Deep Recurrent Neural Networks for Fault Detection and Classification

by

Jorge Ivan Mireles Gonzalez

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Masters of Applied Science
in
Chemical Engineering

Waterloo, Ontario, Canada, 2018

AUTHOR'S DECLARATION

Jorge Ivan Mireles Gonzalez

Abstract

Deep Learning is one of the fastest growing research topics in process systems engineering due to the ability of deep learning models to represent and predict non-linear behavior in many applications. However, the application of these models in chemical engineering is still in its infancy. Thus, a key goal of this work is assessing the capabilities of deep-learning based models in a chemical engineering applications. The specific focus in the current work is detection and classification of faults in a large industrial plant involving several chemical unit operations. Towards this goal we compare the efficacy of a deep-learning based algorithm to other state-of-the-art multivariate statistical based techniques for fault detection and classification. The comparison is conducted using simulated data from a chemical benchmark case study that has been often used to test fault detection algorithms, the Tennessee Eastman Process (TEP). A real time online scheme is proposed in the current work that enhances the detection and classifications of all the faults occurring in the simulation. This is accomplished by formulating a fault-detection model capable of describing the dynamic nonlinear relationships among the output variables and manipulated variables that can be measured in the Tennessee Eastman Process during the occurrence of faults or in the absence of them. In particular, we are focusing on specific faults that cannot be correctly detected and classified by traditional statistical methods nor by simpler Artificial Neural Networks (ANN). To increase the detectability of these faults, a deep Recurrent Neural Network (RNN) is programmed that uses dynamic information of the process along a pre-specified time horizon. In this research we first studied the effect of the number of samples feed into the RNN in order to capture more dynamical information of the faults and showed that accuracy increases with this number e.g. average classification rates were 79.8%, 80.3%, 81% and 84% for the RNN with 5, 15, 25 and 100 number of samples respectively. As well, to increase the classification accuracy of difficult to observe faults we developed a hierarchical structure where faults are grouped into subsets and classified with separate models for each subset. Also, to improve the classification for faults that resulted in responses with low signal to noise ratio excitation was added to the process through an implementation of a pseudo random signal(PRS). By applying the hierarchical structure there is an increment on the signal-to-noise ratio of faults 3 and 9, which translates in an improvement in the classification accuracy in both of these faults by 43.0% and 17.2%

respectively for the case of 100 number of samples and by 8.7% and 23.4% for 25 number samples. On the other hand, applying a PRS to excite the system has showed a dramatic increase in the classification rates of the normal state to 88.7% and fault 15 up to 76.4%. Therefore, the proposed method is able to improve considerably both the detection and classification accuracy of several observable faults, as well as faults considered to be unobservable when using other detection algorithms. Overall, the comparison of the deep learning algorithms with Dynamic PCA (Principal Component Analysis) techniques showed a clear superiority of the deep learning techniques in classifying faults in nonlinear dynamic processes. Finally, we develop these same techniques to different operational modes of the TEP simulation, achieving comparable improvements to the classification accuracies.

Acknowledgements

Foremost, I would like to thank my parents for their constant support and love through my whole life. They have been the rock where I can always lean on to gather strength to accomplish my goals. My brothers and family also deserve a thank you for their warmhearted support.

I would like to sincerely thank my supervisor, Dr. Hector Budman, for his constant support and guidance through this research. As well, I would like to thank my co-supervisor, Dr. Ali Elkamel, for giving me this great opportunity and his feedback during this research completion.

To all my friends, thank you for your constant support. To my girlfriend, thank you for your continuous words of support and encouragement even in moments of crisis, your love has make has motivated me through the completion of this thesis.

Dedication

To my honorable father,

beloved mother,

and to my brothers.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Background on Fault Detection

In the modern process industry, the implementation of control schemes and systems for the detection and diagnosis of process disturbances or faults have become almost mandatory due to the increased demands for quality and profitability. A fault is defined as the deviation of a variable from an acceptable range of operation (Isermann, 2006). Also, since the effect of faults often propagates and leads to other faults it is imperative to detect them soon after their occurrence. Process incidents and shutdowns are frequent in the petroleum and chemical industries and many of these have been considered preventable. These incidents and faulty operations have resulted, just in the US alone, in an estimated loss of 20 billion US dollars per year (Nimmo, 1995). To avoid or mitigate these losses industrial plants nowadays are highly integrated with hundreds of sensors and multiple control loops.

Currently, advanced feedback control schemes such as internal model control (IMC) and model predictive control (MPC), have been successfully used for the stabilization of processes. However, in the presence of large process disturbances and manipulated variable constraints, these control schemes might not be sufficiently resilient to avoid abnormal operation (Chiang, Russell, & Braatz, 2001). A typical process monitoring system is composed of three basic parts: a detection algorithm, a diagnosis algorithm and a control action. The detection algorithm main objective is to detect an abnormal behavior in the system. Most detection algorithms are based on matching the output of the system to a previously defined “normal” operation. After detecting an abnormality, the diagnosis algorithm will gather the data and attempt to detect patterns in the information to determine the root cause of the abnormality. Then, depending on the fault detected and its effect on the system the corrective actions required to return the system to normal operation might be implemented immediately or be scheduled for later implementation depending on the severity of the fault.

A fault is considered observable when its effects on the system can be observable from a set of measured variables (M. Bin Shams, Budman, & Duever, 2011). Observability is an important topic for any fault detection and diagnosis problem because a lack of observabiliy or a low resolution can have detimentral effects on the detection or diagnosis ability of the fault detection scheme. The lack of ability to detect/diagnose the fault is due to unobservability or distinguishability problems. The unobservability problem arises when the effect of a fault on the system cannot be observed by the available measured variables. On the other hand, the distinguishability problem is related to the fact that different faults have similar effect on the measured variables used for detection/diagnosis. It is important to consider that both the control scheme and the observability of faults play key roles with respect to the observability and distinguishability problems. For example, the better the tracking performance of the control scheme is the less variability that can be observed in the controlled variable with respect to its set point and thus the less ability to distinguish the fault from other faults or from normal operation. Therefore, a trade-off commonly exists between the performance of the control scheme and the observability.

Fault detection and diagnosis can be classified into two main approaches depending on the way the data is used to infer the fault: mechanistic model based approach and data-driven model based approach (Chiang et al., 2001). The model based fault-detection approach consists of inferring the fault from input and output data and a mathematical model generally based on first principles. In order to formulate a succesfull model a priori knowledge of the physics and first principles' equations involved are required in addition to the data used for a priori model calibration and for fault inference. Figure 1 display the structure of the model-based fault detection approach. In this approach a set of measured input signals **U** and output signals **Y** are obtained from the process. Then, based on these signals, the detection method generates a set of features, that might be residuals $\boldsymbol{(r)}$, parameter estimates **(**$\hat{\theta}$**)** or state estimates $\boldsymbol{(\hat{x})}$**,** that are compared with normal features, i.e. features that are observed while the system is operated without fault referred to as "normal state". To detect changes on the measured features at any given time and compare them to normal features a set of symptoms $\boldsymbol{s}$ (Isermann, 2004) is defined that is informative about the faults in the process.

On the other hand, data-driven model based detection methods are an alternative to mechanistic model-based algorithms for monitoring the process that does not need an explicit mathematical model for detection. Figure 2 shows the basic structure of the data-driven models (Qin, 2009). For methods based on such data-driven models the online measurements of the process are directly compared with the normal behavior, and occurrence of the fault is assessed from a measure of differences between faulty to normal operation where these differences are calibrated a priori (Yin, Ding, Xie, & Luo, 2014). Nowadays, these data based methods have evolved from just being able to analyze one variable to be able to learn correlations between the process variables and even capture the time-varying behavior of the process.

In the current project we apply data-driven methods for fault detection. Specifically, we have developed deep learning neural network models for the detection and distinguishability of faults in a popular chemical benchmark case study, the Tennessee Eastman Process (TEP) simulator. The schemes tried during the research use both static as well as dynamic correlations for detection and diagnosis where the advantage of the dynamic versus the static approach are studied. In this research the concepts of disturbances, fault and classes are used with same meaning. During the original TEP article developed by Downs and Vogel, the 20 different process disturbances that were considered were also used in fault detection studies as faults.

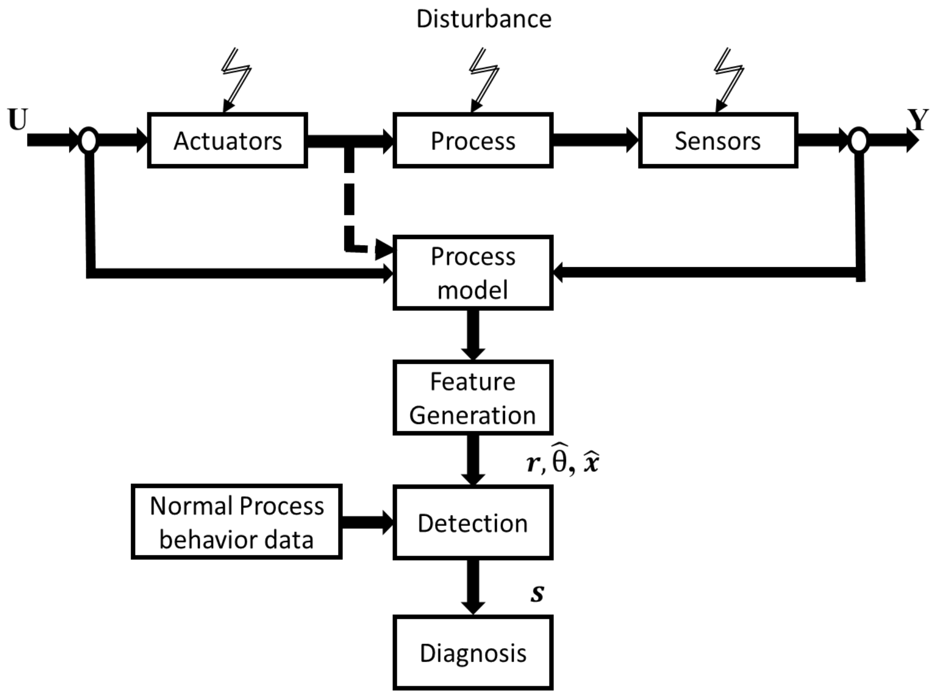


Figure 1. Basic structure model-based fault detection schemes

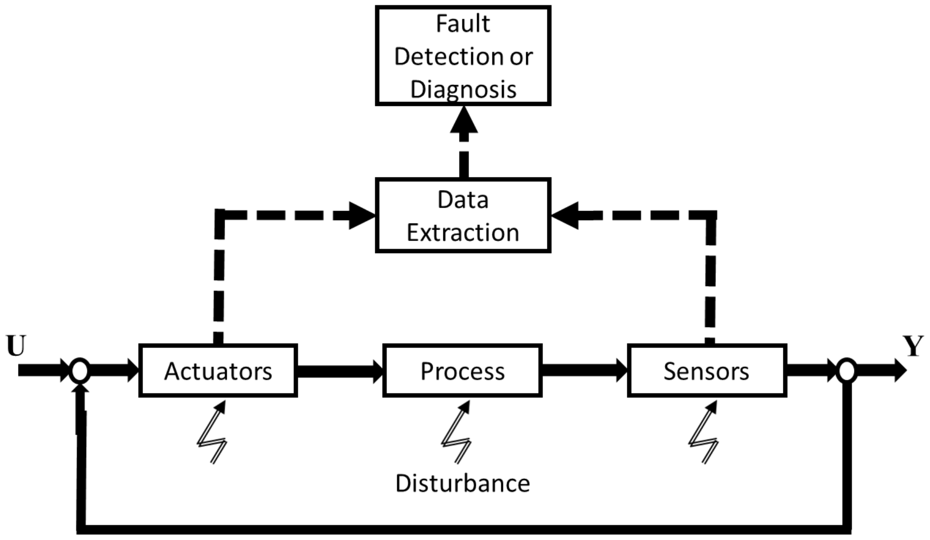


Figure 2. Basic structure of the data-driven models for fault detection

1.2 Background Deep Learning

The behaviors of biological systems have frequently inspired new ideas for mathematical algorithms such as Artificial Neural Networks(ANN). The idea that gave birth to ANN was to develop mathematical models that are able to mimic the decision making process and the basic descriptive functions of the human brain. The first mathematical model for ANN was introduced in 1943 by neurophysiologist Warren McCulloch and mathematician Walter Pitts in their paper "A Logical Calculus of the Ideas Immanent in Nervous Activity". That work described mathematically the way the human nervous system works as a net of interconnected neurons, where each neuron contains an axon and a soma, and how this network propagates excitation forward through the interconnections composing the network (McCulloch & Pitts, 1943). The model developed could recognize from the input two different categories, positive and negative, by testing a function $\mathrm{f(x, w)}$, where $\boldsymbol{x}$ is the input and $\boldsymbol{w}$ is a weight. However, the weights in this original algorithm had to be selected manually.

