UPDRS, respectively (Appendix A).

Table 4.1 Variable description of Parkinson's telemonitoring dataset
Source: (Tsanas et al., 2010)

Variable	Description	Role	Type
subject#	Integer that uniquely identifies each patient	ID	ID
age	Patient's age	input	int
sex	Patient's gender	input	bin
test_time	Time since recruitment into the trial. The integer part is the number of days since recruitment.	input	int
Jitter(%)	Several measures of variation in fundamental frequency	input	int
Jitter(Abs)		input	int
Jitter:RAP,		input	int
Jitter:PPQ5		input	int
Jitter:DDP		input	int
Shimmer	Several measures of variation in amplitude	input	int
Shimmer(dB)		input	int
Shimmer:APQ3		input	int
Shimmer:APQ5		input	int
Shimmer:APQ11		input	int
Shimmer:DDA		input	int
NHR	Measures of ratio of noise to tonal components in the voice	input	int
HNR		input	int
RPDE	A non-linear dynamical complexity measure	input	int
DFA	Signal fractal scaling exponent	input	int
PPE	A nonlinear measure of fundamental frequency variation	input	int
motor_UPDRS	Clinician's motor UPDRS score, linearly interpolated	target	int
total_UPDRS	Clinician's total UPDRS score, linearly interpolated	target	int

4.3 Technical research tools

For this study, Spider 3.3.0 was used as a python integrated development environment. The main packages for statistical analysis and visualisation were Pandas 0.22, Matplotlib 2.2.2 and Seaborn 0.8.1. For scientific computation and implementation of machine learning algorithms, Numpy 1.14.2 and Scikit-learn 0.19.1 were used. Keras 2.0 with TensorFlow 1.7 (CPU) backend was chosen as a library for building neural networks and applying deep learning. The code for running experiments can be found to the url: https://github.com/veronique-ka/autoencoders_prk.

4.4 Initial data comprehension

The exploration of data is primarily aimed at better understanding it through describing, inspecting variables correlation and identification of outliers. For this purpose, variables were analysed through univariate descriptive statistics provided in Appendix A and then, the relationship between continuous variables were examined to determine the correlation via correlation coefficients and with the help of a visualization tool – the correlation matrix heatmap. Identifying the correlation between variables is one of the prerequisites for the application of PCA (Jolliffe & Cadima, 2016). The higher the correlation degree between original variables, the fewer components are required to explain the greater variation (Vyas & Kumaranayake, 2006).

Figure 4.2 shows the correlation matrix heatmap, and correlation coefficients are provided in Appendix B. Both indicate the presence of strong correlations between variables in data. This can imply the existence of multicollinearity among variables which can cause difficulties when building a regression model (Alin, 2010).

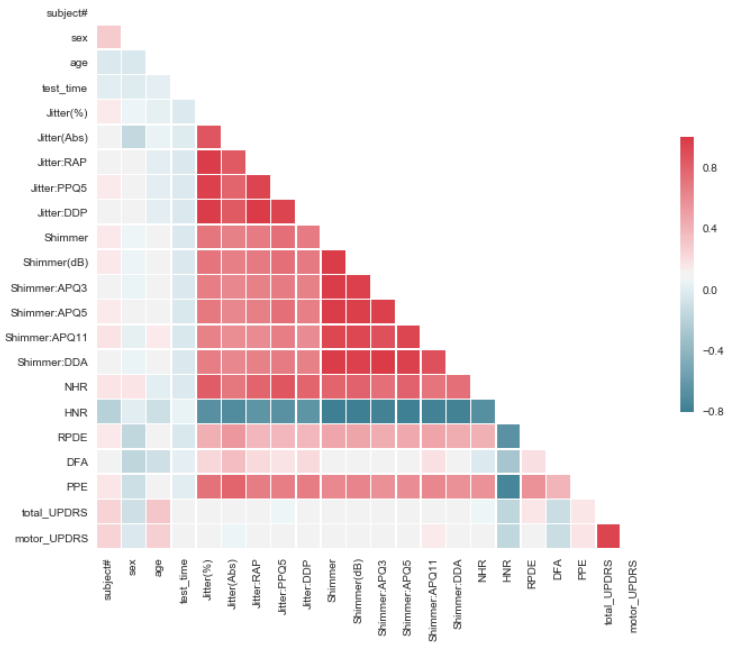


Figure 4.2 Correlation matrix of Parkinson's patient dataset presented with the heatmap

The most correlated variables were identified, and top-10 pairs presented in Appendix B. The detection of highly correlated variables is particularly helpful when deciding whether to include variables in the model if such is assumed. The present study is not aimed at building a predictive model but conducting a comparative analysis to evaluate how different sets of features can cope with the prediction of UPDRS score.

4.5 Extracting features with Principal component analysis

PCA is one of the most abundantly used techniques for reducing the dimensionality of data and extracting features to reveal a simplified data structure (Jolliffe & Cadima, 2016). This method involves a linear transformation of input variables resulting in the formation of independent components that preserve the most valuable part of the original variables.

In this study, the projection of data to a lower dimensional space has been implemented by the method of randomized truncated singular value decomposition proposed by (Halko et al., 2011), and using the probabilistic PCA model of (Tipping & Bishop, 1999). Before the decomposition, data was standardized by removing the mean value of each variable and scaling its variance. Then, the procedure was composed of following steps:

1. Performing PCA transformation on matrices consisting of vocal attributes (16 variables) and total number of attributes (19 variables);
2. Component selection.

There is a plethora of methods used to decide on the optimal number of components to use. In this study, the decision support has been primarily based on heuristic methods commonly used for measuring the quality of PCA. One of the standard measures is the proportion of the variance explained by each component. The list of variance values is presented in Appendix C. Via Table Appendix C-1, one can observe that the first principal component explains 70.38% of total variance; the second – 10.46%; the third – 7.75% and so on. Together, the first three components explain 88.59% of the total variance. The graphical representation of cumulative explained variance becomes an effective tool for substantiating the decision on the choice of components, though often leads to the use of just a few PCs. Through the curve on Figure 4.3 (a), a relative stabilization is observed after the fourth PC, so the cumulative variance explained doesn't increase significantly with each subsequent component.

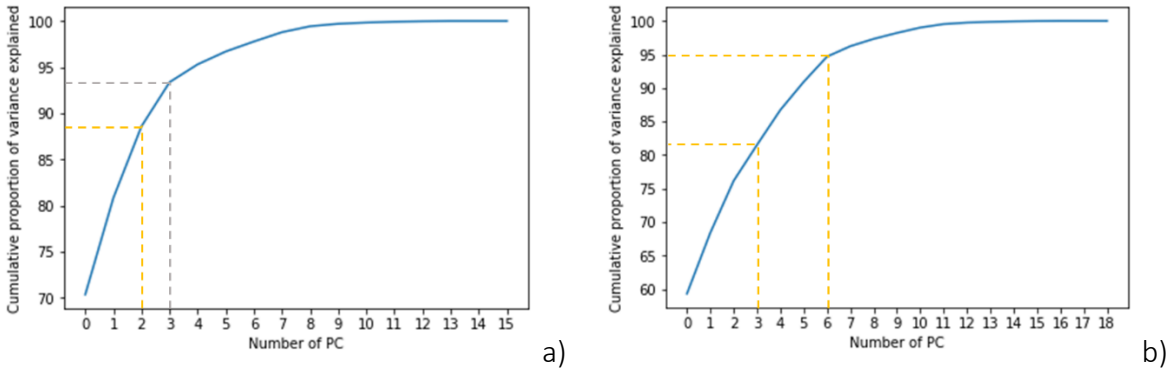


Figure 4.3 Visualization of the cumulative proportion of total variance explained by PCA
a) PCA based on 16 vocal attributes; b) PCA based on total 19 attributes

Another method used to determine the number of PCs is the Kaiser criterion (Kaiser, 1961). It suggests keeping components with eigenvalues larger than one. This approach is quite old and has undergone several criticisms since the 1960s (Preacher & MacCallum, 2003). Nevertheless, it is used as a reference point for deciding on PCs. That is, via Table Appendix C-1, the eigenvalues of the first three components have values above one. Based on both methodologies, it was decided to take 3 components for further analysis when the transformation is done based on 16 vocal attributes. As for the transformation out of 19 variables, the scenario is changing – the first four components have the eigenvalue above one,

and as per "the elbow rule", the stabilization of curve is observed after the seventh component (Figure 4.3 (b)). Each of the sets will be submitted into the analysis, although for the visualization, data will be represented by three principal components.

After transforming original variables, the result was depicted using a 3-dimensional scatter plot where axes correspond to each of the components and the colour scale reflects the value of the target variable, total_UPDS score (Figure 4.4).

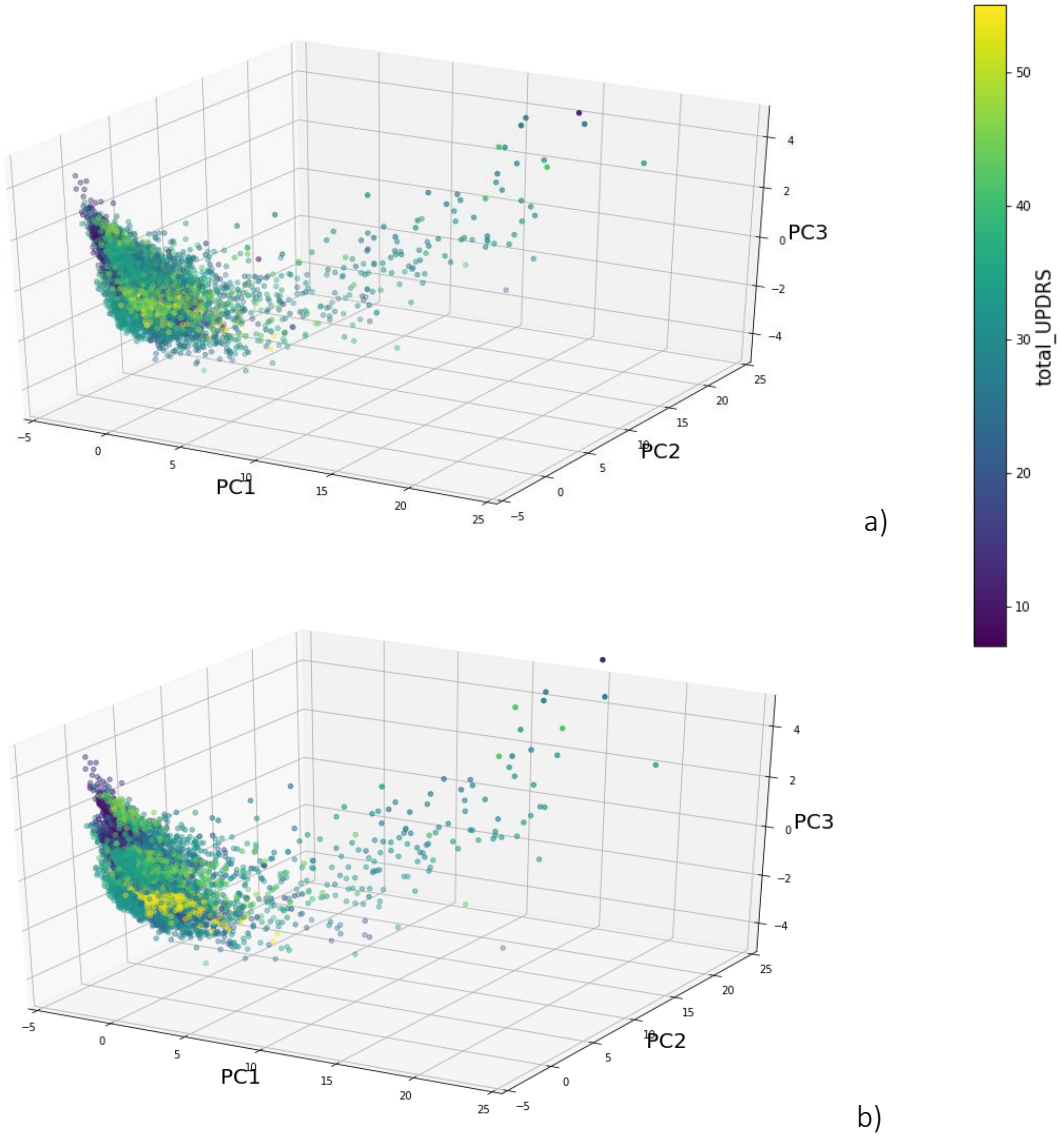


Figure 4.4 Scatter plot of the three-dimensional representation produced by taking the first three principal components obtained from different sets of input attributes with the color specification of total_UPDRS scores a) PCA based on 16 vocal attributes; b) PCA based on total 19 attributes

From the graph, values of the target variable are distributed throughout the sample area revealing more distinctive separation on Figure 4.4 (b). In view of the fact that the total_ and

motor_UPDRS values have strong linear relationships which is reflected by Figure Appendix D-1, hereinafter, the results are given only for one of these target variables. The scatter plots of 3 principal components with the target motor_UPDRS are presented on Figure Appendix D-2.

4.6 Extracting features with Autoencoder

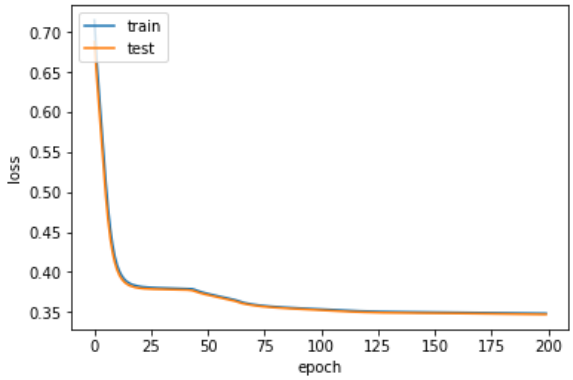
To project from a high-dimensional space to a low-dimensional, one must determine the number of features that best represent the sample in supervised learning. In other words, it is necessary to define the number of hidden units in the middle layer of the autoencoder's architecture. Unlike the literature on PCA in this respect, there is no such clear rule for determining a low-dimensional space when mapping the original data with an autoencoder. Some literature suggests rules of thumb for selecting the number of hidden units in a neural network (Blum, 1992; Swingler, 1996), some papers propose approaches with mathematical evidence as theoretical support (Xu & Chen, 2008). In this study, the number of hidden units was specified with reliance on the conclusion about the number of dimensions needed to capture 70-90% of the variance (Boger & Guterman, 1997) and to maintain equity in terms of the input dimension size when evaluating different feature extraction algorithms.