In the late 1950’s Frank Rosenblatt created the perceptron, which became the first model that solved the weighting parameter problem by self-learning the value of the weights by presenting to the model examples of the inputs of each category (Rosenblatt, 1958). In 1960 Bernard Widrow developed the Adaptive Linear Neuron (ADALINE) that involves units that could also self-learn the weight values (Widrow, 1960). Even though the perceptron and ADELINE were linear, a key difference between both units was that the former used class labels whether the latter used continuous predicted values. Although it was not the first algorithm to self-learn its weights, ADALINE included a special training algorithm called stochastic gradient descent which became one of the most popular training algorithms for models today. During the 1960's there was a lot of interest in these models. However, this excitement dipped when in 1969 Marvin Minsky and Seymour Papert identified some key limitations of the proposed linear networks when trying to learn XOR function by a single perceptron (Minsky & Papert, 1969).

By the mid-1980's a second wave of research on Artificial Neural Networks occurred that was motivated by the development of the back-propagation algorithm, a new learning procedure that improved all the previous results obtained by any neuron-like units (Rumelhart, Hinton, & Williams, 1986). This new wave of research was further promoted by new studies in the field of connectionism (Touretzky & Hinton, 1985). Connectionism stated that by increasing the number of interconnected simple neuron-like units it was possible for a network to learn and identify more complex behaviors. Furthermore, research based on Artificial Neural Networks continued to flourish during the 1990's with advances in sequence modeling that permitted to solve several problems encountered when modeling long sequences of data (Bengio, Simard, & Frasconi, 1994; J. Hochreiter, 1991), by using a novel gated neuron-like cell, the Long Short-Term Memory (LSTM) unit (S. Hochreiter & Urgen Schmidhuber, 1997). By this time the capabilities of Artificial Neural Networks to solve a variety of complex problems that have been recognized. However, the combined effects of advances in other fields, as Kernel Machines and graphical models, and requirements for high computational power required for ANN halted advances in the field.

This partial abandonment of Artificial Neural Networks' research lasted till mid-2000's when a third period of renewed interest in the field started and continue up to this day. Due to the expensive computational power required for successfully training bigger and deeper neural networks, researchers had to investigate solutions to accelerate the convergence and the performance of the algorithms. In 2006 Hinton et al. developed a new training strategy referred to as "*greedy layer-wise pre-training*", which involved an unsupervised pre training each layer separately of a network in order to boost the performance of the networks

(Geoffrey E. Hinton, Osindero, & Teh, 2006). This development gave birth to the term *deep learning*, due to the ability of training deeper and more complex networks (Lecun, Bengio, & Hinton, 2015). Currently, deep learning popularity continues to rise, due to advances and availability in computational power and algorithmic developments that have resulted in many profitable applications and broaden interest in research.

1.3 Research Objectives

The following main objectives were sought in the current project:

- Improve the detection and classifications of faults in a large industrial plant with several chemical processes. During this research we applied data from the Tennessee Eastman Problem, a widely used chemical engineering benchmark simulator for testing fault diagnosis and control strategies (Downs & Vogel, 1993).
- Compare different deep-learning based algorithms to other widely used multivariate statistical based techniques for fault detection and classification.
- Analyze the effect of including increasing amount of historical data to capture the nonlinear dynamic behavior with greater accuracy thus leading to improved fault classification ability.
- Analyze the effect of building a hierarchical structure and implement excitation signals into the system to increase the signal-to-noise ratio of difficult to observe faults.
- Assess the robustness of the proposed techniques by testing their ability to classify faults for different sets of operating conditions in the Tennessee Eastman Problem, i.e. different operating modes.

1.4 Thesis Outline

The current thesis is organized into five chapters covering the following topics:

- Chapter 1: General introduction about the contents of the thesis, research objectives and scientific approach.
- Chapter 2: Comprehensive literature review of the concepts applied during this research, a description of the plant simulation used in the case studies and a review of different methodologies previously used for detection and diagnosis.
- Chapter 3: Description of the procedure followed during this work for static data with deep learning and comparison with other widely used multivariate statistical methods. Comparisons were performed for different operating modes of the TEP plant simulator.
- Chapter 4: Description of the procedure followed during this work for dynamical data with deep learning and statistical methods used for comparison. This chapter includes results about the effect of the number of data samples, the application of a hierarchical structure and the implementation of a pseudo random signal. Moreover, it includes a comparison between the different TEP control schemes using the previous tools.
- Chapter 5: This chapter contains a summary of the finding of the research as well as recommendations for future work.

Chapter 2 Literature Review

2.1 Introduction to Artificial Neural Networks

Artificial Neural Networks were developed as a way to mathematically describe how the biological nervous system is capable of learning and processing complex information. Even though it has not been possible to fully reproduce the way the nervous system works, neural networks are derived as a narrow-sense abstraction which takes advantage of the knowledge of the basic functionality and organization of the neurons. Analogous to the way the human brain learns by experience, neural networks take advantage of examples (data samples) to generalize rules by executing several linear and nonlinear functions and by adjusting tuning weights of these functions (Goodfellow, Yoshua, & Aaron, 2016) to satisfy a specific quality of fit criterion.

To understand the potential engineering applications of artificial neural networks we consider the importance of some basic concepts and fundamentals of the nervous system and the relationship between both. The basic unit in the nervous system is called a neuron, shown in Figure 3, which is composed of three main parts: axon, soma and dendrites. In the presence of a stimuli the dendrites that are tree-like structures, act as receptors of the signal. Then, the soma is responsible of translating and delivering a response. This response is then propagated as an impulse through the axon to other surrounding units. It is estimated that the average human brain contains approximately 86 billion of neurons intertwined together to form a network (Herculano-Houzel, 2009).

An artificial computer model that resembles the main functional parts of a single neuron in the nervous system can be inferred from a comparison between Figure 3 and Figure 4. Just as with the dendrites in biological neurons the artificial neuron j has a number of n inputs $x_1, x_2, \dots, x_n$ represented as vector $\boldsymbol{x} \in \mathrm{R}^n$ and each input have an assigned weight $w_1, w_2, \dots, w_n$. After sensing an impulse, the soma processes the information with the goal of making a decision. The input to each artificial neuron is a weighted summation of the inputs $\boldsymbol{z}$ of the neuron given by

$$\boldsymbol{z} = \sum \boldsymbol{x_i w_i}, \qquad (1)$$

which may also include a bias term. Subsequently, a response to the input signal in the biological neural network is transmitted to another neuron through the axon. In the artificial neuron, this is equivalent to passing the value of z through an activation function f, which produces the value of an output signal

$$\boldsymbol{y} = f(\boldsymbol{z}). \tag{2}$$

Finally, with both biological and artificial neurons the overall network is formed by linking the output of each neuron to the input of another neuron. The number of outputs of a network might also be represented as a vector $\boldsymbol{x} \in R^n$, which depends on the number of outputs for every particular problem. Furthermore, during the learning phase, the weights and biases associated with each neuron can be altered depending on the efficacy of the results.

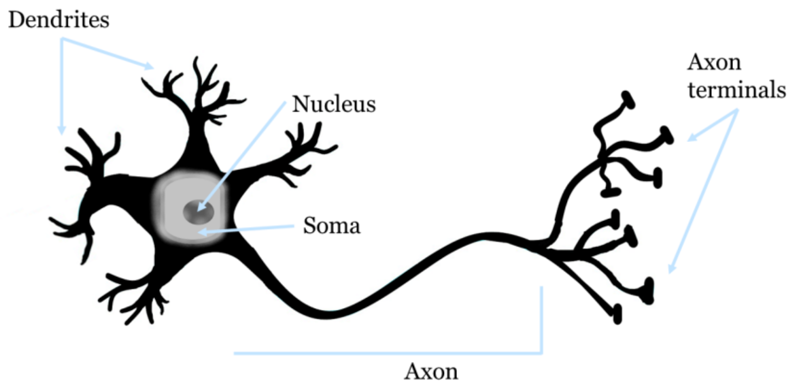


Figure 3. Schematic and functional description of a biological neuron.

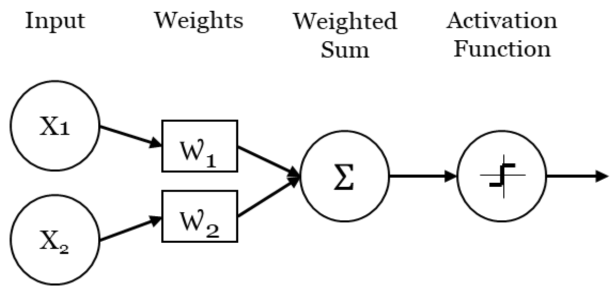


Figure 4. Schematic of a neuron in an artificial neural network

2.1.1 The Feed-Forward Neural Networks

Single perceptron's are only able to solve simple modelling tasks but they are not able to explain more complex phenomena such as those occurring in nonlinear systems, such as a simple XOR function (Goodfellow et al., 2016). So, in order to solve more complex tasks, inspired by the nervous system architecture models with several neurons and layers are assembled. This results in Feed-forward Networks, also called Multi-Layer Perceptron (MLP's), that are considered to be the prototypical models for deep learning and consists of networks of multiple interconnected layers of neurons. Each layer is stacked one after another and each neuron is connected to each neuron in the following layer. Figure 5 shows a schematic for a basic Multi-Layer Perceptron of three layers, which can be used to explain the architecture and mathematical description behind Artificial Neural Networks. In terms of its architecture, the network involves three groups of layers: input layer, one or more middle layers and an output layer. The input layer is fed with the data available in the problem. The middle layers are used by the network to learn the important features, contained in the data necessary to address a particular problem. The last layer referred to as the output layer is responsible for providing a final answer that may be a class label in classification problems or continuous values in general prediction problems. It is important to note that there are no limitations on the number of layers nor number of neurons that a network can be composed of. However, in feed-forward networks, there are no interconnections among neurons within the same layer and thus the data is transmitted only in one forward direction.

To summarize the mechanics of the neural network description let us first consider that each neuron S_j^{L1} is connected to each input in the input layer $\boldsymbol{x_n} = \{\boldsymbol{x_{1j}}, \boldsymbol{x_{2j}}, \ldots.., \boldsymbol{x_{nj}}\}$, which has a specific weights assigned to each input $\boldsymbol{w_n} = \{\boldsymbol{w_{1j}}, \boldsymbol{w_{2j}}, \ldots.., \boldsymbol{w_{nj}}\}$ and a bias term $\boldsymbol{\theta_{ij}}$. The weighted sum of the inputs is then calculated using the following formula

$$\mathbf{z}_{\boldsymbol{j}} = \sum_{i=0}^{n} \mathbf{x}_{\boldsymbol{ij}} \boldsymbol{w_{ij}} + \boldsymbol{\theta_{ij}} \tag{3}$$

The value of $\mathbf{z}_{\boldsymbol{j}}$ is then passed through an activation function $f(\mathbf{z}_{\boldsymbol{j}})$, the key decision making unit, which depending on the task to be accomplished can take different forms. Consider for instance the following activation function

$$f(\mathbf{z}_{\boldsymbol{j}}) = \frac{1}{1 + e^{-\mathrm{z}_j}}, \tag{4}$$

then the output of S_1^{L1} is computed as

$$S_1^{L1} = \frac{1}{1 + e^{-(\sum_{i=0}^{n} x_{ij} w_{ij} + \theta_{ij})}} . \tag{5}$$

As illustrated in Figure 5 below each node in the hidden layer is computed and then the resulting values are used as inputs for the following layer, the output layer in this simplistic example, that will use the same mathematical principles used to compute the previous layer.