4.6.1 Training Vanilla autoencoder

Before training the autoencoder for reconstruction, data was normalized. The experiments were conducted on normalized and standardized data to compare behavior during training, as well as to apply various loss functions. That is, the use of binary cross-entropy requires that data values are scaled from 0 to 1.

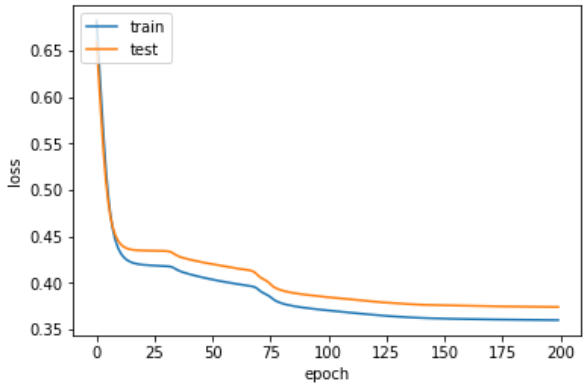
The optimization of hyperparameters is another important pre-phase in the training of neural networks. Several approaches for hyperparameter tuning exist (Bergstra et al., 2011; Bergstra & Bengio, 2012). The paper (Bengio, 2012) offers recommendations for debugging and overcoming difficulties of training multilayered neural networks by hyperparameter tuning. In this study, the manual hyperparameter search was applied by setting various combinations of parameters, and based on results, tweaking the parameters until finding a satisfactory set.

The results of two training processes are shown on Figure 4.5 and trainings with other hyperparameters are presented in Appendix E.



Activation function	sigmoid
Optimizer	Adam
Loss function	Binary cross entropy
Learning rate	0.001
Batch size	64
Epochs	200
Normalization method	MinMax

a)



Activation function	sigmoid
Optimizer	Adam
Loss function	Binary cross entropy
Learning rate	0.001
Batch size	64
Epochs	200
Normalization method	MinMax

b)

Figure 4.5 Training autoencoders of different architectures
a) 16-3-16 autoencoder; b) 19-3-19 autoencoder

From all graphs, including the ones in Appendix E, the loss function curve stabilizes after about 200 epochs, and the residuals do not decrease significantly. The learning rate for each training session has been set by default for each optimizer in Keras.

4.6.2 Visualizing features obtained by training Vanilla autoencoder

After training the autoencoder, three features were visualized with a 3-dimensional scatter plot using total_UPDS to map plot aspects to different colors. Figure 4.6 illustrates features extracted with autoencoders with hyperparameters as shown on Figure 4.5 (b). Alternative visualizations are presented in Appendix F.

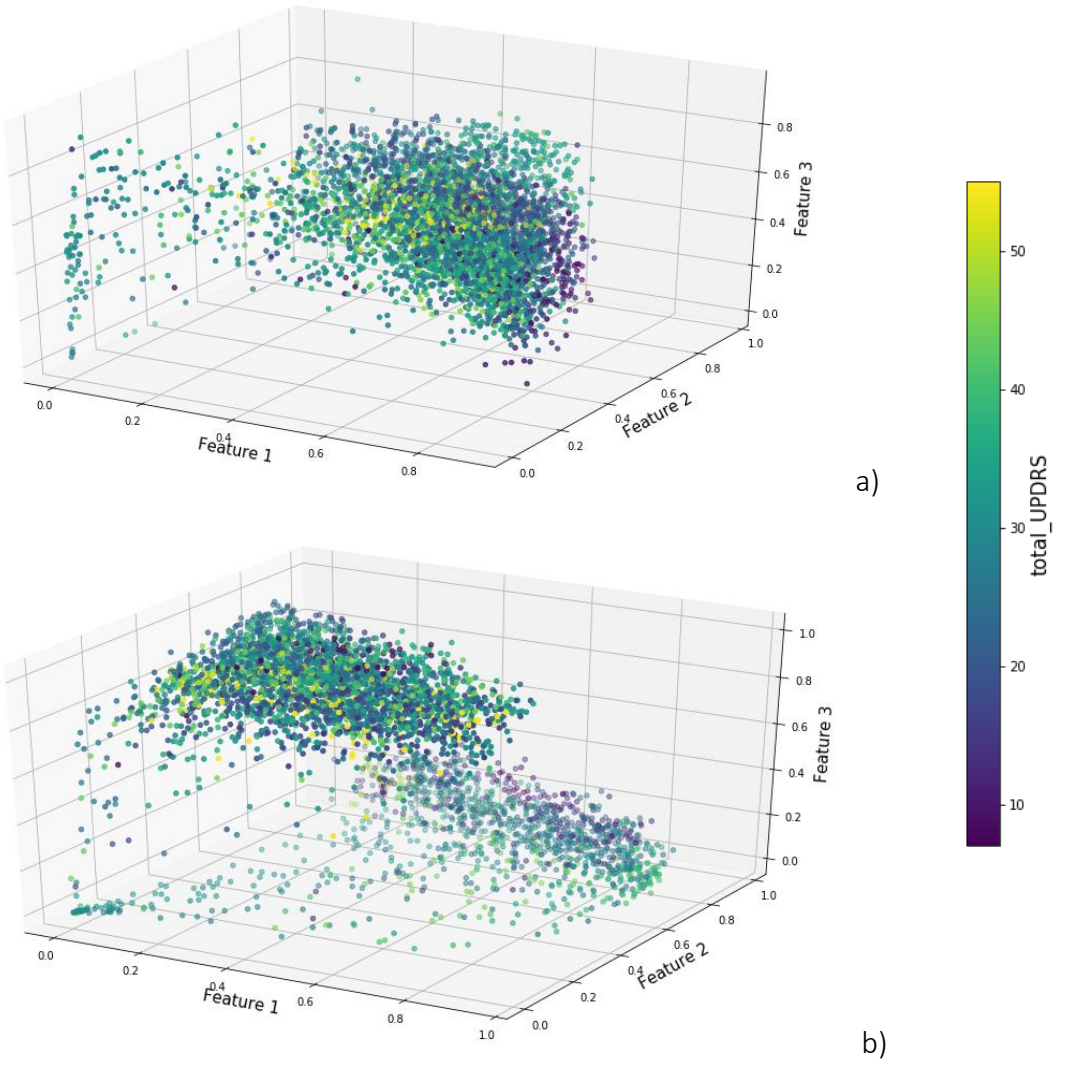


Figure 4.6 Scatter plots of three-dimensional codes produced by training autoencoders based on different sets of input attributes with the color specification of total_UPDRS scores
a) Features extracted out of 16 vocal attributes; b) Features extracted out of total 19 attributes

The visualizations on Figure 4.6 illustrate how initial set of attributes affects the representation through the features. The images in Appendix F show the effect with respect to the different parameterization of the autoencoder.

4.6.3 Training Stacked autoencoder and feature visualization

The stacked autoencoder was trained in a greedy layer-wise manner, each layer at a time, to preserve information. The decrease in the number of features in each successive layer occurred according to different heuristic architectures, and in some of them – with introducing regularization parameters. The components of the deep stacked autoencoder were

parametrized in the same way for each individual training procedure. The input data feeding into the autoencoder was normalized to [0, 1] range.

After training the stacked autoencoder, the code consisted of three features was visualized with a target color palette. Figure 4.7 illustrates the 19-15-10-5-3 fine-tuned autoencoder with no regularizers introduced.

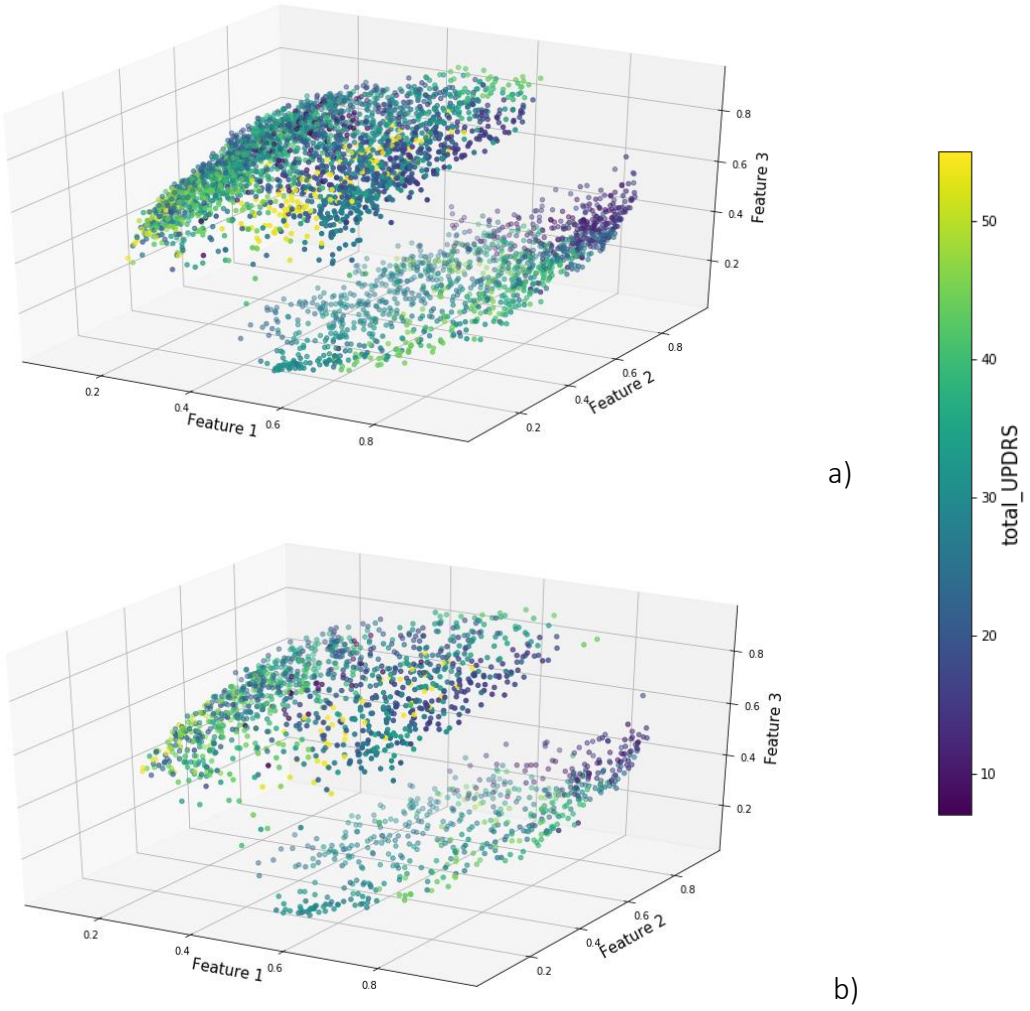


Figure 4.7 Scatter plots of three-dimensional codes produced by training the 19-15-10-5-3 stacked autoencoder with the color specification of total_UPDRS scores
a) Training data; b) Test data

The scatter plot on Figure 4.7 shows even larger separation between observations with similar values of the target variable than on Figure 4.6 (b). Observations with the highest UPDRS values are concentrated in the upper part although blended with others.

4.7 Evaluation method

Feature learning algorithms are usually evaluated in supervised applications by comparing the prediction results for a target variable. The choice of algorithms was based on the rule of thumb considering the specifics of the problem to take advantage of commonly used algorithms for solving regression problems. That is, the linear regression and support vector regression (SVR) were the first choice for the prediction of UPDRS scores. Through the visualization graphs from previous sections, the problem did not manifest clear signs of linearity, reason why non-linear supervised algorithms, single- and multi-layered neural networks were used in addition to the above mentioned.

To measure accuracy for a continuous target variable and understand whether a significant amount of the total variability is explained by the estimator, the relevant statistical metrics were used to assess different models: mean squared error, mean absolute error and R-squared.

4.7.1 Linear Regression

Multiple Linear Regression is a statistical method of fitting a model that describes the relationship between several explanatory variables and target variable which can be denoted as follows:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_k x_k + \varepsilon, \tag{4.1}$$

where $\boldsymbol{y}$ – predicted variable, $\boldsymbol{x_1, x_2, \ldots x_k}$ – explanatory variables, $\boldsymbol{\beta_0, \beta_1, \beta_2, \ldots \beta_k}$ – unknown parameters or regression coefficients, $\boldsymbol{\varepsilon}$ – residuals.

One of the approaches to estimating the parameter vector $\boldsymbol{\beta}$ to be found is to select the values of $\boldsymbol{\beta}$ that minimize the sum of squared errors (SSE). In regression analysis, the term of mean squared error (MSE) is sometimes used to measure the quality of an estimator. In a vector form it can be denoted as shown by Equation 4.2.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y - \hat{Y})^2, \tag{4.2}$$

where $\boldsymbol{n}$ – number of predictions, $\boldsymbol{Y}$ – real values of target variable, $\boldsymbol{\hat{Y}}$ – predicted values of the target variable, $\boldsymbol{(Y - \hat{Y})^2}$ – squares of errors.

Another criterion for assessing the goodness of a regression model is a coefficient of determination R-squared (R^2) which indicates the percentage of the variability in the dependent variable that independent variables explain collectively:

$$R^2 = \frac{SSR}{SST} \; or \; 1 - \frac{SSE}{SST}, \tag{4.3}$$

where $\boldsymbol{SSR}$ is a sum of squares explained by the regression model, $\boldsymbol{SSE}$ is a sum of squared errors, measure of variance unexplained by the regression model, $\boldsymbol{SST}$ is a sum of squares of the total variation ($SST = SSR + SSE$).

4.7.2 Support vector regression

Support vector regression (SVR) is a machine learning technique which uses the same principles as the support vector machine for classification. It estimates a continuous function by mapping the input data onto high-dimensional feature space and transforms the complex relationships into linear forms to make it possible to perform a linear separation. The SVR (Cortes & Vapnik, 1995) builds a hyperplane that separates each observation point in such a way that they fall within a distance with deviation from actual values of not more than a specified value. Different kernel types can be used in the SVR algorithm. In this study, the radial basis function kernel has been applied (Scikit-learn, 2017). Mathematically, a linear model in the feature space is denoted by the formula:

$$f(x_i) = \sum_{i=1}^{n} w_i \, g_i(x) + b, \tag{4.4}$$

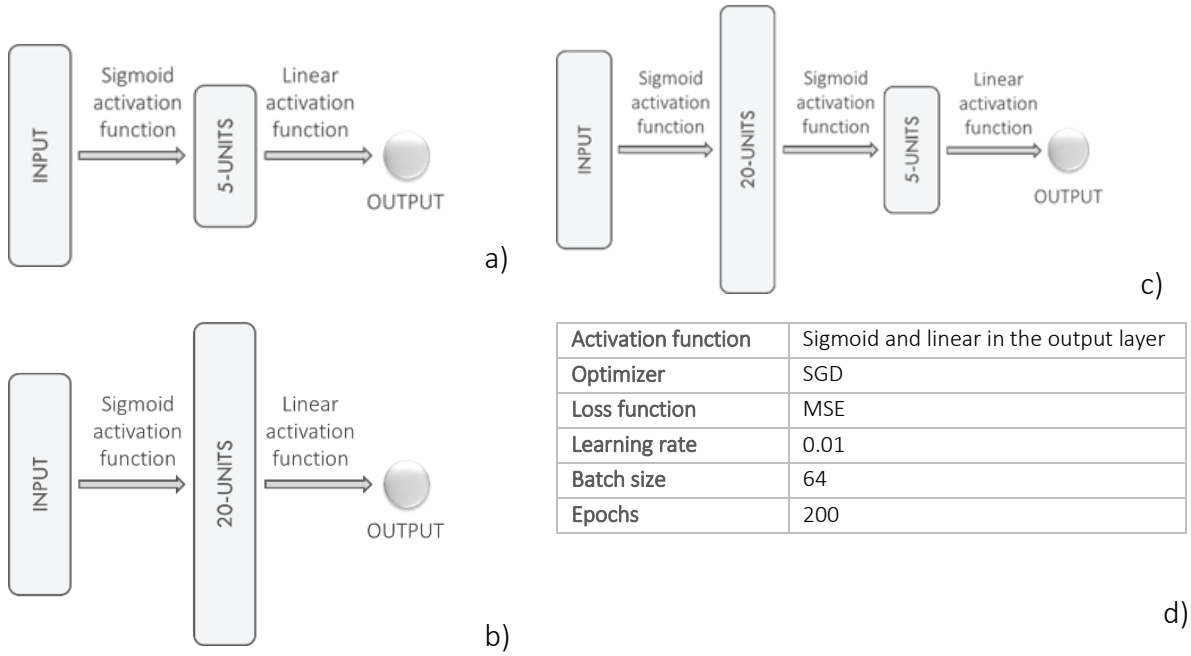
where $\boldsymbol{g_i(x)}$ denotes a set of non-linear transformations, $\boldsymbol{n}$ is a number of features, $\boldsymbol{b}$ is a bias term, $\boldsymbol{w_i}$ – weights or parameters to be minimized to guarantee the flatness which can be written as an optimization problem:

$$\begin{aligned} Minimize \quad & \frac{1}{2}||w||^2 \\ Subject\ to \quad & |y_i - f(x_i)| \leq \varepsilon \end{aligned} \tag{4.5}$$

One of the advantages of SVR, from its essence, is that it can be used to avoid constraints of using linear functions in a high dimensional feature space when the problem at hand is not explicitly linear.

4.7.3 Neural Network model

In this study, different architectures of neural networks were used to compare the performance of shallow and deeper structures in predicting the level of UPDRS. Figure 4.8 schematically presents these architectures and their hyperparameters. Each network has been trained for 200 epochs. Some experiments have been also carried out with an augmented number of epochs to trace the difference and significance of a drastic increase in this parameter.



Activation function	Sigmoid and linear in the output layer
Optimizer	SGD
Loss function	MSE
Learning rate	0.01
Batch size	64
Epochs	200

d)

Figure 4.8 Neural network architectures used in supervised learning
a)1 layer, 5-NN, b) 1 layer, 20-NN; c) 2 layers, 20-5-NN; d) hyperparameters

4.7.4 Validation method

The experiments have been conducted 30 times (10 – for neural network models due to the computation time), and the results are the product of averaged metrics obtained in all trials. The statistical median was used for this purpose being more robust measure compared to the mean. For the validation, the traditional training/test split was used in proportion of 70% to 30%, as well as k-fold cross validation with k equal to 10. The results were tabulated for each of the methods.

It is important to note that data providers initially determined the purpose of the dataset as predicting UPDRS estimates merely based on voice data. The analysis has been conducted

both, on voice data and with the addition of other attributes such as age, gender and testing time. The feature extraction has been also performed based on 16 voice attributes and based on total 19 attributes.

Chapter 5
Results and discussion

This chapter discusses the resulting performance of feature extraction methods compared to the raw data.

5.1 Raw data outcome

Table 5.1 presents the results obtained with the original patients' data.

Table 5.1 Model performance and computation time using patient data represented by original descriptors

Validation method	70/30				10-fold cross validation			
Model \ Metrics	MSE	MAE	R^2	Time	MSE	MAE	R^2	Time
Regression, 16 vars*	103.1281	8.3025	0.0926	00:00:00	103.7291	8.3432	0.0930	00:00:00
Regression, 19 vars	95.1707	8.0885	0.1747	00:00:00	94.6022	8.0526	0.1709	00:00:01
SVR, 16 vars	97.0687	7.5885	0.1453	00:00:01	95.9118	7.5340	0.1567	00:00:16
SVR, 19 vars	83.9886	6.8761	0.2704	00:00:01	80.7242	6.7094	0.2939	00:00:16
1 layer NN-5, 16 vars	91.2179	7.6610	0.2177	00:00:31	86.0042	7.5044	0.2292	00:05:21
1 layer NN-20, 16 vars	78.5532	7.0806	0.3102	00:00:31	75.6411	6.9079	0.3334	00:05:34
2 layers NN-20-5, 16 vars	77.6354	6.8407	0.3307	00:00:32	77.6823	6.7768	0.3111	00:05:39
1 layer NN-5, 19 vars	62.0525	6.1840	0.4497	00:00:32	59.8088	6.1177	0.4739	00:06:07
1 layer NN-20, 19 vars	45.0292	5.0804	0.6135	00:00:33	40.9976	4.7931	0.6419	00:06:14
2 layers NN-20-5, 19 vars	34.8285	4.1521	0.6989	00:00:33	27.4460	3.6707	0.7542	00:06:45
2 layers NN-20-5, 19 vars, 1000 epochs	23.2290	3.3326	0.8000	00:02:15	15.4043	2.6725	0.8681	00:54:41

* vars - *input variables*

The table shows that the simple regression yields poor results with a slight improvement in SVR. As the model becomes more complex, results are getting better – this is clearly seen when scrolling down from the first to the last row of the table. There is a markable difference in the performance of models fed with the different number of attributes. Regarding NN-models, the deeper architecture has yielded better results only when fed with 19 input variables, whereas for 16 variables the difference between shallow and deep networks is not significant.

The best performance has been demonstrated by the neural network model with two layers and augmented number of epochs. From Figure 5.1, the loss curve stabilizes after 200 epochs. Metric values are slightly improved. Of particular interest is the absolute error indicator of 2.67 which could not have been reached by applying other architectures. Nevertheless, the

computation time has notably increased, and after about 500 epochs, there is a slight divergence between training and test curves which requires closer examination for the subject of overfitting. For this reason, the model with an early stop after 200 epochs will be taken as a basis for comparing the performance of features in sub-section 5.5.

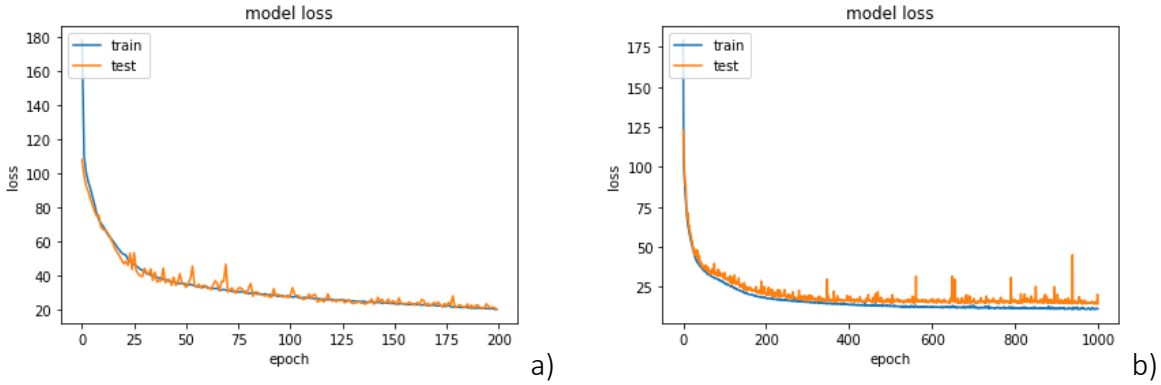


Figure 5.1 Training process of the 20-5-NN model with different number of epochs fed with raw data
a) Training for 200 epochs; b) Training for 1000 epochs

5.2 PCA outcome

Table 5.2 shows performing results obtained using the data pre-processed by principal component analysis.

Table 5.2 Model performance and computation time
using patient data represented by features obtained with PCA

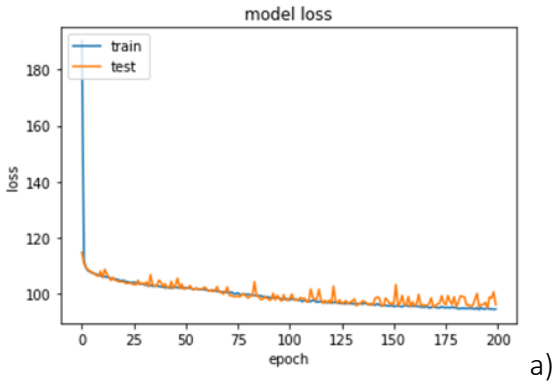
Validation method	70/30				10-fold cross validation			
Model \ Metrics	MSE	MAE	R^2	Time	MSE	MAE	R^2	Time
Regression, **3 PC**, 16 vars	113.8701	8.6314	0.0142	00:00:00	112.9515	8.6031	0.0122	00:00:00
Regression, **3 PC**, 19 vars	112.0777	8.5690	0.0306	00:00:00	110.7220	8.5265	0.0315	00:00:00
Regression, 4 **PC**, 19 vars	102.4713	8.4496	0.1078	00:00:00	101.6863	8.4130	0.1110	00:00:00
Regression, 7 **PC**, 19 vars	96.5330	8.0864	0.1506	00:00:00	98.2743	8.1093	0.1460	00:00:00
Regression, **3 PC + 3 vars**	102.5177	8.4392	0.1166	00:00:00	101.3507	8.4137	0.1121	00:00:00
SVR, **3 PC**, 16 vars	107.8608	8.0577	0.0651	00:00:01	107.1600	8.0571	0.0633	00:00:10
SVR, 3 **PC**, 19 vars	101.9601	7.9311	0.1208	00:00:01	100.9937	7.8506	0.1164	00:00:10
SVR, 4 **PC**, 19 vars	87.1336	7.2594	0.2186	00:00:01	87.8130	7.2679	0.2318	00:00:16
SVR, 7 **PC**, 19 vars	75.6056	6.2244	0.3210	00:00:01	71.9407	6.1179	0.3742	00:00:27
SVR, **3 PC + 3 vars**	76.8913	6.4452	0.3168	00:00:01	76.0686	6.3076	0.3322	00:00:13
1 layer NN-5, **3 PC**, 16 vars	106.1349	8.3239	0.0854	00:00:19	106.5305	8.2575	0.0583	00:02:32
1 layer NN-20, **3 PC**, 16 vars	102.7236	8.0496	0.0821	00:00:20	104.8482	8.1417	0.0663	00:02:38
1 layer NN-20, **3 PC + 3 vars**	60.3045	5.7811	0.4770	00:00:20	65.9381	5.9971	0.4097	00:02:42
2 layers NN-20-5, **3 PC**, 16 vars	104.8615	8.2587	0.0676	00:00:17	104.4934	8.1184	0.0701	00:02:53
2 layers NN-20-5, **3 PC + 3 vars**	39.0611	4.6024	0.6489	00:00:18	35.9732	4.3052	0.6912	00:02:48
1 layer NN-5, **3 PC**, 19 vars	103.8630	8.2057	0.1007	00:00:17	105.0233	8.0878	0.0620	00:02:39
1 layer NN-20, **3 PC**, 19 vars	101.3670	8.0856	0.1292	00:00:18	99.7643	7.9013	0.1253	00:02:43
2 layers NN-20-5, **3 PC**, 19 vars	97.5348	7.8192	0.1391	00:00:18	97.6352	7.7538	0.1283	00:02:49
1 layer NN-20, **4 PC**, 19 vars	87.7884	7.5345	0.2524	00:00:17	85.0505	7.4943	0.2438	00:02:55
2 layers NN-20-5, **4 PC**, 19 vars	83.9455	7.1652	0.2482	00:00:21	83.6697	7.3066	0.2672	00:03:13
1 layer NN-20, **7 PC**, 19 vars	52.1855	5.4298	0.5417	00:00:29	47.1196	5.1493	0.5906	00:17:19
2 layers NN-20-5, **7 PC**, 19 vars	33.6969	4.1099	0.6977	00:00:33	32.1828	3.9414	0.7205	00:04:43
2 layers NN-20-5, **3 PC**, 19 vars, 1000 epochs	100.6539	7.7817	0.1222	00:01:59	98.2104	7.7060	0.1183	00:14:55

The table indicates that the PCA transformation produces rather mediocre results in predicting UPDRS scores when using only 3 principal components for both 19 total and 16 voice variables submitted to the transformation. There are no observed improvements with increasing complexity of models.

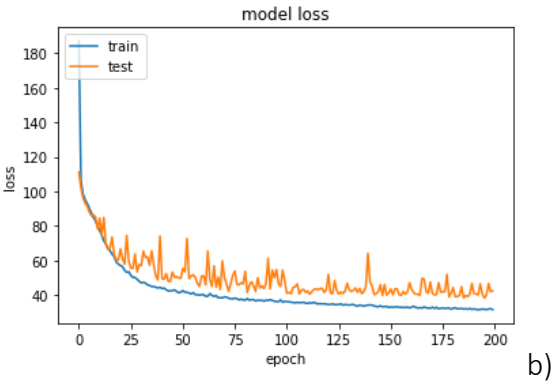
As previously stated, the number of components to be selected out of 19 transformed variables corresponds to at least 4 components or, according to an alternative selection approach, 7 principal components. A significant improvement is noted in the performance of 7 principal components and 3 principal components together with 3 original variables – age, gender and time. These two scenarios have yielded the best result for all metrics. The performance is noticeably improved with increasing the complexity of models. That is, the two-layer NN-20-5 model showed better results than one-layer NN-20 which, in turn, is better than SVR and regression.