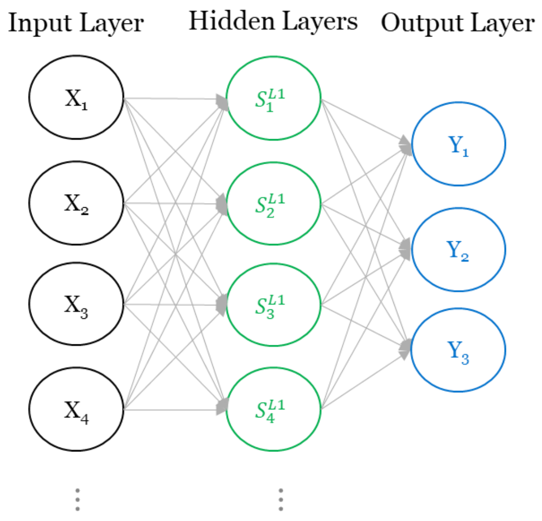
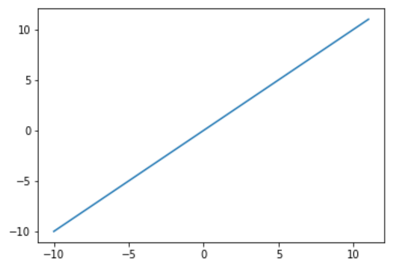
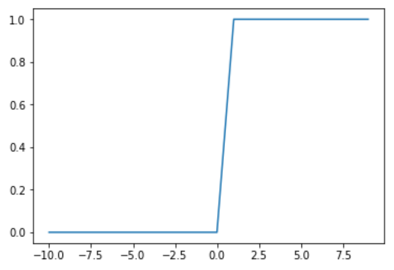
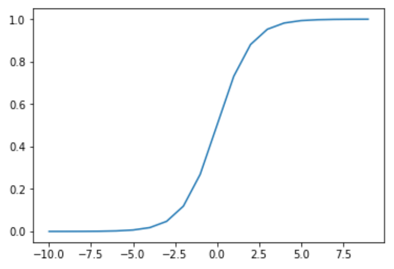


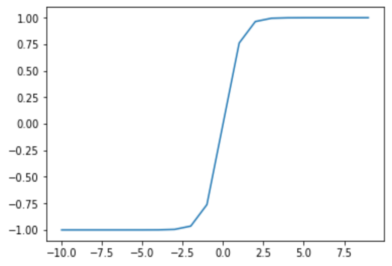
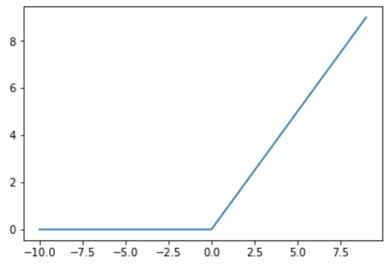
Figure 5. Schematic for a Multi-Layer Perceptron of 3 layers

2.1.2 Activation Functions

Table 1 shows the corresponding equation and graphical representation of the output that are obtained from each one of the most popular activation functions used in neural network models. The first type of activation function is the linear function, which is easy to compute but unable to learn complex nonlinear behavior. The binary step is an activation function usually used to compute the output of a classification network and the result of the function is either 0 or 1. In contrast, the three following rows, in table 1, represent the most popular activation functions used to describe nonlinearities to the network. The sigmoidal function computes an S-shaped output between 0 and 1, which will return an output between 0 and 1 depending on the value of the weighted sum of the inputs. The tanh is also an S-shaped function, though instead of the output being between 0 and 1 it ranges from -1 to 1. The output range that the tanh function computes is often more desirable than the sigmoidal due to the fact that it is zero-centered. Lastly, the rectified linear unit (ReLU) is currently the most used activation function in current neural network applications (Nair & Hinton, 2010). There are two main advantages of using ReLU as activation for units in deep neural networks. The non-linearity of the function will allow the network to create sparse representations, which translate in transferring only important information through the network. Once the neurons are active, the form of the activation function dictates a linear function, which allows for parameter sharing through the network while avoiding the vanishing gradient problem (Glorot & Bordes, 2011).

Table 1 Popular activation functions and plots

Name	Equation	Plot
Linear	$f(z) = az + b$	
Binary Step	$f(z) = \begin{cases} 0 \; for \; z < 0 \\ 1 \; for \; z \geq 0 \end{cases}$	
Sigmoid	$f(z) = \dfrac{1}{1 + e^{-z}}$	

Tanh	$f(z) = \tanh(z) = \dfrac{2}{1 + e^{-2z}} - 1$	
Rectified Linear Unit (ReLU)	$f(z) = \max(0, z) = \begin{cases} 0 \; for \; z < 0 \\ z \; for \; z \geq 0 \end{cases}$	

It is important to mention that any of the activation units presented in table 1 may be used in any of the layers of the network, either hidden or output. In addition, to all the previous layers, in many classification problems it is often desired to make the output vector a probabilistic distribution over n different classes to account for the presence of noise and stochastic disturbances. Such probabilistic classification can be generally addressed by applying softmax activation units in the output layers. The sotfmax unit, in rare cases, may also be used in the hidden layers if it is desired to choose only one of n internal variables. Through the use of softmax units it is possible to estimate the confidence of the network's prediction by analyzing the output vector $\mathbf{p}$, where $\sum_i \mathbf{p}_i = 1$. Therefore, the vector $\mathbf{p}$ will give a probability distribution that a particular data point belongs to a specific category. The formula of the softmax function is obtained by normalizing z_i, where z_i is defined as

$$\boldsymbol{z_i} = \sum \boldsymbol{x_i w_i} + \boldsymbol{b_i}, \tag{6}$$

which then is applied to the softmax activation function as

$$\boldsymbol{y_i} = \mathrm{f}(\boldsymbol{z_i}) = \frac{\exp(\boldsymbol{z_i})}{\sum_j \exp(z_j)}. \tag{7}$$

A strong prediction will result in only one category having a value close to 1, while a weak prediction will have the probability distributed to several categories.

2.1.3 Training Feed-Forward Neural Networks

The previous sections of this chapter have provided the basics of neural networks in terms of structure as well as the typical activation functions used for the neurons. However, a fundamental question for obtaining an efficient and useful network is how to calculate the optimal values of the parameters, i.e. weights and biases, in the network. The process of selecting the values of a network is referred to as training. The training process of a neural network is based on presenting the network with a large number of examples, known as training data, which are processed through the network and derive an iterative calibration of the values of the parameters such as to minimize a cost function.

Let us consider a simple example of a neural network formed only with rectified linear activation units that is presented with a large set of training examples to learn the behavior as well as the corresponding value of the output vector for each training example. Then, the parameters are typically calibrated by minimizing a mean squared error function as follows:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{y_i} - \widetilde{\boldsymbol{y_\imath}})^2 \tag{8}$$

Where $\boldsymbol{y}_i$ is the output vector computed by the neural network and $\widetilde{\boldsymbol{y_\imath}}$ is the desired output vector. Then, the objective is to find the particular combination of weights that will minimize the squared error of all training examples as per the following optimization problem:

$$\min_{\boldsymbol{\theta}} \mathrm{MSE} \tag{9}$$

Where $\boldsymbol{\theta}$ represents all the weights and biases in the network and y_i is given by equations (7) and (8). Additionally, the choice of the activation units and the purpose of the network will define the cost function that will be used throughout the network. For example, if the objective is to solve a binary classification problem then the output activation function can be chosen either of sigmoidal or softmax type. However, for different activation functions the loss function will be different where for sigmodial a binary cross-entropy loss will be used while for softmax a multiclass cross-entropy loss is generally employed.

It should be noticed that by introducing non-linearities into the neural network typical loss functions such as the sum of square error that are convex with respect to parameters when using linear models become non-convex with respect to the parameters of the neural network. This means that the cost function of a neural network, based on non-linear activation units, contains many combinations of parameters and this may result in several local minima and saddle points. Typically, problems that contain many non-linear features are trained iteratively by using gradient descent based algorithms that drive the value of the cost function to a minimum but that may not be necessarily the global minimum. This same procedure is applied in training neural networks with activation units such as sigmoidal or rectified linear units. Thus, in order to understand the training of neural networks the basics of gradient descent must be understood.

Gradient descent is an optimization algorithm for finding the set of parameters that minimize a fit function f based on a data training set, by sequentially iterating along the gradient of the function to find local minima (Shamir, 2003). For example, a simple linear function

$$f(\boldsymbol{x}) = mx + b \qquad (10)$$

with two initially unknown parameters, the slope m and the intercept b, the parameters that minimize the mean squared error function:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - (mx_i + b))^2. \qquad (11)$$

are obtained by the method of steepest descent (Rosenbloom, 1956), as follows. First, the values of the parameters are initialized and the gradients of the error function with respect to each parameter given by the partial derivatives are evaluated, i.e.

$$\frac{\partial E}{\partial m} = \frac{2}{n}\sum_{i=1}^{n} -x_i(y_i-(mx_i+b)) \qquad (12)$$

and,

$$\frac{\partial E}{\partial b} = \frac{2}{n}\sum_{i=1}^{n} -(y_i-(mx_i+b)). \qquad (13)$$

Based on these gradients, successive iterations are conducted in the direction of the steepest descent by updating the values of the parameters,

$$\Delta m = -\epsilon\frac{\partial E}{\partial m} \qquad (14)$$

$$\Delta b = -\epsilon\frac{\partial E}{\partial b}, \qquad (15)$$

where the gain ϵ is a hyper parameter referred to as the learning rate since it determines the magnitude of parameter change allowed in the direction of the gradients. Eventually, after a few iterations, this procedure may reach a minimum in the error function. This same procedure could be applied for updating the values of the weights and biases for each neuron of a neural network, with the difference that the form of the cost function f given above will change depending on the activation function. However, there might be cases where the error function is non convex, as illustrated in Figure 6, where we might get stuck in a local minimum depending on the learning rate ϵ.

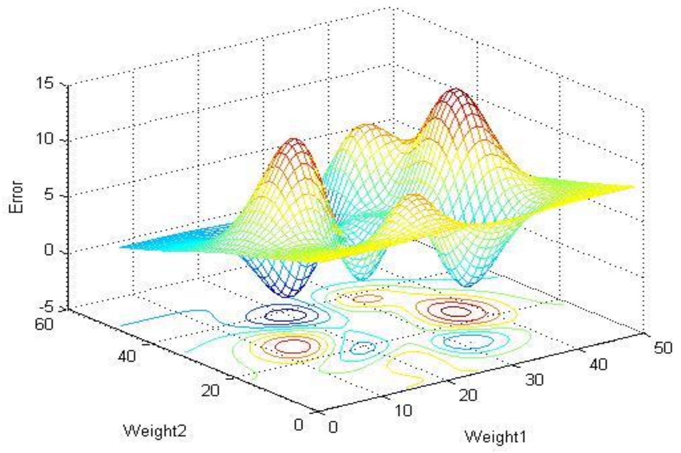


Figure 6. Illustration of error surface with respect to changes in the two weights

When applying a gradient descent method for training a neural network, it should be remembered that feedforward neural networks are applied on a set of inputs **x** that are propagated through the hidden layers to produce an output **y**. Thus, for each new set of inputs it is possible in principle to compute a new error between the output of the network and the expected output. Accordingly, a recursive learning approach can be implemented where the newly calculated error is fed-back through the network to drive changes of the network's parameters to minimize the total error function through a gradient descent approach. The error is propagated backward through the network in the form of gradients of the cost function with respect to the parameters that describe how the cost function changes with respect to changes in the weights of each individual connection in the previous layer thus updating iteratively each weight of the network to minimize the error between the network output and the expected value. This recursive approach for training neural networks with the help of gradient descent is known as backpropagation (Rumelhart et al., 1986).