These findings suggest that the efficiency of PCA transformation is only manifested when using a higher number of components. Three principal components are not enough to achieve satisfactory results.

The visualization on Figure 5.2 shows how different number of principal components affects the training process. The loss curve for 3 principal components and 3 variables (Figure 5.2 (c)) is much smoother and sloping downward gradually. The chart on Figure 5.2 (d) is an evidence of overfitting – there is a divergence between loss curves of train and test data after 200 epochs. Once again, an augmented number of training epochs requires a thorough testing, although sometimes produces good results.



a)



b)

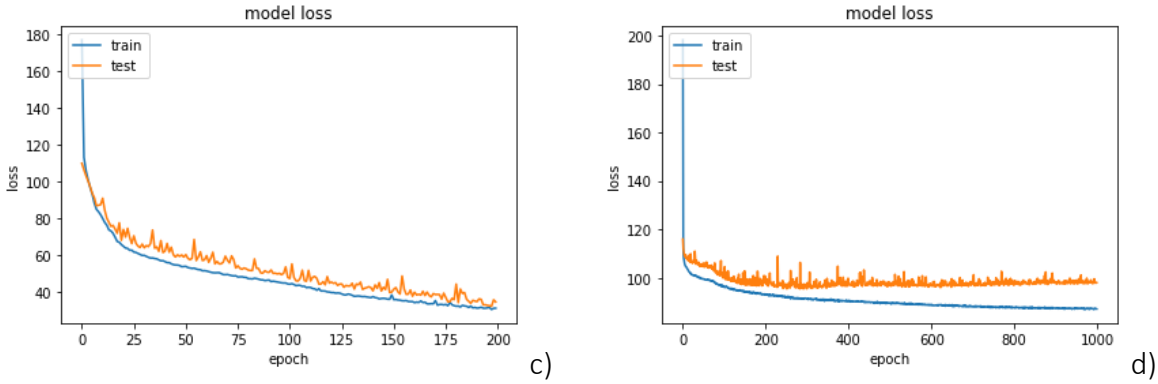


Figure 5.2 Training of 20-5-NN models fed with different number of principal components
a) 3 principal components; b) 7 principal components;
c) 3 principal components and 3 original attributes; d) 3 principle components, 1000 epochs

5.3 Autoencoder outcome

Table 5.3 presents the results obtained with data compressed by autoencoders.

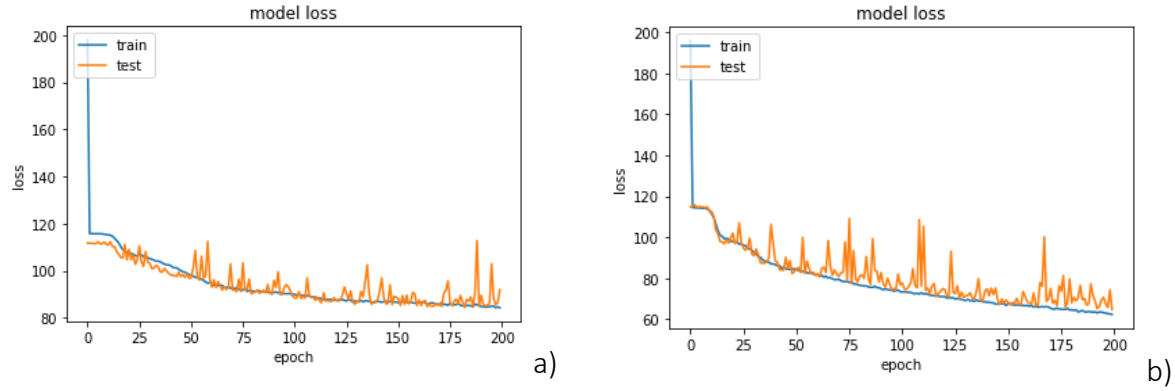
Table 5.3 Model performance and computation time
using patient data represented by features obtained with autoencoders of different architectures

Validation method	70/30				10-fold cross validation			
Model \ Metrics	MSE	MAE	R^2	Time	MSE	MAE	R^2	Time
Regression, **AE 16-3-16**	109.5664	8.4690	0.0461	00:00:22	107.5720	8.3983	0.0404	00:00:24
Regression, **AE 19-3-19**	109.3513	8.5071	0.0315	00:00:23	108.6341	8.4295	0.0346	00:00:23
Regression, **AE 19-4-19**	102.3643	8.2436	0.0980	00:00:25	103.3360	8.3292	0.0846	00:00:24
Regression, **AE 19-7-19**	95.7230	8.0360	0.1593	00:00:29	98.1209	8.0981	0.1214	00:00:36
Regression, **AE 16-3-16 + 3 vars**	98.8580	8.2080	0.1361	00:00:19	98.9415	8.1918	0.1192	00:00:24
SVR, **AE 16-3-16**	111.6937	8.3531	0.0261	00:00:25	107.7238	8.4207	0.0356	00:00:23
SVR, **AE 19-3-19**	110.6844	8.3530	0.0320	00:00:27	110.7880	8.4620	0.0126	00:00:24
SVR, **AE 19-4-19**	105.4576	8.1830	0.0550	00:00:29	104.8148	8.1011	0.0905	00:00:33
SVR, **AE 19-7-19**	100.2220	7.9961	0.1102	00:00:35	99.6690	8.0038	0.1240	00:00:47
SVR, **AE 16-3-16 + 3 vars**	100.2328	7.9447	0.1043	00:00:25	99.4853	8.1903	0.1172	00:00:23
1 layer NN-5, **AE 16-3-16**	106.2073	8.3724	0.0650	00:00:37	102.2896	8.2169	0.0972	00:03:20
1 layer NN-20, **AE 16-3-16**	107.0087	8.3317	0.0733	00:00:39	104.1707	8.1873	0.0881	00:03:31
1 layer NN-20, **AE 16-3-16 + 3 vars**	80.6003	7.2183	0.2971	00:00:39	72.9616	6.6904	0.3620	00:03:45
2 layers NN-20-5, **AE 16-3-16**	106.4400	8.4429	0.0517	00:00:40	102.7107	8.2197	0.1029	00:03:53
2 layers NN-20-5, **AE 16-3-16 + 3 vars**	67.4238	6.1677	0.4244	00:00:30	61.9327	6.0881	0.4686	00:03:47
1 layer NN-5, **AE 19-3-19**	109.0236	8.3981	0.0605	00:00:37	107.9416	8.3807	0.0544	00:04:18
1 layer NN-20, **AE 19-3-19**	105.8200	8.3031	0.0793	00:00:40	108.7877	8.5047	0.0599	00:04:25
2 layers NN-20-5, **AE 19-3-19**	108.7698	8.4841	0.0499	00:00:42	106.5099	8.3938	0.0567	00:04:31
1 layer NN-20, **AE 19-4-19**	93.2880	7.8086	0.1661	00:00:37	91.4001	7.6261	0.2175	00:05:37
2 layers NN-20-5, **AE 19-4-19**	92.3169	7.6355	0.2056	00:00:42	89.6171	7.5230	0.2219	00:05:54
1 layer NN-20, **AE 19-7-19**	71.9801	6.6938	0.3690	00:00:47	71.9425	6.6393	0.3730	00:06:00
2 layers NN-20-5, **AE 19-7-19**	67.9972	6.3231	0.4150	00:00:48	60.5495	6.0022	0.4747	00:06:05
2 layers NN-20-5, **AE 19-3-19**, 1000 epochs	104.0211	8.2995	0.0966	00:01:34	104.2340	8.2072	0.0877	00:23:08

From the table, three features obtained by training the autoencoder and introduced to models have not demonstrated an outstanding performance in predicting UPDRS scores. The results are very similar to obtained by PCA. For the three-dimensional code, number of input variables submitted for the feature extraction doesn't affect the results. Better performance is observed when models are fed with three features together with three original variables or with more features. That is, the 2-layer NN-20-5 model fed with 7 features encoded by the autoencoder

has yielded the best results. Nevertheless, they are not as good as obtained with 7 principal components. Looking through the metrics PCA markedly outperformed the autoencoder: MSE - 32.1828 (PCA) against 60.5495 (AE); MAE - 3.9414 (PCA) against 6.0022 (AE); R-squared - 0.7205 (PCA) against 0.4747 (AE). In addition, the average computation time for the models fed with a product of training the autoencoder is 1,7 higher than needed to handle the principal components.

Figure 5.3 provides some graphical visualizations of training models fed with different inputs. The visualization on Figure 5.3 (a) supports the results from Table 5.3 – training of the model fed with 3-dimensional input shows lack of convergence, the loss curve remains almost flat meaning that weight adjustment does not lead to a loss reduction. The loss curve on the graphs 5.3 (b) and 5.3 (c) indicates that the model copes with data in a more efficient way gradually reducing the loss. After about 100 iterations, the loss curve on Figure 5.3 (c) reveals more frequent spikes. This can be a sign of too high learning rate from this point and requires some adjustments in further iterations with the use of call-back functions to be applied at this stage of the training procedure. The batch size, a series of observations updated at each iteration, also seriously affects the smoothness of the loss curve. In this study, it was heuristically found that the batch size of 64 is optimal, both in terms of results and computation time. That is, Figure 5.3 (d) illustrates the training process with the batch size equal to 1 revealing frequent spikes and lack of convergence. The computation time in this scenario has increased by more than 4 times.



a) b)

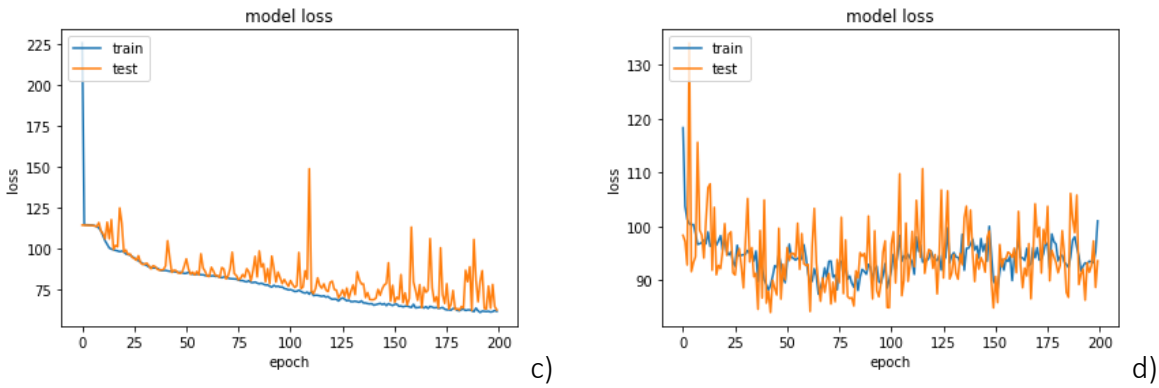


Figure 5.3 Training of the 20-5-NN model fed with different number of features obtained by training the autoencoder
a) 3 features; b) 7 features;
c) 3 features and 3 original attributes; d) 3 features + 3 original variables, training batch size = 1

5.4 Stacked Autoencoder outcome

Table 5.4 summarizes the results from pre-trained stacked autoencoders.

Table 5.4 Model performance and computation time using patient data represented by features obtained with stacked autoencoders

Validation method	70/30				10-fold cross validation			
Metrics / Architecture	MSE	MAE	R^2	Time	MSE	MAE	R^2	Time
Stacked AE 19-16-13-10-7, Regression	100.9402	8.2132	0.1077	00:01:24	98.2915	8.1990	0.1334	00:20:49
Stacked AE 19-16-13-10-7, SVR	112.6470	8.4659	0.0066	00:01:29	114.6821	8.5216	0.0100	00:34:25
Stacked AE 19-10-3, NN-20	76.2842	6.8817	0.3278	00:02:21	71.2500	6.5631	0.3794	00:38:59
Stacked **AE 19-15-10-5-3**, NN-20	71.6864	6.6122	0.3752	00:03:10	70.1393	6.5387	0.3955	00:43:50
Stacked AE 19-17-15-13-11-9-7-5-3, NN-20	72.7667	6.6586	0.3508	00:09:53	71.7385	6.5253	0.3823	02:09:56
Stacked AE 19-16-13-10-7-4, NN-5	71.8061	6.5574	0.3832	00:02:23	67.9075	6.3579	0.4200	00:51:43
Stacked AE 19-16-13-10-7, NN-20	63.7522	6.2119	0.4474	00:02:47	62.3679	6.1090	0.4568	00:47:43
Stacked **AE 19,15,11,7**, NN 20-5	62.5761	5.9115	0.4591	00:02:31	51.2863	5.5175	0.5606	00:45:21
Stacked AE 19-Drp-30-Drp-20-Drp-10-Drp-3 , NN-20	73.5631	6.5984	0.3624	00:03:07	70.8404	6.5470	0.3776	03:18:01
Stacked AE 19-Drp-30-Drp-20-Drp-7 , NN-20	66.9256	6.4302	0.4131	00:03:00	64.1785	6.2311	0.4380	00:33:54
Stacked **AE 19-Drp-30-Drp-20-Drp-7** , NN-20-5	64.1745	6.0761	0.4348	00:03:25	53.9133	5.6011	0.5251	00:35:42
Stacked **AE 19-15-11-7** , NN-20-5, 1000 epochs	41.3529	4.5276	0.6274	00:05:05	37.9246	4.2279	0.6764	01:14:00

** Drp - dropout*

The table shows that the input encoded by the stacked autoencoder 19-15-11-7 and supplied to the NN-20-5 model showed the best results among models trained for 200 epochs (MSE = 51.2863, MAE = 5.5175, R-squared = 0.5606). Training each layer at a time has managed to preserve more information than a single compression from 19 to 7 dimensions. That is, from the Table 5.3, the vanilla autoencoder 19-7-19 has been only able to produce results reflected by metrics: MSE=60.5495 (18.06% ↓[1]), MAE=6.0022 (8.78% ↓), R-squared=0.4747 (8.59% ↓), which is on average 11.81% worse than the stacked autoencoder. The deepest architecture of

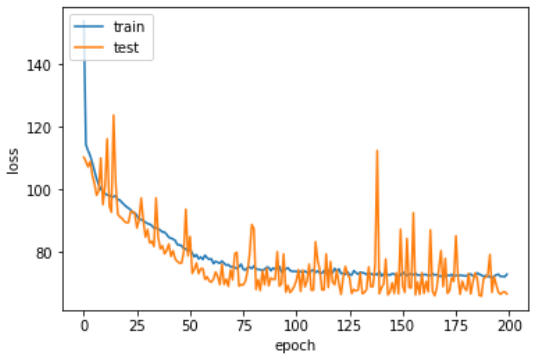
[1] The symbol "↓" here means deterioration and not decrease.

eight hidden layers has yielded slightly better outcome compared to the architecture with the least hidden layers but it took about four times as long to compute.

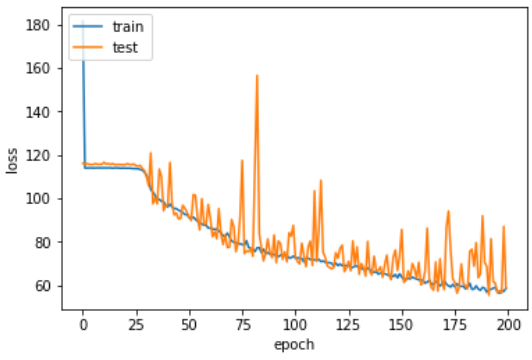
The most interesting are the results of the architecture with the code size equal to three dimensions. PCA and vanilla autoencoder failed to produce satisfactory outcomes with three-dimensional input – more features have been required to feed models for better results. In the meantime, the stacked autoencoder showed some progress in compressing data into a three-dimensional representation. The NN-20-5 model fed with 3 features produced by PCA, vanilla 19-3-19, and stacked autoencoder 19-15-10-5-3 has yielded the following results, respectively: MSE = 97.6352 (39.30%↓), 106.5099 (51.85%↓), 70.1393; MAE = 7.7538 (18.58%↓), 8.3938 (28.37% ↓), 6.5387; R-squared = 0.1283 (26.72%↓), 0.0567 (33.88%↓), 0.3955. In average, the stacked autoencoder outperformed PCA by 23.52% and vanilla autoencoder by 30.04% when the model has a three-dimensional input.

The model with an augmented number of training epochs and seven-dimensional input has shown the best result in terms of metrics. At the same time, the increase in epochs requires additional assessment both in terms of computational time and possible overfitting although the chart on Figure 5.4 (c) shows no signs of such.

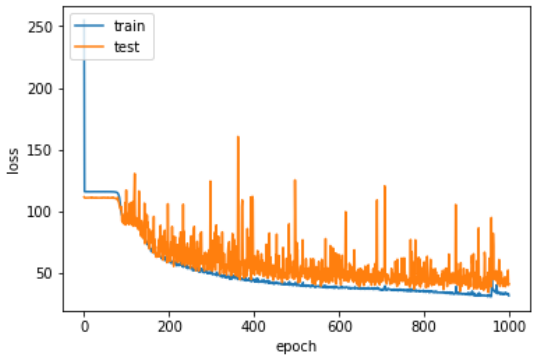
The introduction of the dropout regularizer equal to 20% in each layer of the stacked autoencoder has not drastically shaken the outcome. Nonetheless, the training process of the regularized autoencoder illustrated on Figure 5.4 (d) reveals a smoother loss curve.



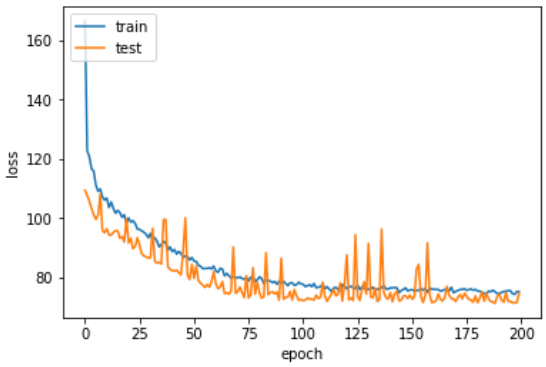
a)



b)



c)



d)

Figure 5.4 Training stacked autoencoders of different architectures
a) Stacked autoencoder 19-15-10-5-3; b) Stacked autoencoder 19-15-11-7
c) Stacked autoencoder 19-15-11-7, 1000 epochs;
d) 19-30-20-10-3 stacked autoencoder with dropout regularizer

5.5 Comparative analysis of results from different inputs

In this sub-section, a perspective of results based on the best performance in each input category will be given. Table 5.5 shows the performance of best models fed with the raw data and data transformed by feature extraction algorithms. It is worth noting that in all scenarios listed in Table 5.5, the NN-20-5 model was considered as an estimator for predicting UPDRS scores, and 19 original variables were processed by feature extraction algorithms. The results are based on the 10-fold validation method.

Table 5.5 Total UPDRS scores prediction results using patient data represented by 19 original descriptors and pre-processed by principal component analysis, vanilla and stacked autoencoders

Input layer of NN 20-5 model	MSE	MAE	R^2	Time
19-Dimensional input				
Raw data, 19 attributes	27.4460	3.6707	0.7542	00:06:45
Raw data, 19 attributes, model trained for 1000 epochs	15.4043	2.6725	0.8681	00:54:41
7-Dimensional input				
PCA, 7 features	32.1828	3.9414	0.7205	00:04:43
Autoencoder 19-7-19	60.5495	6.0022	0.4747	00:06:05
Stacked AE 19-15-11-7	51.2863	5.5175	0.5606	00:45:21
Stacked AE 19-15-11-7, model trained for 1000 epochs	37.9246	4.2279	0.6764	01:14:00
3-Dimensional input				
PCA, 3 features	97.6352	7.7538	0.1283	00:02:49
Autoencoder 19-3-19	106.5099	8.3938	0.0567	00:04:31
Stacked AE, 19-15-10-5-3	70.1393	6.5387	0.3955	00:43:50
6-Dimensional input (Transformed and original variables)				
PCA, 3 PC + 3 vars	35.9732	4.3052	0.6912	00:02:48
AE 16-3-16 + 3 vars	61.9327	6.0881	0.4686	00:03:47

The original attributes have reported better performance metrics than transformed attributes. Nevertheless, when comparing the performance of compression algorithms, the deep stacked

autoencoder showed the best output for the tree-dimensional input – the mean absolute error has been reduced from 7.75 (PCA) and 8.22 (Vanilla AE) to 6.5 (Stacked AE) – by 15.67% and 22.10%, respectively; the mean squared error – from 97.63 (PCA) and 106.51 (Vanilla AE) to 70.14 (Stacked AE) – by 28.16% and 34.15%, respectively; the coefficient of determination – improved from 12.83% (PCA) and 5.67% (Vanilla autoencoder) to 39.55% (Stacked autoencoder) – by 26.72% and 33.88%, respectively.

In the scenario where Parkinson's disease patients were represented by seven transformed attributes, the performance of algorithms in predicting UPDRS scores was improved, especially for PCA and vanilla AE. That is, 7 principal components provided better results by 58.48% than 3 components; the 7-dimensional representation encoded by the vanilla autoencoder – 37.81% better than 3-dimensional; the stacked AE 19-15-11-7 – 19.67% better than stacked AE 19-15-10-5-3. The model fed with the product of stacked AE 19-15-11-7 and trained for 1000 epochs showed significantly improved results outperforming by 36.45% 3-dimensional stacked AE.

Slightly improved results were also achieved when models were fed with 3 original variables and 3 transformed features extracted out of vocal variables, thereby reducing the total number of input variables from 19 to 6.

Chapter 6
Conclusions and future work

In this thesis, the effectiveness of an autoencoder as a feature extraction algorithm has been examined in predicting scores according to the unified Parkinson's disease rating scale. By forcing an autoencoder to learn a compressed representation of patients it has been trained to find patterns in vocal and other patients' data. The technique has been compared with other feature reduction methods by feeding generated features to different models in a supervised manner to test their estimating accuracy and track how different scenarios affected the performance. Some experiments on parametrization for the training optimization have been done, and outcomes of algorithms with different hyperparameters recorded.

It can be concluded that among all the compression methods, the deep stacked autoencoder trained in a greedy manner showed the best result for the three-dimensional representation. By using the stacked autoencoder instead of reducing the dimensionality in one step, richer features were obtained, and more information preserved. Even though inputs of a larger dimension have shown better performance and 19 original attributes outperformed all feature reduction algorithms, the potential and utility of the deep stacked autoencoder would become even more expressive when dealing with a larger number of original input attributes feeding to the model. In addition, the visualization of three-dimensional data transformed by feature learning algorithms provides an excellent picture of what is happening when data is subjected to compression and how efficient the separation of observations by a target variable is.

As a result of conducted experiments with inputs of different dimensions, interesting insights have been drawn. Only patient vocal data is not enough for accurate estimation of UPDRS scores. Voice attributes together with age, gender and testing time provide a more accurate estimate of UPDRS both for original and transformed input features. When it comes to decision-making about the code size of an autoencoder, the rules commonly used in PCA can also be applied to autoencoders.

For the parametrization of neural networks, different hyperparameter combinations as well as different neural network architectures have been tested. From training the autoencoder and

NN-models, it became evident that early stopping sometimes is required to avoid overfitting while the loss function still has large derivatives. This condition makes the training process different from the optimization in general where convergence is considered when the gradient becomes very small. The introduction of the dropout regularizer in each of the autoencoder layers was aimed at increasing the generalization performance of models. It was also noted that different batch sizes remarkably affect the training process. The size of 64 observations was considered optimal in terms of results and computational time. There is still a margin for experiments with other hyperparameters and optimization techniques to be applied in future works.

There is a need for a specialist from the medical side to assess the results and guide research towards possible improvements in terms of model effectiveness for specific tasks and clinical application. That is, some studies (Martínez-Martín et al., 2015) have provided a disease severity scale by determining cut-off points for Parkinson's disease severity levels between mid, moderate and severe stages based on movement disorder UPDRS scores, thereby discretizing continuous target variable. So, the regression problem can be transformed into a classification with the subsequent application of classification machine learning algorithms to discover whether it is possible to benefit from the discretization in predicting the level of disease severity when patients are represented by features obtained with the deep autoencoder. In general, learning higher-level descriptors for solving medical problems may have a benefit for the presentation of patients that have atypical clinical status for a specific medical institution but can be represented using features derived from other hospital data where their conditions might be common.

Bibliography

Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science*, *169*(1), 147–169. https://doi.org/10.1016/S0364-0213(85)80012-4

Alain, G., & Bengio, Y. (2014). What Regularized Auto-Encoders Learn from the Data Generating Distribution. *The Journal of Machine Learning Research*, *15*(1), 3563–3593. Retrieved from https://arxiv.org/abs/1211.4246

Alin, A. (2010). Multicollinearity. *Wiley Interdisciplinary Reviews: Computational Statistics*, *2*(3), 370–374. https://doi.org/10.1002/wics.84

Asgari, M., & Shafran, I. (2010). Predicting severity of Parkinson's disease from speech. In *Proceedings of the 2010 Annual International Conference of the IEEE Engineering in Medicine and Biology Society* (pp. 5201–5204). Buenos Aires, Argentina: IEEE. https://doi.org/10.1109/IEMBS.2010.5626104

Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(12), 2481–2495. https://doi.org/10.1109/TPAMI.2016.2644615

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. Retrieved from http://arxiv.org/abs/1409.0473

Baldi, P. (2012). Autoencoders , Unsupervised Learning , and Deep Architectures. In *UTLW'11 Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning workshop* (Vol. 27, pp. 37–50). Retrieved from http://proceedings.mlr.press/v27/baldi12a/baldi12a.pdf

Baldi, P., & Hornik, K. (1989). Neural Networks and Principal Component Analysis : Learning from Examples Without Local Minima. *Neural Networks*, *2*(1), 53–58. https://doi.org/10.1016/0893-6080(89)90014-2

Beam, A. L. (2017). Deep Learning 101 - Part 1: History and Background. Retrieved from http://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

Bengio, Y. (2009). Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning*, *2*(1), 1–127. https://doi.org/10.1561/2200000006

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade* (2nd ed., pp. 437–478). Springer Berlin Heidelberg. Retrieved from https://arxiv.org/abs/1206.5533

Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A Neural Probabilistic Language Model. *The Journal of Machine Learning Research*, *3*, 1137–1155. Retrieved from http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy Layer-Wise Training of

Deep Networks. In *NIPS'06 Proceedings of the 19th International Conference on Neural Information Processing Systems* (pp. 153–160). Vancouver, Canada. Retrieved from https://papers.nips.cc/paper/3048-greedy-layer-wise-training-of-deep-networks

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for Hyper-Parameter Optimization. In *NIPS'11 Proceedings of the 24th International Conference on Neural Information Processing Systems* (pp. 2546–2554). Granada, Spain. Retrieved from https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf

Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, *13*(1), 281–305. Retrieved from http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Blum, A. (1992). *Neural networks in C++ : An object-oriented framework for building connectionist systems* (1st ed.). New York, USA: John Wiley & Sons, Inc. Retrieved from https://books.google.pt/books/about/Neural_Networks_in_C++.html?id=9H0pAQAAMAAJ&redir_esc=y

Boger, Z., & Guterman, H. (1997). Knowledge extraction from artificial neural network models. In *1997 IEEE International Conference On Systems, Man, And Cybernetics* (Vol. 4, pp. 3030–3035). https://doi.org/10.1109/ICSMC.1997.633051

Bourlard, H., & Kamp, Y. (1988). Auto-Association by Multilayer Perceptrons and Singular Value Decomposition. *Biological Cybernetics*, *59*(4–5), 291–294. https://doi.org/10.1007/BF00332918

Bryant, M. S., Rintala, D. H., Hou, J.-G., & Protas, E. J. (2015). Relationship of Falls and Fear of Falling to Activity Limitations and Physical Inactivity in Parkinson's Disease. *Journal of Aging and Physical Activity*, *23*(2), 187–193. https://doi.org/10.1123/japa.2013-0244