Mathematically, the backpropagation algorithm is illustrated for a simple example for a network with two parameters w_1 and w_5. Figure 7 shows a simple 3 layered network that is used to illustrate this situation. In this case the error function is computed as

$$\mathrm{E} = \frac{1}{2} \sum_{j \in output} \left(y_j - O_{Y_j}\right)^2, \tag{16}$$

where O_{Y_j} is defined as the output computed by the neural network and y_j is the expected value. First, define all Z_i as the sum of weighted inputs and O_i as the output of the neurons, so from feedforward training the following Z_i and O_i are defined:

$$\mathrm{Z}_{\mathrm{H}_1} = \mathrm{w}_1 * \mathrm{X}_1 + \mathrm{w}_2 * \mathrm{X}_2,\ O_{H_1} = Sigmoid(Z_{H_1}), \tag{17}$$

$$Z_{Y_1} = w_5 * O_{H_1} + w_6 * O_{H_2},\ O_{Y_1} = Sigmoid(Z_{Y_1}). \tag{18}$$

The backpropagation procedure is started by computing the gradient of the error

$$\frac{\partial E}{\partial O_{Y_j}} = -\left(y_j - O_{Y_j}\right). \tag{19}$$

The gradient of the total error with respect to w_5, i.e. $\frac{\partial E}{\partial w_5}$ is obtained by applying the chain rule as follows:

$$\frac{\partial \mathrm{E}}{\partial \mathrm{w}_5} = \frac{\partial \mathrm{E}}{\partial \mathrm{O}_{\mathrm{Y}_1}} \frac{\partial \mathrm{O}_{\mathrm{Y}_1}}{\partial \mathrm{Z}_{\mathrm{Y}_1}} \frac{\partial \mathrm{Z}_{\mathrm{Y}_1}}{\partial \mathrm{w}_5}\ , \tag{20}$$

where

$$\frac{\partial E}{\partial O_{Y_1}} = -\left(y_1 - O_{Y_1}\right), \tag{21}$$

$$\frac{\partial O_{Y_1}}{\partial Z_{Y_1}} = O_{Y_1}\left(1 - O_{Y_1}\right), \text{ and} \tag{22}$$

$$\frac{\partial Z_{Y_1}}{\partial w_5} = O_{H_1}. \tag{23}$$

Then, the update of the weight w_5 at each step of the backpropagation procedure is

$$\Delta \mathrm{w}_5 = \epsilon \frac{\partial \mathrm{E}}{\partial \mathrm{w}_5}, \tag{24}$$

where ϵ is the learning rate. Due to the back-propagation procedure and the interconnectivity of the network, the deeper the network is, the larger the effect of the changes in the parameters on the outputs of the network. This propagation effect can be seen by applying the back-propagation procedure to the middle layer in the network of Figure 7. In this case the gradient of the total error with respect to w_1, i.e. $\frac{\partial E}{\partial w_1}$, is obtained by applying the chain rule to this problem as follows:

$$\frac{\partial \mathrm{E}}{\partial \mathrm{w}_1} = \frac{\partial \mathrm{E}}{\partial \mathrm{O}_{\mathrm{H}_1}} \frac{\partial \mathrm{O}_{\mathrm{H}_1}}{\partial \mathrm{Z}_{\mathrm{H}_1}} \frac{\partial \mathrm{Z}_{\mathrm{H}_1}}{\partial \mathrm{w}_1}, \tag{25}$$

Where H_1 is the output from one of the neurons in the middle layer. Using the same procedure as for the output layer we define

$$\frac{\partial O_{H_1}}{\partial Z_{H_1}} = O_{H_1}\left(1 - O_{H_1}\right) \tag{26}$$

$$\frac{\partial Z_{H_1}}{\partial \mathrm{w}_1} = X_1. \tag{27}$$

However, in this case, we have to compute the $\frac{\partial E}{\partial O_{H_1}}$ term as

$$\frac{\partial E}{\partial O_{H_1}} = \frac{\partial E_{O_{Y_1}}}{\partial O_{H_1}} + \frac{\partial E_{O_{Y_2}}}{\partial O_{H_1}}, \tag{28}$$

since a change in w_1 will affect both outputs. First, the error for the first output with respect to the output of the first middle neuron is:

$$\frac{\partial E_{O_{Y_1}}}{\partial O_{H_1}} = \frac{\partial E_{O_{Y_1}}}{\partial Z_{Y_1}} * \frac{\partial Z_{Y_1}}{\partial O_{H_1}}, \tag{29}$$

where

$$\frac{\partial E_{O_{Y_1}}}{\partial Z_{Y_1}} = \frac{\partial E_{O_{Y_1}}}{\partial O_{Y_1}} * \frac{\partial O_{Y_1}}{\partial Z_{Y_1}} = -\left(y_1 - O_{Y_1}\right) * O_{Y_1}\left(1 - O_{Y_1}\right) \tag{30}$$

which is computed from equation (21) and (22). Then, we can compute

$$\frac{\partial Z_{Y_1}}{\partial O_{H_1}} = \mathrm{w}_5. \tag{31}$$

The same procedure is applied to $\frac{\partial E_{O_{Y_2}}}{\partial O_{H_1}}$, resulting in

$$\frac{\partial E_{O_{Y_2}}}{\partial O_{H_1}} = \frac{\partial E_{O_{Y_2}}}{\partial Z_{Y_2}} * \frac{\partial Z_{Y_2}}{\partial O_{H_1}} = -\left(y_2 - O_{Y_2}\right) * O_{Y_2}\left(1 - O_{Y_2}\right) * w_7. \tag{32}$$

Finally, we have all the terms in order to compute the update in w_1 as

$$\Delta \mathrm{w}_1 = \epsilon \frac{\partial \mathrm{E}}{\partial \mathrm{w}_1} \tag{33}$$

where the derivative in the RHS of this equation is calculated from equation (25). This same procedure is applied to each of the other weights from the grid and the values are improved iteratively during the training process.

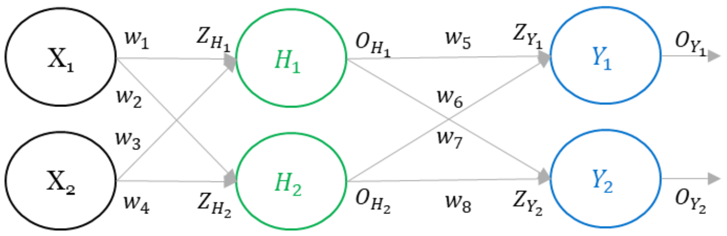


Figure 7. Simple 3 layered neural network

2.1.4 Challenges and Optimization of Gradient Descent Algorithm

Studies on the Gradient Descent algorithm for training neural networks revealed several challenges. For example, the use of the Gradient Descent method with randomly initialized parameter values for training very deep networks, i.e. typically networks with a number of hidden layers larger than 3, revealed that the procedure often converged to local minima (Tesauro, 1992). To address this problem Bengio et al. proposed the greedy layer-wise training procedure for deep networks, which was essentially an unsupervised method for initializing the parameters of each layer near a local minimum in order to avoid the gradient descent to converge to local minima (Bengio, Lamblin, Popovici, & Larochelle, 2007). Then, over the years several additional optimization procedures have been proposed to improve the convergence of the gradient descent algorithm. Therefore, this subsection will discuss some of the challenges for training deep neural networks and the advances and modifications that were proposed to improve the convergence of the gradient descent algorithm.

One of the key challenges for the original gradient descent was that initially the algorithm was applied for the entire training data thus resulting in poor convergence when the algorithm evolved in flat regions of the error objective function. To address this problem modification to the original gradient descent algorithm known as Stochastic Gradient Descent (SDG) was proposed where the only difference is that the gradient is calculated with respect to each sample in the training data (L Bottou, 1991). This change improves the ability of the algorithm to avoid flat regions but at the cost of significant increase in computational time due to the need of computing the error and gradient for each sample. Later, in order to reduce the increase in computational time this algorithm was combined to a batch learning approach to develop Batch Gradient Descent. Batch Gradient Descent is based on applying information through the network as small batches instead than for individual samples, which leads to advantages in generalization and computational time (Léon Bottou, 1998).

The second major challenge for convergence of the gradient descent method is related to whether the error surface function changes sufficiently in the direction of steepest descent. Theoretically, it is possible to assess whether the error surface changes sufficiently along a certain direction by computing the Hessian matrix. The Hessian matrix is basically a matrix of second derivatives of the gradients with respect to changes in the value of the weights. By checking the Hessian, it is possible to identify particularly singular situations where the gradients vanish corresponding to an ill-conditioned Hessian matrix. This challenging situation was solved by introducing a new term into the cost function referred to as momentum, which helps filtering possible fluctuations in the surface of the error function, due to the non linearities in the system and/or random noise, in generally flat regions of this function. Essentially, when using momentum, a memory cell is introduced that keeps track of an exponential decaying moving average of the gradients in order to continuously force advancement in the direction of steepest descent (Polyak, 1964). Mathematically, this memory cell is described with the following formula:

$$\mathbf{v_i} = \alpha\mathbf{v_{i-1}} - \epsilon\mathbf{g_i}, \tag{34}$$

where α is a new hyper parameter with value between 0 and 1, which will determine the exponential decay rate of the gradients and where ϵ and g_i are the learning rate and the gradient of i respectively. It is important to noticed that the relationship between α and ϵ informs on how the previous gradients affect the direction of the steepest descent.

Furthermore, to improve convergence, the weight of the momentum term in the objective error function is updated as per the following function:

$$\Delta\mathbf{w_i} = \mathbf{v_i}. \quad (35)$$

The third important challenge for convergence of the gradient descent is the selection of the optimal learning rate hyper parameter ϵ. Usually, there is a trade-off in the selection of the learning factor: selecting a too large learning factor results in larger parameter corrections along the surface of the error function that could hamper its convergence by skipping over the optimum while selecting a learning factor that is too small will slow down the convergence speed. To resolve this problem, the concept of adaptive learning rates was developed. For example in the delta-bar-delta algorithm the individual learning rates are adapted depending on the sign of the partial derivatives of the error function (Jacobs, 1988). Recently, three main methods have been developed in order to optimize the selection of the learning rate: AdaGrad, RMSProp, and Adam.

AdaGrad is a method that individually adapts the learning rates of each parameter based on the scaling of the error function such as the values of each learning rate is divided by the sum of squares of the historical accumulation of each gradient (Duchi, Hazan, & Singer, 2011). Mathematically, the implementation is based on a recursive formula for the accumulated gradients;

$$\mathbf{r}_\mathrm{i} = \mathbf{r}_{\mathrm{i}-1} + \frac{\partial E}{\partial \theta_i} * \frac{\partial E}{\partial \theta_i}, \quad (36)$$

Then the gradient equation is modified as follows:

$$\Delta\mathbf{w_i} = -\frac{\boldsymbol{\epsilon}}{\boldsymbol{\delta}+\sqrt{\mathbf{r}_\mathrm{i}}}\frac{\partial E}{\partial \theta_i}, \quad (37)$$

Where δ is a very small number that is solely introduced to avoid a division by 0. Even though AdaGrad performs well for problems of convex optimization, it has been founf inefficient in non-convex problems where the updating algorithm drops excessively the learning rate due to the squared accumulation of the gradient (Goodfellow et al., 2016).

RMSProp was developed based on the AdaGrad given above to achieve better performance in non-convex problems by implementing an exponential weighted moving average filter for the accumulation function of the gradients (Geoffrey E. Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012) that is described by the following formula:

$$\mathbf{r}_\mathrm{i} = \rho \mathbf{r}_{\mathrm{i}-1} + (1-\rho)\frac{\partial E}{\partial \theta_i} * \frac{\partial E}{\partial \theta_i}, \tag{38}$$

Where ρ is a user selected decay factor that will constrained the window of gradients that is retained at each instance of the gradient search. Currently, RMSProp is one of the most popular optimization algorithms used to train deep neural networks.

Lastly, Adam, named after the term “Adaptive moments”, is another optimization method that involves a combination between RMSProp and momentum. When applying momentum to RMSProp the decaying moving average of the gradients which is referred to as the first moment of the gradient must be stored. On the other hand, in the RMSProp method, the exponential weighted moving average of the accumulation of the gradients, known as the second moment of the gradient (Kingma & Ba, 2014) must be calculated and stored. Mathematically, the first moment is estimated via the following recursive formula

$$\mathbf{s}_\mathrm{i} = \rho_1 s_{\mathrm{i}-1} + (1-\rho_1)\frac{\partial E}{\partial \theta_i} \tag{39}$$

while the second moment is:

$$\mathbf{r}_\mathrm{i} = \rho_2 \mathbf{r}_{\mathrm{i}-1} + (1-\rho_2)\frac{\partial E}{\partial \theta_i} * \frac{\partial E}{\partial \theta_i}. \tag{40}$$

Then, following Adam’s method, corrections to both moments can be made in order to account for any initialization bias as per the following formulas:

$\tilde{\boldsymbol{s}}_\mathrm{i} = \frac{\mathbf{s}_\mathrm{i}}{1-\rho_1^i}$ and

(41)

$$\tilde{\mathbf{r}}_\mathrm{i} = \frac{\boldsymbol{r}_\mathrm{i}}{1-\boldsymbol{\rho}_2^\mathrm{i}}. \tag{42}$$

After computing the moment corrections, the weights are updated by taking into account the corrections according to the following function

$$\Delta\boldsymbol{\theta}_\mathbf{i} = -\epsilon\frac{\tilde{\mathbf{s}}_{\boldsymbol{i}}}{\boldsymbol{\delta}+\sqrt{\tilde{\mathbf{r}}_\mathrm{i}}}. \tag{43}$$

In a recent comparison paper that takes into account most of the modern gradient descent optimization methods it has been demonstrated that the Adam method performs better than

Adagrad and RMSProp due its bias-correction step (Ruder, 2016). Thus, Adam is currently considered the most robust optimization method to train deep neural networks and will be applied during this project for the optimization problem.