Caliskan, A., Badem, H., Başturk, A., & Yüksel, M. E. (2017). Diagnosis of the Parkinson disease by using deep neural network classifier. *Istanbul University - Journal of Electrical and Electronics Engineering*, *17*(2), 3311–3318. Retrieved from http://dergipark.gov.tr/download/article-file/326952

Challa, K. N. R., Pagolu, V. S., Panda, G., & Majhi, B. (2016). An improved approach for prediction of Parkinson's disease using machine learning techniques. In *Proceedings of the 2016 International conference on Signal Processing, Communication, Power and Embedded System (SCOPES)* (pp. 1446–1451). https://doi.org/10.1109/SCOPES.2016.7955679

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *EMNLP'2014 Proceedings of the 19th Conference on Empirical Methods in Natural Language Processing* (pp. 1724–1734). Doha, Qatar. https://doi.org/10.3115/v1/D14-1179

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2015). Gated Feedback Recurrent Neural Networks. In *ICML'15 Proceedings of the 32nd International Conference on International Conference on Machine Learning* (p. Vol 37 2067-2075). Lille, France. Retrieved from https://arxiv.org/abs/1502.02367

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297.

https://doi.org/10.1023/A:1022627411411

Deng, L., Way, O. M., Yu, D., & Way, O. M. (2014). Deep Learning : Methods and Applications. *Foundations and Trends in Signal Processing*, *7*(3–4), 197–387. https://doi.org/10.1561/2000000039

Doersch, C. (2016). Tutorial on Variational Autoencoders. Pittsburgh, PA, USA: Carnegie Mellon University. Retrieved from https://arxiv.org/abs/1606.05908

Du, B., Xiong, W., Wu, J., Zhang, L., Zhang, L., & Tao, D. (2017). Stacked Convolutional Denoising Auto-Encoders for Feature Representation. *Proceedings of the IEEE Transactions on Cybernetics*, *47*(4), 1017–1027. https://doi.org/10.1109/TCYB.2016.2536638

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179–211. https://doi.org/10.1016/0364-0213(90)90002-E

Erhan, D., Courville, A., & Vincent, P. (2010). Why Does Unsupervised Pre-training Help Deep Learning? *Journal of Machine Learning Research*, *11*(Feb), 625–660. https://doi.org/10.1145/1756006.1756025

Eskidere, Ö., Ertaş, F., & Hanilçi, C. (2012). A comparison of regression methods for remote tracking of Parkinson's disease progression. *Expert Systems with Applications*, *39*(5), 5523–5528. https://doi.org/10.1016/j.eswa.2011.11.067

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, *36*(4), 193–202. https://doi.org/10.1007/BF00344251

Gallinari, P., Lecun, Y., Thiria, S., & Fogelman Soulie, F. (1987). Distributed associative memories: A comparison. In *Proceedings of COGNITIVA 87*. Paris, La Villette, France. Retrieved from http://yann.lecun.com/exdb/publis/pdf/gallinari-87.pdf

Gibrat, C., Saint-Pierre, M., Bousquet, M., Lévesque, D., Rouillard, C., & Cicchetti, F. (2009). Differences between subacute and chronic MPTP mice models: Investigation of dopaminergic neuronal degeneration and α-synuclein inclusions. *Journal of Neurochemistry*, *109*(5), 1469–1482. https://doi.org/10.1111/j.1471-4159.2009.06072.x

Gibson, A., & Patterson, J. (2017). *Deep Learning. A Practitioner's Approach* (1st ed.). Sebastopol,CA, USA: O'Reilly Media. Retrieved from https://books.google.pt/books/about/Deep_Learning.html?id=rLcuDwAAQBAJ&redir_esc=y

Glorot, X., Bengio, Y., & Bordes, A. (2011). Deep Sparse Rectifier Neural Networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics* (Vol. 15, pp. 315–323). Retrieved from http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

Goetz, C. G., Poewe, W., Rascol, O., Sampaio, C., & Stebbins, G. T. (2003). The Unified Parkinson's Disease Rating Scale (UPDRS): Status and recommendations. *Movement Disorders*, *18*(7), 738–750. https://doi.org/10.1002/mds.10473

Gonzalez, R. C., Woods, R. E., Scott, A. H., & Hall, P. P. (2007). *Digital Image Processing* (3rd ed.). Upper Saddle River, NJ: Pearson. Retrieved from

http://web.ipac.caltech.edu/staff/fmasci/home/astro_refs/Digital_Image_Processing_3rdEd_truncated.pdf

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. Cambridge, MA, USA: The MIT Press. Retrieved from https://www.deeplearningbook.org/

Halko, N., Martinsson, P. G., & Tropp, J. A. (2011). Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Review*, *53*(2), 217–288. https://doi.org/10.1137/090771806

Haykin, S. (2008). *Neural Networks and Learning Machines* (3rd ed.). Hamilton, Ontario, Canada: Pearson. Retrieved from http://dai.fmph.uniba.sk/courses/NN/haykin.neural-networks.3ed.2009.pdf

Heaton, J. (2015). *Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks* (Edition 1.). Scotts Valley, CA, USA: CreateSpace Independent Publishing Platform. Retrieved from https://books.google.pt/books/about/Artificial_Intelligence_for_Humans.html?id=q9mijgEACAAJ&redir_esc=y

Hinton, G. E. (2002). Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, *14*(8), 1771–1800. https://doi.org/10.1162/089976602760128018

Hinton, G. E. (2012a). Boltzmann machine learning. Lecture from the course Neural Networks for Machine Learning. Coursera, University of Toronto. Retrieved from https://www.coursera.org/learn/neural-networks/lecture/iitiK/boltzmann-machine-learning-12-min

Hinton, G. E. (2012b). Restricted Boltzmann Machines. Lecture from the course Neural Networks for Machine Learning. Coursera, University of Toronto. Retrieved from https://www.coursera.org/learn/neural-networks/lecture/TIqjI/restricted-boltzmann-machines-11-min

Hinton, G. E. (2012c). Shallow autoencoders for pre-training. Lecture from the course Neural Networks for Machine Learning. Coursera, University of Toronto. Retrieved from https://www.coursera.org/learn/neural-networks/lecture/cxXuG/shallow-autoencoders-for-pre-training-7-mins

Hinton, G. E., & Osindero, S. (2006). A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, *18*(7), 1527–1554. https://doi.org/10.1162/neco.2006.18.7.1527

Hinton, G. E., & Salakhutdinov, R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, *313*(5786), 504–508. https://doi.org/10.1126/science.1127647

Hinton, G. E., Srivastava, N., & Swersky, K. (2012). The softmax output function. Lecture from the course Neural Networks for Machine Learning. Coursera, University of Toronto. Retrieved from https://www.coursera.org/lecture/neural-networks/another-diversion-the-softmax-output-function-7-min-68Koq

Hinton, G. E., & Zemel, R. S. (1994). Autoencoders , Minimum Description Length and Helmholtz Free Energy. In *NIPS'93 Proceedings of the 6th International Conference on Neural Information Processing Systems* (pp. 3–10). Retrieved from https://papers.nips.cc/paper/798-autoencoders-minimum-description-length-and-

helmholtz-free-energy.pdf

Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: The difficulty of learning long-term dependencies. In *Field Guide to Dynamical Recurrent Networks* (1st ed., pp. 464–479). New York: Wiley-IEEE Press. https://doi.org/10.1109/9780470544037.ch14

Hochreiter, S., & Urgen Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, *79*(8), 2554–8. https://doi.org/10.1073/pnas.79.8.2554

Jain, V., & Seung, H. (2008). Natural Image Denoising with Convolutional Networks. In *NIPS'08 Proceedings of the 21st International Conference on Neural Information Processing Systems* (pp. 769–776). Retrieved from https://papers.nips.cc/paper/3506-natural-image-denoising-with-convolutional-networks.pdf

Jolliffe, I. T., & Cadima, J. (2016). Principal component analysis : a review and recent developments. *Philosophical Transactions. Series A. Mathematical, Physical, and Engineering Sciences*, *374*(2065), 1–16. https://doi.org/10.1098/rsta.2015.0202

Jordan, M. I. (1986). *Serial order: A parallel distributed processing approach*. *Technical report*. San Diego, CA. Retrieved from http://cseweb.ucsd.edu/~gary/258/jordan-tr.pdf

Kaiser, H. F. (1961). A note on Guttman's lower bound for the number of common factors. *British Journal of Statistical Psychology*, (14), 1–2. https://doi.org/10.1111/j.2044-8317.1961.tb00061.x

Kalchbrenner, N., & Blunsom, P. (2013). Recurrent Continuous Translation Models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (pp. 1700–1709). Seattle, WA. Retrieved from https://www.aclweb.org/anthology/D13-1176

Karpathy, A. (2015). Convolutional Neural Networks for Visual Recognition. Lecture notes CS231n. Stanford University. Stanford, CA. Retrieved from http://cs231n.github.io/neural-networks-1/

Kingma, D. P., & Welling, M. (2013). Auto-Encoding Variational Bayes. Amsterdam, The Netherlands: University of Amsterdam. Retrieved from https://arxiv.org/abs/1312.6114

Krizhevsky, A., Hinton, G. E., & Sutskever, I. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems* (Vol. 1, pp. 1097–1105). Retrieved from https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Larochelle, H. (2013). Neural networks class [6.2] : Autoencoder - loss function. Hugo Larochelle. Retrieved from https://www.youtube.com/watch?v=xTU79Zs4XKY

Lecun, Y. (1987). *Modeles connexionnistes de l'apprentissage (connectionist learning models)*. PhD Dissertation, Universite P. et M. Curie.

LeCun, Y., Boser, B. E., Denker, J., Henderson, D., Howard, R. E., Hubbard, W. E., & Jackel, L. (1989). Back-propagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541–551. https://doi.org/10.1162/neco.1989.1.4.541

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE* (Vol. 86, pp. 2278–2324). https://doi.org/10.1109/5.726791

Lee, H., & Ng, A. Y. (2008). Sparse deep belief net model for visual area V2. In *NIPS'07 Proceedings of the 20th International Conference on Neural Information Processing Systems* (pp. 873--880). Retrieved from https://papers.nips.cc/paper/3313-sparse-deep-belief-net-model-for-visual-area-v2

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., … Sánchez, C. I. (2017). A Survey on Deep Learning in Medical Image Analysis. *Medical Image Analysis*, *42*, 60–88. https://doi.org/10.1016/j.media.2017.07.005

Mairal, J., Bach, F., Ponce, J., & Sapiro, G. (2009). Online dictionary learning for sparse coding. In *ICML '09 Proceedings of the 26th Annual International Conference on Machine Learning* (pp. 689–696). Montreal, Quebec. https://doi.org/10.1145/1553374.1553463

Makhzani, A., & Frey, B. (2014). k-Sparse Autoencoders. Toronto, Ontario, Canada: University of Toronto. Retrieved from https://arxiv.org/pdf/1312.5663.pdf

Martínez-Martín, P., Rodríguez-Blázquez, C., Mario Alvarez, Arakaki, T., Arillo, V. C., Chaná, P., … Merello, M. (2015). Parkinson's disease severity levels and MDS-Unified Parkinson's Disease Rating Scale. *Parkinsonism & Related Disorders*, *21*(1), 50–54. https://doi.org/10.1016/j.parkreldis.2014.10.026

Masci, J., Meier, U., Ciresan, D., & Schmidhuber, J. (2011). Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction. In *ICANN 2011 Proceedings of the 21st International Conference on Artificial Neural Networks* (Vol. 6791, pp. 52–59). Espoo, Finland. https://doi.org/10.1007/978-3-642-21735-7_7

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*(4), 115–133. https://doi.org/10.1007/BF02478259

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. In *NIPS'13 Proceedings of the 26th International Conference on Neural Information Processing Systems* (pp. 3111–19). Lake Tahoe, Nevada. Retrieved from https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

Minsky, M. L., & Papert, S. A. (1969). *Perceptrons*. Oxford, England: M.I.T. Press.

Miotto, R., Li, L., Kidd, B. A., & Dudley, J. T. (2016). Deep Patient : An Unsupervised Representation to Predict the Future of Patients from the Electronic Health Records. *Scientific Reports*, *6*(26094), 1–10. https://doi.org/10.1038/srep26094

Nair, V., & Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. In *ICML'10 Proceedings of the 27th International Conference on International*

Conference on Machine Learning (pp. 807–814). Haifa, Israel. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.165.6419&rep=rep1&type=pdf

Ng, A. (2011). *Sparse autoencoder. Lecture notes CS294A*. Stanford University. Stanford, CA. Retrieved from https://web.stanford.edu/class/cs294a/sparseAutoencoder.pdf

Ng, A., Katanforoosh, K., & Mourri, Y. B. (2017a). Padding. Lecture from the course Convolutional Neural Networks. Coursera, deeplearning.ai. Retrieved from https://es.coursera.org/learn/convolutional-neural-networks/lecture/o7CWi/padding

Ng, A., Katanforoosh, K., & Mourri, Y. B. (2017b). Why do you need non-linear activation functions? Lecture from the course Neural Networks and Deep Learning. Coursera, deeplearning.ai. Retrieved from https://www.coursera.org/lecture/neural-networks-deep-learning/why-do-you-need-non-linear-activation-functions-OASKH

Nilashi, M., Ibrahim, O., & Ahani, A. (2016). Accuracy Improvement for Predicting Parkinson's Disease Progression. *Scientific Reports*, *6*(34181), 1–18. https://doi.org/10.1038/srep34181

Olah, C. (2015). Understanding LSTM Networks. Retrieved October 3, 2018, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Olshausen, B. a, & Field, D. J. (1997). Sparse coding with an incomplete basis set: a strategy employed by V1? *Vision Research*, *37*(23), 3311–3325. https://doi.org/10.1016/S0042-6989(97)00169-7

Parker, D. B. (1985). *Parker DB 1985 Learning logic Technical Report TR-47*. Cambridge, MA.