2.1.5 Data sets and Regularization

One of the major challenges for neural network models is to provide them with the ability to predict accurate results for inputs that have not been used for model calibration. This capacity of models to predict with data never seen before is known as generalization. To achieve good model generalization ability, the data is generally divided in at least two sets of information, a training set and a testing set. During the training procedure the neural network is presented with the training data only and the model parameters are calculated so as to minimize the error function, known as training error. After minimizing the training error, the generalization error is tested by applying the trained neural network for the testing set of data. In order to evaluate the performance of the model, the gap between the testing and training error is particularly assessed in order to identify whether the model might be underfitting or overfitting the information. The underfitting problem arises when the model is not able to reduce the training error to an acceptable standard. On the other hand, overfitting arises when the model is not able to generalize, which translates in a widening of the gap between the training and testing data.

When training a model, usually, the training error and testing error are simultaneously monitored until either one of them stops decreasing. If the errors achieved are not satisfactory the training procedure will continue with the goal of improving the model's feature extraction capabilities or by taking actions to prevent overfitting. Several techniques have been developed to prevent overfitting in neural networks, which includes regularization, early stopping and dropout.

Regularization is a broad concept that has been applied in many statistical models in a number of ways. However, in deep neural networks regularization refers to penalizing a particular norm of a parameter/s within the objective function of the optimization problem, e.g. the sum of square errors between predictions and data. Therefore, in order to calculate the total gradient of the objective function the gradients of both the error function and the regularization functions have to be calculated. Mathematically, in the objective function the regularization term is added as follows:

$J(\mathbf{\theta}) = Error + \alpha R(\boldsymbol{\theta}),$ (44)

where α is a hyper parameter that weights the norm penalty $\mathrm{R}(\theta)$. Typically, in deep neural networks two types of norm are used for regularization of the parameters L^2 or L^1. When L^1 regularization is applied to a model the penalty term is the sum of absolute value of all the parameters, defined as

$\mathrm{R}_{L1}(\mathbf{\theta}) = \sum_{\mathrm{i}} |\mathbf{\theta}_{\mathrm{i}}|.$ (45)

In contrast, L^2 regularization is described as the sum of squared magnitude of the parameters,

$\mathrm{R}_{L2}(\mathbf{\theta}) = \frac{1}{2}\sum_{\mathrm{i}} \mathbf{\theta}_{\mathrm{i}}{}^2.$ (46)

Correspondingly, the gradients of these penalties terms, L^1 or L^2, can be compute as

$\frac{\partial R_{L2}}{\partial \boldsymbol{\theta}_{\mathrm{i}}} = sign(\boldsymbol{\theta}_{\mathrm{i}})$ and (47)

$\frac{\partial R_{L2}}{\partial \boldsymbol{\theta}_{\mathrm{i}}} = \boldsymbol{\theta}_{\mathrm{i}}.$ (48)

The main difference between both types of regularization is that L^1 generally leads to a sparse representation where many parameters are left unchanged in the training procedure, whereas the L^2 norm tends to distribute the changes required for minimization of the objective more equally among the parameters

The early stopping algorithm is a simple way for preventing overfitting when training deep neural networks. In this method of regularization, both the training error and the validation error are simultaneously monitored to ensure a certain balance between them at the solution. In theory, at the start of training, both errors will start decreasing. Then, if after certain time the validation error will start increasing but the validation error does not improve, the network is not trained any further (Prechelt, 1998).

Dropout was developed as a way to inexpensively prevent overfitting in deep neural networks with large number of parameters (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). The theory behind dropout is based on randomly dropping units in the neural network or preventing units from adapt during the training phase with the expectation that the model will generalize better in the absence of information. An illustration of dropout is shown in Figure 8. Technically, during each iteration or batch of iterations in the training procedure the value of each individual neuron is either kept depending on a probability p (hyper parameter) or dropped out by reducing its value to zero and cutting all the incoming and outgoing information to the neuron. It is important to denote that the process of dropping a neuron is only done during the training phase but when the model is tested for generalization the hyper parameter p value associated to the neuron is changed back to 1 as the neuron is used. Some of the most important advantages for dropout is that it is a computational inexpensive operation and it is able to work well with any type of model (feed-forward, recurrent, etc.) and its compatibility with other regularizers.

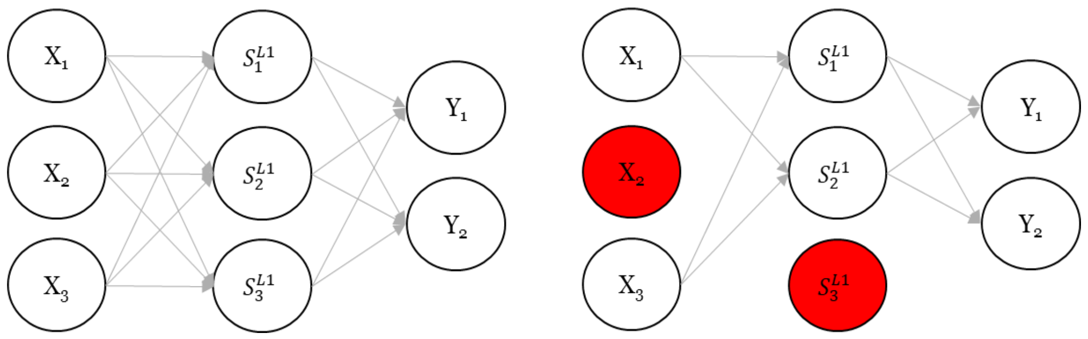


Figure 8 Dropout technique illustration. (Left) Network without dropout. (Right) Network with dropout.

2.1.6 Batch Normalization

Batch normalization is a tool developed by Ioffe and Szegedy, which main goal is to provide to every layer in a network inputs with zero mean and variance by applying normalization after each layer (Ioffe & Szegedy, 2015) . The goal of batch normalization is to address the so called “internal covariate shift”, where during training, the inputs to the next layers change as changes in the parameters from the previous layers occurs. To mitigate the “internal covariate shift” it is required to slow down the training when dealing with problems with significant nonlinearities by adopting small learning rates. Instead, this method implements a normalization layer in the model, which will normalize the data for each mini-batch. Batch normalization has proven to be a good additional tool for any network that results in speedier convergence time and also acts as a regularization technique (Ioffe & Szegedy, 2015).

Mathematically, batch normalization is defined for the values of $\mathbf{X}$ on a mini-batch $\boldsymbol{\beta} = [\boldsymbol{x_1} \ldots m]$, where we first start by calculating the mean $\boldsymbol{\mu_\beta}$ and variance $\boldsymbol{\sigma_\beta^2}$ for the mini-batch. Mathematically, these parameters are calculated as follows: $\mathbf{X_i}$

$$\boldsymbol{\mu_\beta} = \frac{1}{\mathrm{m}}\sum_{\mathrm{i}=1}^{\mathrm{m}}\mathbf{X}_{\boldsymbol{i}} \tag{49}$$

$$\boldsymbol{\sigma_\beta^2} = \frac{1}{\mathrm{m}}\sum_{\mathrm{i}=1}^{\mathrm{m}}(\mathbf{X}_{\boldsymbol{i}} - \boldsymbol{\mu_\beta})^2. \tag{50}$$

Then, applying the mean and the variance the data is normalized again as follows

$$\widehat{\mathbf{X}}_{\mathrm{i}} = \frac{\mathrm{x}_i - \boldsymbol{\mu_\beta}}{\sqrt{\sigma_\beta^2 + \epsilon}} \tag{51}$$

In addition to the normalization given above the output of the batch normalization layer is mathematically defined as

$$\boldsymbol{y_i} = \boldsymbol{\gamma}\boldsymbol{\widehat{x}_i} + \boldsymbol{\delta} \tag{52}$$

Where $\boldsymbol{\widehat{x}_i}$ is defined as our mini-batch normalize data and $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ will be new parameter to be learned by the new layer in order to scale and shift the data appropriately if necessary. Hence, while the outputs parameters are initialized to result in a zero mean and variance distribution, these parameters will be able to learn, through further training, another distribution that might yield better results.

2.1.7 Auto Encoder

An auto encoder is a type of unsupervised neural network model that it is trained to produce outputs that are a replicate of the values of the input. The idea of auto encoders was born soon after the idea of backpropagation learning. The motivation of auto encoders was to use the inputs as outputs of the network in an unsupervised fashion to extract relevant nonlinear higher-order properties of the inputs (Zemel & Hinton, 1994). An auto encoder is generally made of two functions: an encoder and a decoder function. The encoder function, represented as $\mathbf{h} = f(\mathbf{x})$, is responsible for detecting valuable properties of the data to generate a representational vector of the important properties of the input data. Alternatively, a decoder function , described as $\mathbf{r} = f(\mathbf{h})$, transforms the representation vector outputted from the encoder back to a close representation of the input $\mathbf{x}$.

When combining the encoder and the decoder the model is forced to select and learn valuable features of the input data that can be translated back to a similar representation of the input. Typically, when designing auto encoders, the model is restricted to learn an approximate representation of the input. Therefore, the model is encouraged to learn the most important characteristics of the data and avoid to only copy the values of the input. An illustration of a simple auto encoder is shown in Figure 9. In Figure 9, we can notice that we only have one hidden layer $\mathbf{h}$, an input layer $\mathbf{X}$, and the output layer $\widetilde{\boldsymbol{X}}$, which is an approximate representation of the input. In Figure 9, the encoder is the connection between the input layer and the hidden nodes $\mathbf{h} = f(\mathbf{x})$ and the decoder is the connection of the hidden nodes to the output $\mathbf{r} = g(\mathbf{h})$, where the functions $h(\mathbf{z})$ and $g(\mathbf{z})$ can be any activation functions. Furthermore, auto-encoders are trained with all of the same techniques as used to train feedforward networks.

Even though, auto encoders might seem superfluous since they are designed such as the output replicates the inputs, they have been found very effective for training deep neural networks. One of the most important applications of auto encoders is the ability of compressing the input dimensions without dropping relevant data and the ability of these models to detect and extract important features of the input data (G. E. Hinton & Salakhutdinov, 2006). It has also been observed that auto encoders, as an unsupervised pretraining tool for deep networks, enhance the generalization of deep models (Erhan, Courville, & Vincent, 2010). Lately, auto encoders have been found to be a powerful tool in generative models. There are several variations of the basic auto encoder model depending on their application where the two most popular ones are sparse and denoising auto encoders. The former auto-encoder model is trained where the parameters of the auto-encoder are added to the loss function for regularization purpose (Aurelio Ranzato et al., 2007), while the denoising auto encoder is used by altering the loss function to take into account the noise of the input thus acting as a noise filter (Vincent, Larochelle, Bengio, & Manzagol, 2008).

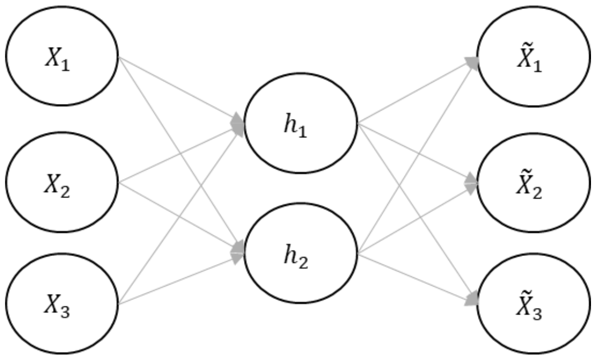


Figure 9. Simple auto encoder composed of 1 hidden layer

2.2 Recurrent Neural Networks

2.2.1 Recurrent Neural Networks

In order to understand the training of Recurrent Neural Networks it is important to emphasize their key differences from feed-forward networks. Feed-forward networks are considered as an "stateless", i.e. without dynamic states, neural network method, since for training the data is treated as static. In other words, feed-forward networks are trained to only describe relations between the variables of each observation at any given time but do not take into account history. Therefore, in some cases, valuable relationships may be lost as a result of neglecting the effect of dynamic correlations of current and past input and output data. In contrast, the idea of "statefull" models were developed as a way take advantage of the sequential relationships of the data with respect to previous inputs by storing the information in a "memory cell" over a span of time. The aim of storing the information in a memory cell is to force the model to consider previous values of the sequence such as to capture important pieces of the sequential information.

Accordingly, Recurrent Neural Networks is a type of neural network unit that was developed with the purpose of handling sequential data by expressing the values of the inputs as vectors through a sequence as $\mathrm{x}^{(1)}$,$\mathrm{x}^{(2)}$, ..., $\mathrm{x}^{(t)}$ (Rumelhart et al., 1986). One of the key elements that is used to capture the sequential nature of the data in recurrent models is parameter sharing, which enhances the model by sharing its parameters across a certain time span. This parameter sharing feature is usually applied to tying up the parameters of each variable through the network independently of the index of the sequence. This allows the model to enhance its generalization capacity to sequences not seen before and to sequences of different lengths. Figure 10 shows a schematic description of a simple example of recurrent neural networks. By comparing Figure 10 to a computational graph of a feed-forward network we could spot that the main difference consists in the idea of a cycle. This cycle represents the influence of the present and previous historical values of a variable on its own future value.

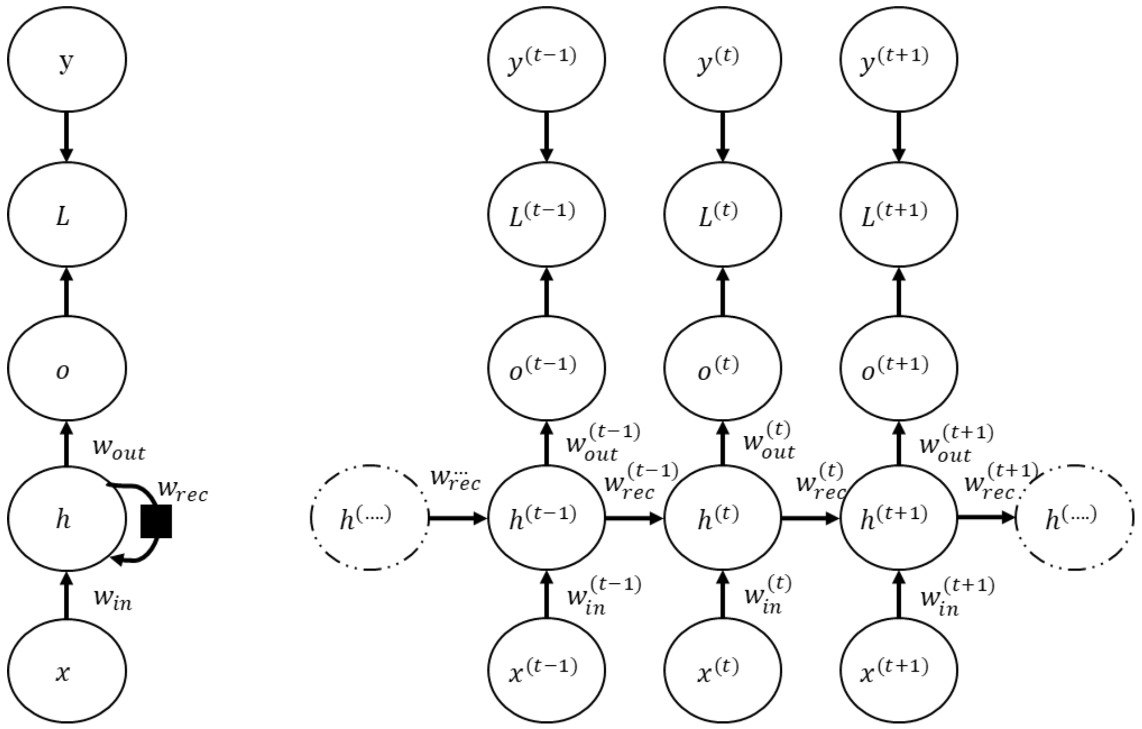


Figure 10. A fully connected recurrent neural network with a single neuron. (Left) Folded representation of the RNN with feedback connection in its hidden layer. (Right) Same RNN model, but this image describes the time-unfolded representation.

In order to understand how recurrent neural networks, work we first review how the information is handled by the network. Following the explanation above since recurrent networks are based on sequential information rather than static one, the input data in organized into matrixes to take into account the time span to be captured by the network model. As shown in Figure 10, recurrent models are composed of feedforward connections, which represent the flow of information from a neuron to another ($\mathbf{w}_{in}$ and $\mathbf{w}_{out}$), and recurrent connections, which represents important information stored from previous time steps ($\mathbf{w}_{rec}$). In the model in Figure 10, we can observe a fully connected recurrent model that produces an output at each time step and contain a recurrent connection in its hidden layer. In this case, we have to adapt the equation of the hidden layer to take into account the recurrent relation, which can be mathematically described as:

$$\boldsymbol{h}^{(\boldsymbol{t})} = f(\mathbf{z}) = f(\boldsymbol{w}_{\boldsymbol{in}}\boldsymbol{x}^{(\boldsymbol{t})} + \boldsymbol{w}_{\boldsymbol{rec}}\boldsymbol{h}^{(\boldsymbol{t-1})} + \boldsymbol{b}). \quad (53)$$

However, this recurrent connection can be altered depending on the problem. Thus, it is possible to build models with different feedback connections. In addition, we can also build models that produces one single output at the end of the sequence instead of producing an output every single time step.

Another key element to consider in recurrent models is that at each time step a sequence is processed and subsequently the model will restart itself. This means that the model will only learn features within the time span defined by the observations and not in between different observations. In principle, the optimization of the recurrent models with respect to the network parameters is done the same way as with feed-forward models. Few differences in training between feedforward and recurrent networks are presented in the following subsection.

2.2.2 Back-Propagation Through Time

One difference between backpropagation and recurrent networks is that for the latter a feedback connection is used to store important information of the sequence. This feedback connection alters the way the gradient is calculated in the backpropagation algorithm in order to take into account the sequential data and the recurrent parameter. The modification of the backpropagation algorithm that takes into account the sequential nature of the data is known as Backpropagation Through Time (BPTT). It is important to denote that even though the backpropagation algorithm must be modified the resulting algorithm is based on steepest gradient-descent concept.

The modification to the gradient calculation used for the recurrent network is illustrated for a simple example that is schematically described in Figure 11. As shown in Figure 11, we assume that the network is composed of 3 layers: input, hidden and output layer. To simplify the explanation we only consider 3 parameters to optimize W_{hy}, W_{xh} and W_{hh}. Then, by using feed forward propagation the following pertinent equation is obtained:

$$z_t = x_t W_{xh}^T + h_{t-1} W_{hh}^T, \tag{54}$$

$$h_t = \sigma_1(z_t), \tag{55}$$

$$y_t = h_t W_{hy}^T, \tag{56}$$

$$o_t = \sigma_2(y_t), \tag{57}$$

and the error function is defined as

$$E = f(O_t, Y_t^{real}). \tag{58}$$

Where σ_1 and σ_2 can be any type of activation function and the sub index t denotes the time step, where we consider that the recurrent model relates among the sequences of only two time steps. We observe from Figure 11 that a change in W_{hy} only affects E_t at that single time step. Therefore, using the chain rule the gradient of the output weight W_{hy} is

$$\frac{\partial E_t}{\partial W_{hy}} = \sum_t \frac{\partial E_t}{\partial W_{hy}} = \frac{\partial E_2}{\partial W_{hy}} + \frac{\partial E_1}{\partial W_{hy}} = \frac{\partial E_2}{\partial O_2}\frac{\partial O_2}{\partial y_2}\frac{\partial y_2}{\partial W_{hy}} + \frac{\partial E_1}{\partial O_1}\frac{\partial O_1}{\partial y_1}\frac{\partial y_1}{\partial W_{hy}} \tag{59}$$

On the other hand, any changes in W_{hh} and W_{xh} will impact the error function E_t depending on the previous time steps. Formally, it can be observed that a change in W_{hh} will impact E_2 when computing the value of h_1 and h_2, which depends on h_1.

$$\frac{\partial E_t}{\partial W_{hh}} = \sum_t \frac{\partial E_t}{\partial W_{hh}} = \frac{\partial E_2}{\partial W_{hh}} + \frac{\partial E_1}{\partial W_{hh}} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial z_k}\frac{\partial z_k}{\partial W_{hh}} = \frac{\partial E_2}{\partial o_2}\frac{\partial o_2}{\partial y_2}\frac{\partial y_2}{\partial h_2}\frac{\partial h_2}{\partial z_2}\frac{\partial z_2}{\partial W_{hh}} + \frac{\partial E_2}{\partial o_2}\frac{\partial o_2}{\partial y_2}\frac{\partial y_2}{\partial h_2}\frac{\partial h_2}{\partial z_2}\frac{\partial z_2}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z_1}{\partial W_{hh}} + \frac{\partial E_1}{\partial o_1}\frac{\partial o_1}{\partial y_1}\frac{\partial y_1}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z_1}{\partial W_{hh}} \tag{60}$$

This same procedure is applied to W_{xh}, which change will also impact the error function E_t. This results in a similar gradient as for W_{hh}, which can also be expanded in the same way as before, and it is given as:

$$\frac{\partial E_t}{\partial W_{xh}} = \sum_t \frac{\partial E_t}{\partial W_{xh}} = \frac{\partial E_2}{\partial W_{xh}} + \frac{\partial E_1}{\partial W_{xh}} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial z_k}\frac{\partial z_k}{\partial W_{xh}} \tag{61}$$

Alter computing all the gradients for each parameter, it is possible to update the weight using the same updating formula as for backpropagation thus ensuring that the updating proceeds in the direction of gradient descent.

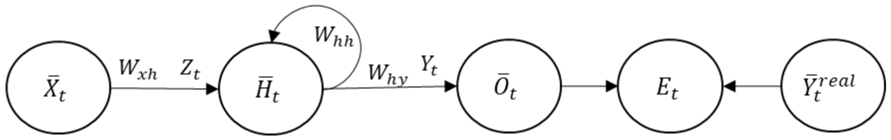


Figure 11. Illustration of simple recurrent network

2.2.3 Challenges in Recurrent Networks

Even though recurrent networks appear to be very powerful tools for learning time-based dependencies in sequential data, it is very challenging to train them properly especially when faced with long-term dependencies. Typically, when training recurrent networks, we might face either a vanishing gradient or an exploding gradient problem. The vanishing gradient problem is presented in a model when, during the training, the norm of the gradients corresponding to long term dependencies decreases exponentially to 0 along the training procedure. This makes the model to be unable to learn important long temporal correlations of the variables (J. Hochreiter, 1991). In contrast, the exploding gradient problem presents itself when the norm of the gradients corresponding to long term dependencies increases exponentially to an infinite value. Exploding gradient has the effect of making the model unstable hampering its overall learning ability (Bengio et al., 1994).

To explain the theory behind both of these training problems we should first remember the recurrent relationship, which can be expressed mathematically as

$$\boldsymbol{h}_{\mathbf{t}} = \boldsymbol{w}_{rec}\boldsymbol{h}_{\mathbf{t-1}}). \tag{62}$$

Therefore, we can observe that depending if the value of the $\boldsymbol{w}_{\boldsymbol{rec}}$ is either $w_{rec} < 1$ or $w_{rec} > 1$ the product in this recursive relation may vanish or explode. Generally, a network model is composed of traditional activation functions that have gradients in the range of 0 and 1. Therefore, when applying the backpropagation algorithm which uses the chain rule to compute the gradients, recursive equations between current and past gradients will appear which are susceptible to the vanishing or exploding gradient problems.

Due to increase in demand of deeper models and the popularity and efficacy of recurrent networks several research teams have devoted effort into solving the problems of vanishing or exploding gradients. There have been several proposed methods to solve both of these problems such as, adjusting the activation units for a more suitable options as ReLU’s and ELU’S (Clevert, Unterthiner, & Hochreiter, 2015) and echo state networks

(Jaeger, 2010). However, at the moment, the best practice when dealing with recurrent networks has been the application of new gated-type unit structures that are able to deal with both problems, specifically Long-Short Term Memory units (LSTM) (S. Hochreiter & Urgen Schmidhuber, 1997) and Gated Recurrent Units (GRU) (Cho et al., 2014). Some important characteristics of gated-type units is that they are able to change the model weights at each time step thus accumulating relevant information over multiple time steps and forgetting the state value once this information has been used.

2.2.4 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) units, a type of gated neuron unit, were developed as a way to propagate important long term dependencies in sequential data by introducing self-loops (S. Hochreiter & Urgen Schmidhuber, 1997). The LSTM is a complex gated unit composed of three gate units and a memory cell. Figure 12 shows a single LSTM unit. By inspecting Figure 12 we can see the four major operations in a LSTM unit the forget gate ($\mathrm{f_t}$), the input gate ($\mathrm{i_t}$), the output gate ($\mathrm{o_t}$), and memory cell ($\mathrm{c_t}$). The key component of the LSTM unit is the memory cell ($\mathrm{c_t}$), a tensor responsible for holding all the critical long term dependencies learned over time. These critical long term dependencies will be written and modified by the LSTM cell every time step depending of the information computed by the forget gate ($\mathrm{f_t}$),the input gate ($\mathrm{i_t}$) and the output gate ($\mathrm{o_t}$).