Piccinini, G. (2004). The first computational theory of mind and brain: a close look at Mcculloch and Pitts's "Logical calculus of ideas immanent in nervous activity." *Synthese*, *141*(2), 175–215. https://doi.org/10.1023/B:SYNT.0000043018.52445.3e

Poole, B., Sohl-dickstein, J., & Ganguli, S. (2014). Analyzing noise in autoencoders and deep networks. Stanford University. Stanford, CA. Retrieved from https://arxiv.org/abs/1406.1831

Portilla, J., & Mosconi, F. (2017). Zero to Deep Learning™ with Python and Keras. Udemy, Data Weekends. Retrieved from https://www.udemy.com/zero-to-deep-learning/learn/v4/overview

Postuma, R. B., & Montplaisir, J. (2009). Predicting Parkinson's disease – why, when, and how? *Parkinsonism & Related Disorders*, *15*(3), S105–S109. https://doi.org/10.1016/S1353-8020(09)70793-X

Preacher, K. J., & MacCallum, R. C. (2003). Repairing Tom Swift's Electric Factor Analysis Machine. *Understanding Statistics*, *2*(1), 13–43. https://doi.org/10.1207/S15328031US0201_02

Ranzato, M. A. (2007). Sparse Feature Learning for Deep Belief Networks. In *NIPS'07 Proceedings of the 20th International Conference on Neural Information Processing Systems* (pp. 1185–1192). Vancouver, British Columbia, Canada. Retrieved from https://papers.nips.cc/paper/3363-sparse-feature-learning-for-deep-belief-networks.pdf

Ranzato, M. A., Poultney, C., Chopra, S., Cun, Y. L., Ranzato, M., Poultney, C., … Cun, Y. L. (2006). Efficient Learning of Sparse Representations with an Energy-Based Model. In *NIPS'06 Proceedings of the 19th International Conference on Neural Information Processing Systems* (pp. 1137–1144). Cambridge, MA: MIT Press. Retrieved from https://papers.nips.cc/paper/3112-efficient-learning-of-sparse-representations-with-an-energy-based-model

Raschka, S. (2015). *Python Machine Learning Unlock* (2nd ed.). Birmingham, UK.: Packt Publishing. Retrieved from http://books.tarsoit.com/Python Machine Learning.pdf

Ravi, D., Wong, C., Deligianni, F., Berthelot, M., Andreu-Perez, J., Lo, B., & Yang, G. Z. (2017). Deep Learning for Health Informatics. *IEEE Journal of Biomedical and Health Informatics*, *21*(1), 4–21. https://doi.org/10.1109/JBHI.2016.2636665

Rezende, D. J., Mohamed, S., & Wierstra, D. (2014). Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *ICML'14 Proceedings of the 31st International Conference on International Conference on Machine Learning* (pp. 1278–1286). Beijing, China. Retrieved from http://arxiv.org/abs/1401.4082

Rifai, S., Muller, X., Vincent, P., & Bengio, Y. (2011). Contractive Auto-Encoders : Explicit Invariance During Feature Extraction. In *ICML'11 Proceedings of the 28th International Conference on International Conference on Machine Learning* (pp. 833–840). Bellevue, Washington, USA. Retrieved from http://www.icml-2011.org/papers/455_icmlpaper.pdf

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 19–27. https://doi.org/10.1037/h0042519

Rouzbahani, H. K., & Daliri, M. R. (2011). Diagnosis of Parkinson's disease in human using voice signals. *Basic and Clinical Neuroscience*, *2*(3), 12–20. Retrieved from https://pdfs.semanticscholar.org/b4f8/e3a43692b1e0a79e0909eb939b8c55b1f7da.pdf

Ruder, S. (2016). An overview of gradient descent optimization algorithms. Dublin, Ireland: Insight Centre for Data Analytics, NUI Galway Aylien Ltd. Retrieved from https://arxiv.org/abs/1609.04747

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533–536. https://doi.org/10.1038/323533a0

Salakhutdinov, R. (2015). Learning Deep Generative Models. *Annual Review of Statistics and Its Application*, *2*(1), 361–385. https://doi.org/10.1146/annurev-statistics-010814-020120

Salakhutdinov, R., & Hinton, G. (2009). Deep Boltzmann Machines. In *AISTATS 2009 Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics* (Vol. 5, pp. 448–455). Retrieved from http://proceedings.mlr.press/v5/salakhutdinov09a/salakhutdinov09a.pdf

Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673–2681. https://doi.org/10.1109/78.650093

Scikit-learn. (2017). Support vector regression — scikit-learn 0.19.1 documentation.

Retrieved May 30, 2018, from http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

Seung, H. S. (1998). Learning Continuous Attractors in Recurrent Networks. In *NIPS '97 Proceedings of the 1997 conference on Advances in neural information processing systems* (Vol. 10, pp. 654–660). Denver, Colorado, USA. Retrieved from https://papers.nips.cc/paper/1369-learning-continuous-attractors-in-recurrent-networks

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In *Parallel distributed processing: explorations in the microstructure of cognition* (pp. 194–281). Cambridge, MA, USA: MIT Press. Retrieved from https://www.researchgate.net/publication/239571798_Information_processing_in_dynamical_systems_Foundations_of_harmony_theory

Socher, R., Huang, E., Pennington, J., Ng, A. Y., & Manning, C. D. (2011). Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection. In *NIPS'11 Proceedings of the 24th International Conference on Neural Information Processing Systems* (pp. 801–809). Granada, Spain. Retrieved from https://papers.nips.cc/paper/4204-dynamic-pooling-and-unfolding-recursive-autoencoders-for-paraphrase-detection.pdf

Sokolov, E., Zimovnov, A., Panin, A., Lobacheva, E., & Kazeev, N. (2017). Gradient descent extensions. Lecture from Introduction to Deep Learning course. Coursera, National Research University Higher School of Economics. Retrieved from https://www.coursera.org/lecture/intro-to-deep-learning/gradient-descent-extensions-lYGBt

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*(1), 1929–1958. Retrieved from http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. In *NIPS'14 Proceedings of the 27th International Conference on Neural Information Processing Systems* (pp. 3104–3112). Montreal, Canada. Retrieved from https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf

Swingler, K. (1996). *Applying neural networks : a practical guide*. London, UK: Morgan Kaufmann.

Tipping, M. E., & Bishop, C. M. (1999). Probabilistic principal components analysis. *Journal of the Royal Statistical Society Series B (Statistical Methodology)*, *61*(3), 611–622. https://doi.org/10.1111/1467-9868.00196

Tsanas, A., Little, M. A., McSharry, P. E., & Ramig, L. O. (2010). Accurate Telemonitoring of Parkinson's Disease Progression by Noninvasive Speech Tests. *IEEE Transactions on Biomedical Engineering*, *57*(4), 884–893. https://doi.org/10.1109/TBME.2009.2036000

Vincent, P. (2011). A Connection Between Score Matching and Denoising Autoencoders. *Neural Computation*, *23*(7), 1661–1674. https://doi.org/10.1162/NECO_a_00142

Vincent, P., & Larochelle, H. (2008). Extracting and Composing Robust Features with

Denoising Autoencoders. In *ICML '08 Proceedings of the 25th international conference on Machine learning* (pp. 1096--1103). Helsinki, Finland. https://doi.org/10.1145/1390156.1390294

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P.-A. (2010). Stacked Denoising Autoencoders : Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research*, *11*, 3371–3408. Retrieved from https://dl.acm.org/citation.cfm?id=1756006.1953039

Vyas, S., & Kumaranayake, L. (2006). Constructing socio-economic status indices: how to use principal components analysis. *Health Policy and Planning*, *21*(6), 459–468. https://doi.org/10.1093/heapol/czl029

Wang, H., & Raj, B. (2017). On the Origin of Deep Learning. Pittsburgh, PA, USA: Carnegie Mellon University. Retrieved from https://arxiv.org/pdf/1702.07800.pdf

Werbos, P. (1974). *Beyond regression : New tools for predicting and analysis in the behavioral sciences*. PhD Dissertation, Harvard Uiversity. Retrieved from https://www.researchgate.net/publication/279233597_Beyond_Regression_New_Tools_for_Prediction_and_Analysis_in_the_Behavioral_Science_Thesis_Ph_D_Appl_Math_Harvard_University

Widrow, B., & Hoff, M. (1960). Adaptive switching circuits. *IRE WESCON Convention Record*, 96–104. Retrieved from http://www.dtic.mil/dtic/tr/fulltext/u2/241531.pdf

Widrow, B., & Lehr, M. A. (1990). 30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation. *Proceedings of the IEEE*, *78*(9), 1415–1442. https://doi.org/10.1109/5.58323

Xu, S., & Chen, L. (2008). A novel approach for determining the optimal number of hidden layer neurons for FNN's and its application in data mining. In *ICITA 2008 Proceedings of the 5th International Conference on Information Technology and Applications* (pp. 683–686). Cairns, Queensland, Australia. Retrieved from https://eprints.utas.edu.au/6995/

Zafar, M., Zhu, D., Li, X., Yang, K., & Levy, P. (2017). SAFS : A Deep Feature Selection Approach for Precision Medicine. Retrieved from https://arxiv.org/abs/1704.05960

Appendix A

Table Appendix A-1 Descriptive statistics of Parkinson's patient dataset

Variable	Description	Type	Statistics			
			Min	Max	Mean	SD
subject#	Integer that uniquely identifies each patient	ID				
age	Patient's age	int	36	85	65	8.8215
sex	Patient's gender	bin	0	1	9	
test time	Time since recruitment into the trial. The integer part is the number of days since recruitment.	int	-4.2600	215.49	92.86372	53.4456
Jitter(%)	Several measures of variation in fundamental frequency	int	0.000830	0.099990	0.006154	0.005624
Jitter(Abs)		int	0.000002	0.000446	0.000044	0.000036
Jitter:RAP,		int	0.000330	0.05754	0.002987	0.003124
Jitter:PPQ5		int	0.000430	0.069560	0.003277	0.003732
Jitter:DDP		int	0.000980	0.172630	0.008962	0.009371
Shimmer	Several measures of variation in amplitude	int	0.003060	0.268630	0.034035	0.025835
Shimmer(dB)		int	0.026000	2.107000	0.310960	0.230254
Shimmer:APQ3		int	0.001610	0.162670	0.017156	0.0132
Shimmer:APQ5		int	0.001940	0.167020	0.020144	0.016664
Shimmer:APQ11		int	0.002490	0.275460	0.027481	0.019986
Shimmer:DDA		int	0.004840	0.488020	0.051467	0.039711
NHR	Measures of ratio of noise to tonal components in the voice	int	0.000286	0.748260	0.059692	0.032120
HNR		int	1.659000	37.875000	21.679495	4.291096
RPDE	A nonlinear dynamical complexity measure	int	0.151020	0.966080	0.541473	0.100986
DFA	Signal fractal scaling exponent	int	0.514040	0.865600	0.653240	0.070902
PPE	A nonlinear measure of fundamental frequency variation	int	0.021983	0.731730	0.219589	0.091498
motor_UPDRS	Clinician's motor UPDRS score, linearly interpolated	int	7.000000	54.992000	29.018942	10.700283
total_UPDRS	Clinician's total UPDRS score, linearly interpolated	int	5.037700	39.511000	21.296229	8.129282

Appendix B

Table Appendix B-1 Correlation coefficients between variables in Parkinson's patient dataset

	age	sex	test_time	Jitter(%)	Jitter(Abs)	Jitter:RAP	Jitter:PPQ5	Jitter:DDP	Shimmer	Shimmer(dB)	Shimmer:APQ3	Shimmer:APQ5	Shimmer:APQ11	Shimmer:DDA	NHR	HNR	RPDE	DFA	PPE	total_UPDRS
age	1.000000	-0.041602	0.019884	0.023071	0.035685	0.010255	0.013199	0.010258	0.101554	0.111130	0.098912	0.089983	0.135238	0.098913	0.007093	-0.104842	0.090208	-0.092870	0.120790	0.310290
sex	-0.041602	1.000000	-0.009805	0.051422	-0.154661	0.076718	0.087995	0.076703	0.058736	0.056481	0.044937	0.064819	0.023360	0.044938	0.168170	-0.000167	-0.159262	-0.165113	-0.099901	-0.096559
test_time	0.019884	-0.009805	1.000000	-0.022837	-0.011349	-0.028888	-0.023290	-0.028876	-0.033870	-0.030962	-0.029020	-0.036504	-0.039110	-0.029017	0.026357	0.036545	-0.038887	0.019261	-0.000563	0.075263
Jitter(%)	0.023071	0.051422	-0.022837	1.000000	0.865574	0.984181	0.968214	0.984184	0.709791	0.716704	0.664149	0.694002	0.645965	0.664147	0.825294	-0.675188	0.427128	0.226550	0.721849	0.074247
Jitter(Abs)	0.035685	-0.154661	-0.011349	0.865574	1.000000	0.844622	0.790534	0.844626	0.649041	0.655866	0.623825	0.621397	0.589992	0.623823	0.699954	-0.706420	0.547097	0.352264	0.787848	0.066926
Jitter:RAP	0.010255	0.076718	-0.028888	0.984181	0.844622	1.000000	0.947196	1.000000	0.681729	0.685551	0.650226	0.659831	0.603082	0.650225	0.792373	-0.641473	0.382891	0.214881	0.670652	0.064015
Jitter:PPQ5	0.013199	0.087995	-0.023290	0.968214	0.790534	0.947196	1.000000	0.947203	0.732747	0.734591	0.676711	0.734021	0.668413	0.676710	0.864864	-0.662409	0.381503	0.175359	0.663491	0.063352
Jitter:DDP	0.010258	0.076703	-0.028876	0.984184	0.844626	1.000000	0.947203	1.000000	0.681734	0.685556	0.650228	0.659833	0.603090	0.650227	0.792377	-0.641482	0.382886	0.214893	0.670660	0.064027
Shimmer	0.101554	0.058736	-0.033870	0.709791	0.649041	0.681729	0.732747	0.681734	1.000000	0.992334	0.979828	0.984904	0.935457	0.979827	0.795158	-0.801416	0.468235	0.132540	0.615709	0.092141
Shimmer(dB)	0.111130	0.056481	-0.030962	0.716704	0.655866	0.685551	0.734591	0.685556	0.992334	1.000000	0.968015	0.976373	0.936338	0.968014	0.798077	-0.802496	0.472409	0.126111	0.635163	0.098790
Shimmer:APQ3	0.098912	0.044937	-0.029020	0.664149	0.623825	0.650226	0.676711	0.650228	0.979828	0.968015	1.000000	0.962723	0.885695	1.000000	0.732736	-0.780697	0.436878	0.130735	0.576704	0.079363
Shimmer:APQ5	0.089983	0.064819	-0.036504	0.694002	0.621397	0.659831	0.734021	0.659833	0.984904	0.976373	0.962723	1.000000	0.938935	0.962723	0.798173	-0.790638	0.450890	0.128038	0.593677	0.083467
Shimmer:APQ11	0.135238	0.023360	-0.039110	0.645965	0.589992	0.603082	0.668413	0.603090	0.935457	0.936338	0.885695	0.938935	1.000000	0.885694	0.711546	-0.777974	0.480739	0.179648	0.623416	0.120838
Shimmer:DDA	0.098913	0.044938	-0.029017	0.664147	0.623823	0.650225	0.676710	0.650227	0.979827	0.968014	1.000000	0.962723	0.885694	1.000000	0.732734	-0.780696	0.436872	0.130736	0.576702	0.079363
NHR	0.007093	0.168170	-0.026357	0.825294	0.699954	0.792373	0.864864	0.792377	0.795158	0.798077	0.732736	0.798173	0.711546	0.732734	1.000000	-0.684412	0.416660	-0.022088	0.564654	0.060952
HNR	-0.104842	-0.000167	0.036545	-0.675188	-0.706420	-0.641473	-0.662409	-0.641482	-0.801416	-0.802496	-0.780697	-0.790638	-0.777974	-0.780696	-0.684412	1.000000	-0.659053	-0.290519	-0.758722	-0.162117
RPDE	0.090208	-0.159262	-0.038887	0.427128	0.547097	0.382891	0.381503	0.382886	0.468235	0.472409	0.436878	0.450890	0.480739	0.436872	0.416660	-0.659053	1.000000	0.192030	0.566065	0.156897
DFA	-0.092870	-0.165113	0.019261	0.226550	0.352264	0.214881	0.175359	0.214893	0.132540	0.126111	0.130735	0.128038	0.179648	0.130736	-0.022088	-0.290519	0.192030	1.000000	0.394650	-0.113475
PPE	0.120790	-0.099901	-0.000563	0.721849	0.787848	0.670652	0.663491	0.670660	0.615709	0.635163	0.576704	0.593677	0.623416	0.576702	0.564654	-0.758722	0.566065	0.394650	1.000000	0.156195
total_UPDRS	0.310290	-0.096559	0.075263	0.074247	0.066926	0.064015	0.063352	0.064027	0.092141	0.098790	0.079363	0.083467	0.120838	0.079363	0.060952	-0.162117	0.156897	-0.113475	0.156195	1.000000