First, the input gate ($\mathrm{i_t}$) is responsible for evaluating what part if any of the past historical data should be kept for further training. Mathematically, the input gate is described as follows:

$$\boldsymbol{i}_{\mathbf{t}} = \sigma(\mathbf{W}^{(\mathbf{i})}\boldsymbol{x}_{\mathbf{t}} + \mathbf{U}^{(\mathbf{i})}\boldsymbol{h}_{\mathbf{t-1}} + \mathbf{b}^{(\mathbf{i})}) \,, \tag{63}$$

where $\mathbf{W}^{(\mathbf{i})}$, $\mathbf{U}^{(\mathbf{i})}$ and $\mathbf{b}^{(\mathbf{i})}$are respectively the input weight, recurrent weight and bias of the input gate and σ represents a sigmoidal activation function. The idea of the input gate is to allow the network to keep relevant information from the previous time step, but at the same time allow it to ignore "stale" information. The input gate is controlled by a sigmoidal function, which will produce a tensor with values of 0 and 1 that will express if either a value of a certain location in the previous step is relevant or not.

Once we have decided about the amount of information that it is worth keeping as determined by the input gate the information that is worth recording is determined in the memory cell ($\mathrm{c_t}$). The process of identifying what information is worth recording in the memory cell is composed of two parts, identifying the new information we want to record and the old information we wish to keep from the previous memory state. Mathematically, the memory cell is a combination of the forget gate and the input gate described by the following recursive equation:

$$\boldsymbol{c}_{\mathbf{t}} = \mathbf{f_t} * \mathbf{c_{t-1}} + \mathbf{i_t} * \tanh(\mathbf{W^{(c)}x_t} + \mathbf{U^{(c)}}\,\mathbf{h_{t-1}} + \mathbf{b^{(c)}}), \tag{64}$$

where $\mathbf{c_{t-1}}$, $\mathbf{W^{(c)}}$, $\mathbf{U^{(c)}}$ and $\mathbf{b^{(c)}}$are respectively the previous memory state, the input weight, recurrent weight and bias of the memory cell. The first part in equation 64 above is the information of the previous memory state worth keeping and the second part is the new information worth keeping. In addition, the forget gate is described as

$$\mathbf{f_t} = \sigma(\mathbf{W^{(f)}x_t} + \mathbf{U^{(f)}h_{t-1}} + \mathbf{b^{(f)}}), \tag{65}$$

Where now $\mathbf{W^{(f)}}$, $\mathbf{U^{(f)}}$ and $\mathbf{b^{(f)}}$ are respectively the input weight, recurrent weight and bias of the forget gate.

To conclude, as with most recurrent networks, the output is computed after every time step where this operation is also composed of two parts an output gate (o_{t}) and the final output ($\mathrm{h_t}$). The output gate is described as

$$\mathbf{o_t} = \sigma(\mathbf{W^{(o)}x_t} + \mathbf{U^{(o)}h_{t-1}} + \mathbf{b^{(o)}}), \tag{66}$$

Where $\mathbf{W^{(o)}}$, $\mathbf{U}^{(\boldsymbol{o})}$ and $\mathbf{b}^{(\boldsymbol{o})}$are respectively the input weight, recurrent weight and bias of the output gate. The final output of the LSTM is computed as

$$\mathbf{h_t} = \mathbf{o_t} * \tanh(\mathbf{c_t}), \tag{67}$$

where the $\tanh$ function part computes an intermediate tensor from the memory cell and where the output gate is a gate responsible of shutting off the output if necessary, therefore filtering the output. This output will prove an internal signal or a final value depending on the model.

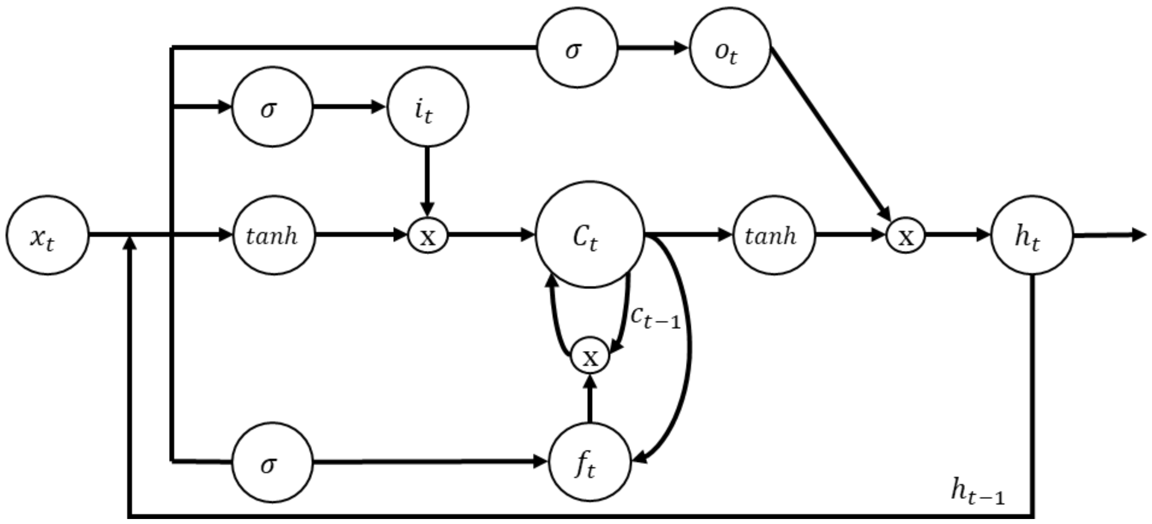


Figure 12. Illustration of the LSTM recurrent network "cell"

2.2.5 Gated Recurrent Units (GRU)

As explained earlier, LSTM units have been successful in capturing the dynamic behavior of data through the introduction of 4 self-loops, but how much of this architecture is actually necessary to capture the behavior. Gated Recurrent Units (GRU) were developed as a way to study other possible architectures capable of capture the dynamic behavior of the data while allowing the time scale control and forgetting capabilities (Chung, 2014, 2015). The difference between LSTM and GRU relies that GRU is composed of only two gated units, in charge of updating or resetting the state unit, and exposes the hidden content by not using a memory unit. Figure 13 shows us an illustration of a single GRU unit. By inspecting Figure 13 we can see the three major operations in a GRU unit the update gate ($\mathrm{z_t}$), the reset gate (r_t), and state ($\mathrm{h_t}$).

In GRU's the update gate ($\mathrm{z_t}$) is responsible of determining how much of the data presented is worth keeping to be passed to the future. Mathematically, the update gate is described as

$$\mathbf{z_t} = \sigma(\mathbf{W^{(z)}x_t} + \mathbf{U^{(z)}h_{t-1}} + \mathbf{b^{(z)}}) \ , \tag{68}$$

where $\mathbf{W^{(z)}}$, $\mathbf{U^{(z)}}$ and $\mathbf{b^{(z)}}$ are respectively the input weight, recurrent weight and bias of the update gate and σ represents a sigmoidal activation function. On the other hand, the reset gate ($\mathrm{r_t}$) is responsible of determining which parts of the previous information should be forgotten. The reset gate is described as

$$\mathbf{r_t} = \sigma(\mathbf{W^{(r)}x_t} + \mathbf{U^{(r)}h_{t-1}} + \mathbf{b^{(r)}}) , \tag{69}$$

where $\mathbf{W^{(r)}}$, $\mathbf{U^{(r)}}$ and $\mathbf{b^{(r)}}$ are respectively the input weight, recurrent weight and bias of the reset gate.

Finally, the state ($\mathrm{h_t}$) is in charge of holding the valuable information of the sequence for the current unit and propagating the value down the network. The final state is a combination of the information contained in both the update gate and the reset gate. Mathematically, described as

$$\mathbf{h_t} = \mathbf{z_t} * \tanh(\mathbf{W^{(h)}x_t} + \mathbf{r_t U^{(h)} h_{t-1}} + \mathbf{b^{(h)}}) + (1 - \mathbf{z_t}) * \mathbf{h_{t-1}} , \tag{70}$$

where $\mathbf{h_{t-1}}$, $\mathbf{W^{(h)}}$, $\mathbf{U^{(h)}}$ and $\mathbf{b^{(h)}}$ are respectively the previous state, the input weight, recurrent weight and bias of the state.

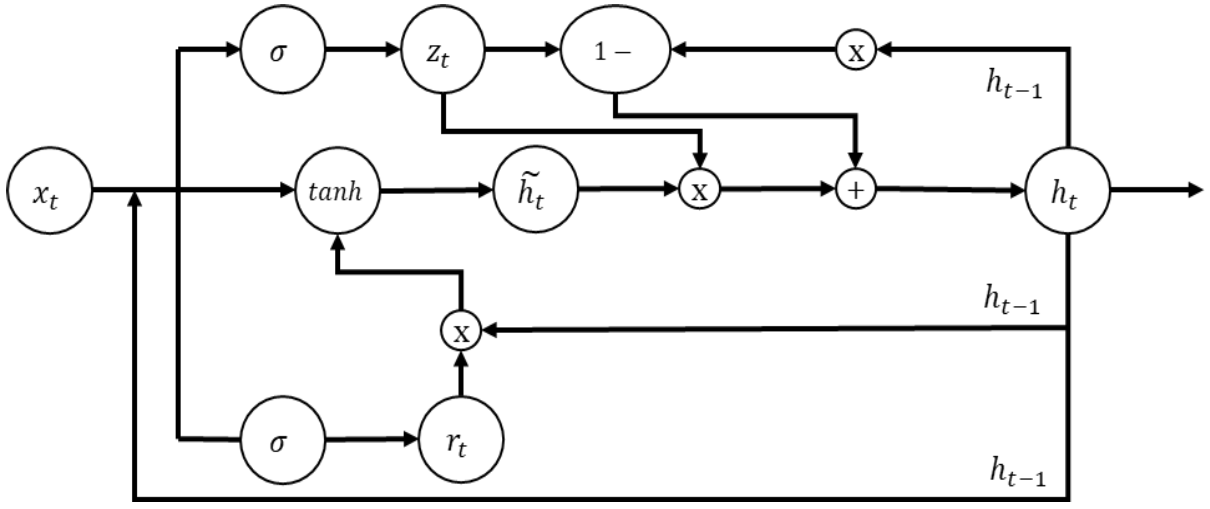


Figure 13. Illustration of the GRU recurrent network “cell”

2.3 Tennessee Eastman Process (TEP)

2.3.1 TEP Problem Description

The Tennessee Eastman Process (TEP) is a process which model was developed by Downs and Vogel (1993) in collaboration with the Eastman Chemical Company. The model has become a benchmark problem for testing, control and optimization strategies by academia and industry (Downs & Vogel, 1993). Different simulators have been produced for this problem either in Matlab or in other coding languages. The usefulness of this simulator stems from the fact that it represents an actual functioning chemical plant that involves several typical engineering units such as reactors and distillation units, several control loops and several controlled, manipulated variables and disturbances. In addition, the Tennessee Eastman plant has been widely successful as a source of information for several process monitoring and fault detection solutions (Bathelt, Ricker, & Jelali, 2015; Chiang et al., 2001; Kulkarni, Jayaraman, & Kulkarni, 2005; Larsson, Hestetun, Hovland, & Skogestad, 2001; Lau, Ghosh, Hussain, & Che Hassan, 2013; Rato & Reis, 2013; Ricker, 1996; Xie & Bai, 2015). The simulator of the TEP is used in the current study to test the proposed deep learning fault detection techniques.

The process simulated in the Tennessee Eastman is composed of 5 major unit operations: reactor, condenser, stripper, separator and compressor. In addition, the simulation contains several chemical reactions that translate on 8 different components in the simulation: 2 products (G and H), 4 reactants (A, C, D and E), a byproduct (F) and an inert (B). Figure 14 shows the process flowsheet of the Tennessee Eastman plant that describes the plant operation. The Tennessee Eastman simulation contains 53 total measurement composed of 22 continuous process variables, 19 sampled analyzer process variables and 12 manipulated variables. All these measurements can be simulated to contain Gaussian noise. The description and the label of each process measurements are listed in tables 2, 3 and 4 respectively.

Furthermore, it should be pointed out that the plant simulation in open loop is unstable due to the presence of highly exothermic reactions in the process reactor. Therefore, there have been several control schemes implemented in order to stabilize the simulation. The most popular Tennessee Eastman process control scheme was developed by Ricker, applying a decentralized system control scheme (Ricker, 1996). In addition to the process inherent instability, the simulation contains 20 pre-programmed faulty scenarios, which are described thoroughly in table 5. In 2015, Ricker release a revision to his original control scheme for the Tennessee Eastman that brought updates in some algorithms and structure of the code and additional process measurements and disturbances (Bathelt et al., 2015).