Table Appendix B-2 Top-10 the most correlated variables

1st variable	2nd variable	Correlation Coefficient
Shimmer: APQ3	Shimmer:DDA	1.000000
Jitter:RAP	Jitter DDP	1.000000
Shimmer	Shimmer (dB)	0.992334
Shimmer	Shimmer:APQ5	0.984904
Jitter(%)	Jitter:DDP	0.984184
Jitter(%)	Jitter:RAP	0.984181
Shimmer	Shimmer:APQ3	0.979828
Shimmer	Shimmer:DDA	0.979827
Shimmer(dB)	Shimmer: APQ5	0.976373
Jitter (%)	Jitter:PPQ5	0.968214

Appendix C

Table Appendix C-1 Proportion of the variance explained by principal components obtained via transformation of 16 vocal attributes

PC	Eigenvalue	Difference	Proportion	Cumulative
1	11.26296090	9.58977485	0.7038	0.7038
2	1.67318605	0.43246464	0.1046	0.8084
3	1.24072141	0.47572392	0.0775	0.8859
4	0.76499749	0.45569438	0.0478	0.9337
5	0.30930312	0.08638185	0.0193	0.9530
6	0.22292127	0.05012698	0.0139	0.9670
7	0.17279430	0.01102358	0.0108	0.9778
8	0.16177072	0.05867585	0.0101	0.9879
9	0.10309487	0.05956273	0.0064	0.9943
10	0.04353213	0.02324121	0.0027	0.9970
11	0.02029092	0.00627372	0.0013	0.9983
12	0.01401720	0.00534915	0.0009	0.9992
13	0.00866805	0.00420307	0.0005	0.9997
14	0.00446498	0.00446461	0.0003	1.0000
15	0.00000038	0.00000036	0.0000	1.0000
16	0.00000002		0.0000	1.0000

Table Appendix C-2 Proportion of the variance explained by principal components obtained via transformation of 19 original attributes

PC	Eigenvalue	Difference	Proportion	Cumulative
1	11.27166550	9.54593448	0.5931	0.5931
2	1.72573102	0.25384181	0.0908	0.6840
3	1.47188921	0.45273415	0.0775	0.7614
4	1.01915506	0.02835759	0.0536	0.8150
5	0.99079747	0.19456970	0.0521	0.8672
6	0.79622777	0.08022044	0.0419	0.9091
7	0.71600733	0.41912340	0.0377	0.9468
8	0.29688394	0.09013182	0.0156	0.9624
9	0.20675212	0.03797892	0.0109	0.9733
10	0.16877320	0.01650824	0.0089	0.9821
11	0.15226497	0.05333870	0.0080	0.9902
12	0.09892627	0.05800530	0.0052	0.9954
13	0.04092096	0.02069104	0.0022	0.9975
14	0.02022992	0.00633885	0.0011	0.9986
15	0.01389107	0.00522913	0.0007	0.9993
16	0.00866194	0.00420548	0.0005	0.9998
17	0.00445645	0.00445608	0.0002	1.0000
18	0.00000038	0.00000036	0.0000	1.0000
19	0.00000002		0.0000	1.0000

Appendix D

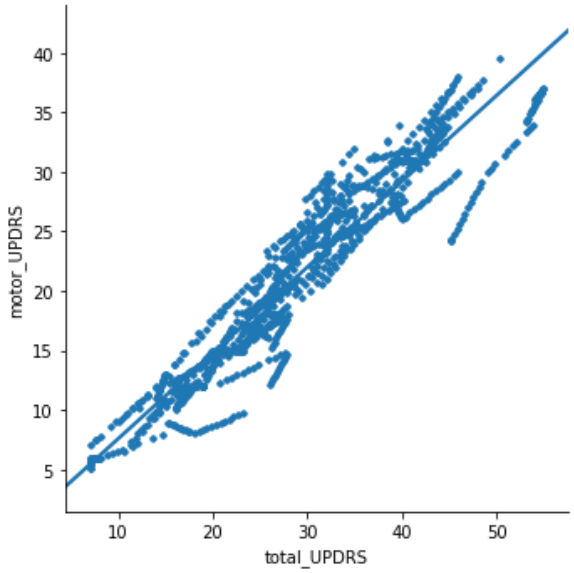


Figure Appendix D-1 Scatter plot of total_UPDRS vs motor_UPDRS

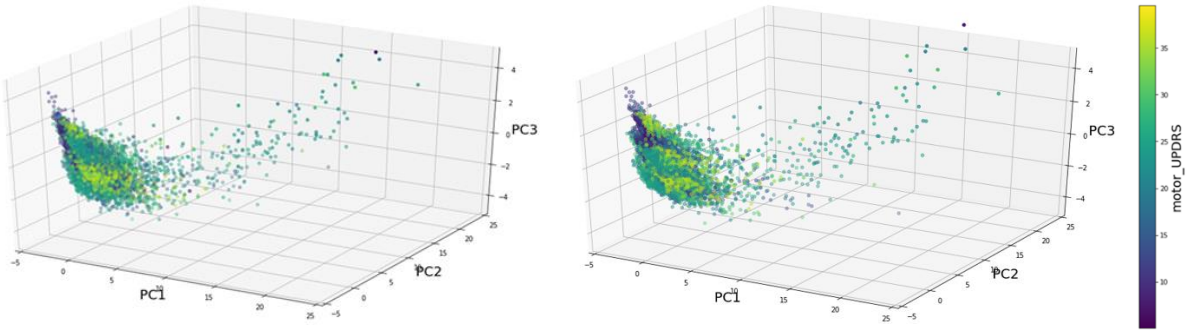
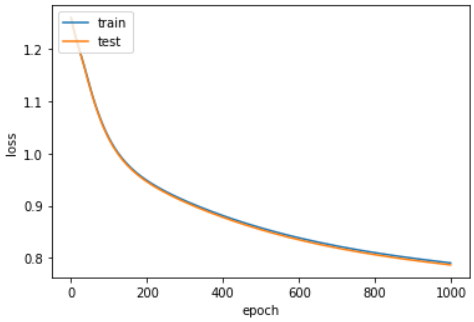


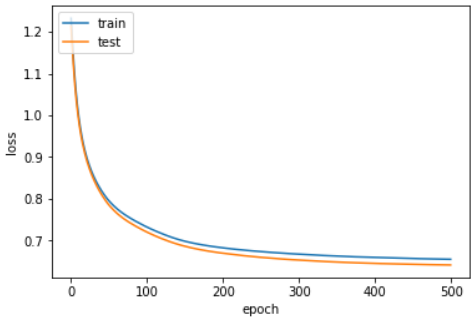
Figure Appendix D-2 Scatter plot of the three-dimensional representation produced by taking the first three principal components obtained from different sets of input attributes with the color specification of motor_UPDRS scores

a) PCA based on 16 vocal attributes; b) PCA based on total 19 attributes

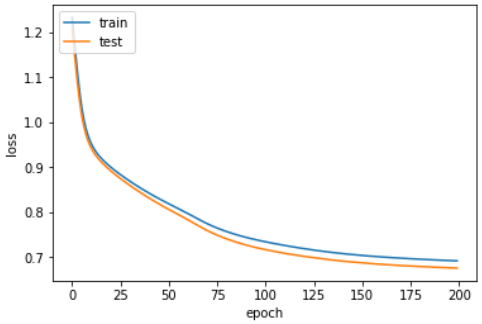
Appendix E



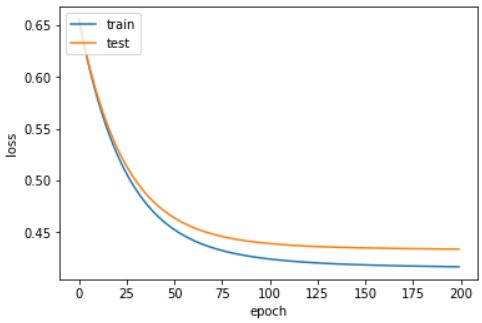
AE architecture	16-3-16
Activation function	sigmoid
Optimizer	SGD
Loss function	MSE
Learning rate	0.01
Batch size	64
Epochs	1000



AE architecture	16-3-16
Activation function	sigmoid
Optimizer	Adam
Loss function	MSE
Learning rate	0.001
Batch size	64
Epochs	500



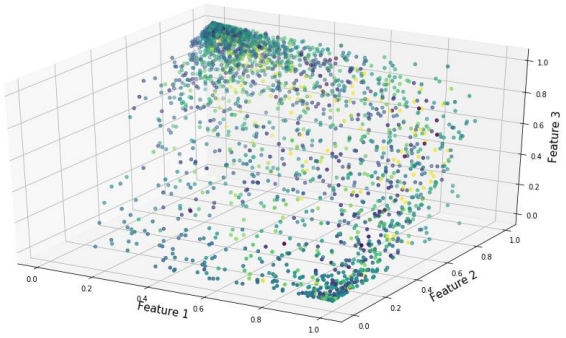
AE architecture	19-3-19
Activation function	sigmoid
Optimizer	Adam
Loss function	MSE
Learning rate	0.001
Batch size	64
Epochs	200



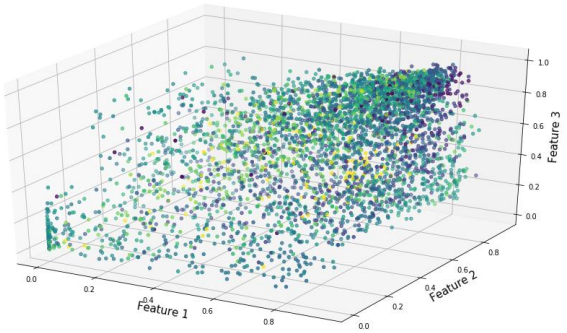
AE architecture	19-7-19
Activation function	sigmoid
Optimizer	SGD
Loss function	Binary cross entropy
Learning rate	0.01
Batch size	100
Epochs	200

Figure Appendix E-1 Visualization of training autoencoders with different hyperparameters and number of epochs

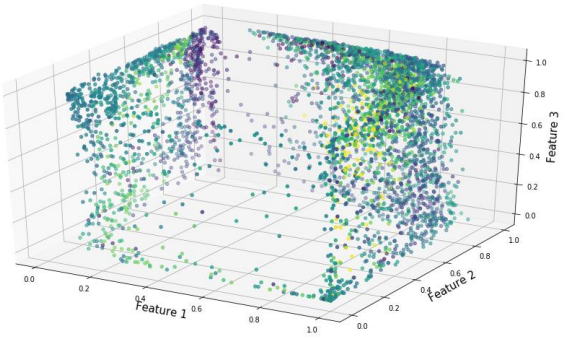
Appendix F



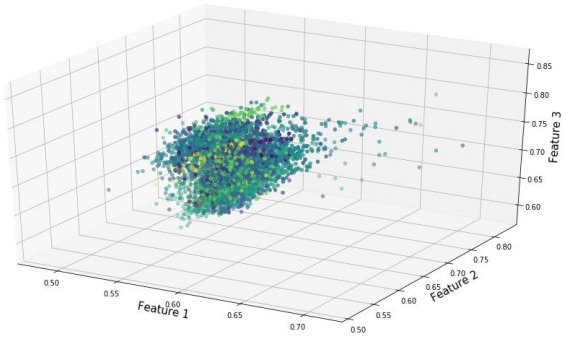
AE architecture	16-3-16
Activation function	sigmoid
Optimizer	SGD
Loss function	MSE
Learning rate	0.01
Batch size	64
Epochs	1000



AE architecture	16-3-16
Activation function	sigmoid
Optimizer	Adam
Loss function	MSE
Learning rate	0.001
Batch size	64
Epochs	500



AE architecture	19-3-19
Activation function	sigmoid
Optimizer	Adam
Loss function	MSE
Learning rate	0.001
Batch size	64
Epochs	200



AE architecture	19-7-19
Activation function	sigmoid
Optimizer	SGD
Loss function	Binary cross entropy
Learning rate	0.01
Batch size	100
Epochs	200

Figure Appendix F-1 Scatter plots of the three-dimensional codes produced by training autoencoders of different architectures and trained with different hyperparameters