The labels considered in Tables 2 through 5 are the labels used in the original Tennessee Eastman Simulation. By giving these labels we intend to facilitate the traceability of these measurements for the readers. More information about the process model can be found in the original paper (Downs & Vogel, 1993). More details about the control schemes applied to the simulation can be found in the original Ricker control scheme (Ricker, 1996) and its revised version (Bathelt et al., 2015), which can be downloadable in http://depts.washington.edu/control/LARRY/TE/download.html.

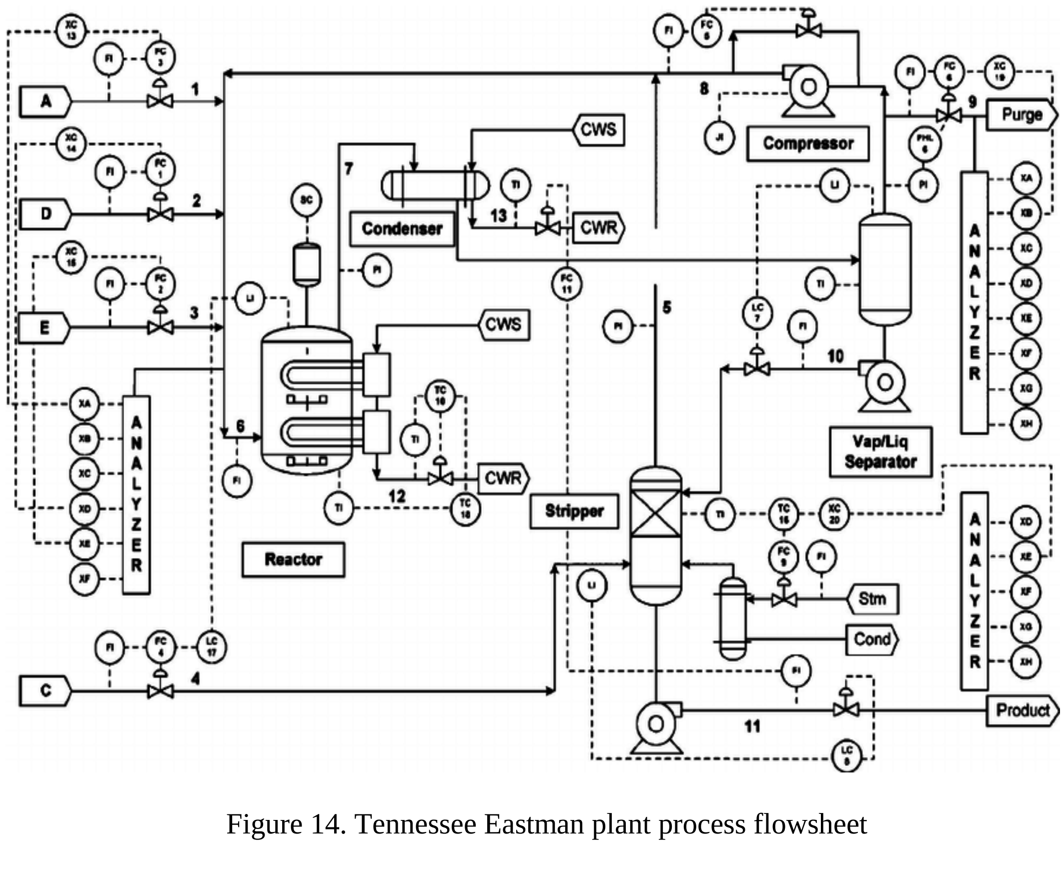


Figure 14. Tennessee Eastman plant process flowsheet

Table 2. Manipulated variables in Tennessee Eastman plant

Description	Label
D feed flow	XMV(1)
E feed flow	XMV(2)
A feed flow	XMV(3)
A and C feed flow	XMV(4)
Compressor recycle valve	XMV(5)
Purge valve	XMV(6)
Separator pot liquid flow	XMV(7)
Stripper liquid product flow	XMV(8)
Stripper steam valve	XMV(9)
Reactor cooling water flow	XMV(10)
Condenser cooling water flow	XMV(11)
Agitator speed	XMV(12)

Table 3. Continuous process measurements in Tennessee Eastman plant

Description	Label
A feed	XMEAS(1)
D feed	XMEAS(2)
E feed	XMEAS(3)
A and C feed	XMEAS(4)
Recycle flow	XMEAS(5)
Reactor feed rate	XMEAS(6)
Reactor pressure	XMEAS(7)
Reactor level	XMEAS(8)
Reactor temperature	XMEAS(9)
Purge rate	XMEAS(10)
Product separator temperature	XMEAS(11)
Product separator level	XMEAS(12)
Product separator pressure	XMEAS(13)
Product separator underflow	XMEAS(14)
Stripper level	XMEAS(15)
Stripper pressure	XMEAS(16)
Stripper underflow	XMEAS(17)
Stripper temperature	XMEAS(18)
Stripper steam flow	XMEAS(19)
Compressor work	XMEAS(20)
Reactor cooling water outlet temperature	XMEAS(21)
Separator cooling water outlet temperature	XMEAS(22)

Table 4. Sampled process measurements in Tennessee Eastman plant

Description	Label
Reactor feed analysis of A	XMEAS(23)
Reactor feed analysis of B	XMEAS(24)
Reactor feed analysis of C	XMEAS(25)
Reactor feed analysis of D	XMEAS(26)
Reactor feed analysis of E	XMEAS(27)
Reactor feed analysis of F	XMEAS(28)
Purge gas analysis of A	XMEAS(29)
Purge gas analysis of B	XMEAS(30)
Purge gas analysis of C	XMEAS(31)
Purge gas analysis of D	XMEAS(32)
Purge gas analysis of E	XMEAS(33)
Purge gas analysis of F	XMEAS(34)
Purge gas analysis of G	XMEAS(35)
Purge gas analysis of H	XMEAS(36)
Product analysis of D	XMEAS(37)
Product analysis of E	XMEAS(38)
Product analysis of F	XMEAS(39)
Product analysis of G	XMEAS(40)
Product analysis of H	XMEAS(41)

Table 5. Process disturbances in Tennessee Eastman plant

Description	Label	Type
A/C feed ratio, B composition constant	IDV(1)	Step
B composition, A/C ratio constant	IDV(2)	Step
D feed temperature	IDV(3)	Step
Reactor cooling water inlet temperature	IDV(4)	Step
Condenser cooling water inlet temperature	IDV(5)	Step
A feed loss	IDV(6)	Step
C header pressure loss-reduced availability	IDV(7)	Step
A, B, C feed composition	IDV(8)	Random
D feed temperature	IDV(9)	Random
C feed temperature	IDV(10)	Random
Reactor cooling water inlet temperature	IDV(11)	Random
Condenser cooling water inlet temperature	IDV(12)	Random
Reactor kinetics	IDV(13)	Drift
Reactor cooling water valve	IDV(14)	Stiction
Condenser cooling water valve	IDV(15)	Stiction

Deviations of heat transfer within stripper	IDV(16)	Random
Deviations of heat transfer within reactor	IDV(17)	Random
Deviations of heat transfer within condenser	IDV(18)	Random
Recycle valve of compressor, underflow separator, underflow stripper and steam valve stripper	IDV(19)	Stiction
Unknown	IDV(20)	Random

2.3.2 Previous Fault Detection and Diagnosis Techniques based on Statistical Approaches

Several data-driven statistical process monitoring approaches have been applied to the detection and diagnoses of disturbances in the Tennessee Eastman simulation. Each of these methods have shown different levels of success in detecting and diagnosing the 20 faults contained in the simulation. It is important to denote that several statistical studies have reported faults 3, 9 and 15 as unobservable due to the close similarity in the responses of the measured variables used to detect these faults (Chiang, Russell, & Braatz, 2000; Du & Du, 2018; Lau et al., 2013; Mohamed Bin Shams, Budman, & Duever, 2010). Therefore, during this section, we are reviewing the statistical approaches that have been used for the TEP and the corresponding level of performance of these methods.

Principal Component Analysis (PCA) is an unsupervised learning data driven technique based on orthogonal transformations for simplifying a multivariate data set by reducing the dimensionality while conserving the most relevant information (Hotelling, 1933; Pearson, 1901). PCA and many variations of the general PCA algorithm has been successfully applied to several studies for fault detection and diagnosis problem (Lau et al., 2013; Yin, Ding, Haghani, Hao, & Zhang, 2012; Zhang, 2009). This has usually yielded good performance detection rate in many observable faults, but deficient performance for the difficult to observe faults (<3%). One important variation to the normal PCA algorithm is the adjustment of the algorithm to take advantage of dynamic relationships in the data, known as Dynamic Principle Component Analysis (DPCA). DPCA implements time lag shifts to the data matrix to create duplicate vectors that, in theory, could learn dynamical relationships occurring in the data (Ku, Storer, & Georgakis, 1995). DPCA has been found to be able to improve the results of static PCA on observable faults, but it is still unable to significantly improve detection with respect to the faults that are difficult to observe faults (Chiang et al., 2001; Ku et al., 1995; Odiowei & Yi, 2010; Rato & Reis, 2013; Yin et al., 2012).

Independent Component Analysis (ICA) is also an unsupervised method based on decomposing multivariate data set into independent non-Gaussian vectors, addressing higher order dependencies (Comon, 1994). Typically, the detection rate of ICA is comparable to the detections rates obtained by PCA; however, it does provide improvement for some faults in the Tennessee Eastman simulation (Lee, Qin, & Lee, 2006; Rashid & Yu, 2012; Yang, Chen, Chen, & Liu, 2012; Yin et al., 2012; Zhang, 2009). As with PCA, we can make the ICA algorithm can be modified to take advantage of dynamic relationships in the data, known as Dynamic Independent Component Analysis (DICA). DICA has been found to be able to obtain high detection rates in all observable faults, but it is still unable to detect efficiently faults 3, 9 and 15 (Stefatos & Hamza, 2010).

Canonical Variate Analysis (CVA) is another dimensionality reduction algorithm that involves the selection of pair of variables between the inputs and output of two sets in order to maximize the correlation. In contrast with other methods, CVA is able to take into account sequential correlations in data sets with dynamic behavior (Larimore, 1990). The CVA algorithm is applied mathematically as a linear state-space model that is identified from time series input data and an output data. Therefore, CVA examine the data for linear combinations of the past that could explain the variation of the data in the future. CVA has been successfully applied to several studies for fault detection and diagnosis problem (Jiang, Huang, Zhu, Yang, & Braatz, 2015; Stubbs, Zhang, & Morris, 2012; Yang et al., 2012). With the help of lagged data CVA performance for detection of fault has achieved very good results for most faults in the Tennessee Eastman simulation except for 3, 9 and 15 but it has also been able to improve the detection rates for 3, 9 and 15 (Cao & Samuel, 2016; Chiang et al., 2001; Odiowei & Yi, 2010; Samuel & Cao, 2015).

In addition to the methods reviewed, there are other prominent data-driven statistical process monitoring solutions that have been applied to the Tennessee Eastman simulation. Partial Least Squares (PLS) is another dimensionality reduction method that based on maximizing the covariance between a predictor and a predicted matrix for each component in the space (Wold, 1985). The applications of the general PLS algorithm and other modified PLS algorithms has provided good results comparable to the performances of PCA and ICA (Chiang et al., 2000; Dong, Zhang, Huang, Li, & Peng, 2015; Odiowei & Yi, 2010; Yang et al., 2012).

Typically, when applying the algorithms showed above that compare the values of the normal state with values of the state in the presence of the fault, it is possible to answer the fault detection problem, i.e. whether operation is normal or faulty. However, by combining the results of these algorithms with classification techniques it is also possible to solve the diagnosis problem. Fisher Discriminant Analysis (FDA) is a supervised fault diagnosis method that takes into account the information between classes in order to maximizes the variance between classes and compute discriminants (directions) (Fisher, 1936; Mika, Ratsch, Weston, & Scholkopf, 1999). FDA have achieved good performance for the detection of observable faults in the Tennessee Eastman simulation (Chiang, Kotanchek, & Kordon, 2004; Chiang et al., 2000; Yang et al., 2012). Support Vector Machines (SVM) is another supervised classification technique based on transforming the input data into a high dimensional space to maximize the distance between two classes (Vapnik, 1998). SVM have been applied in the Tennessee Eastman simulation as a solution for both detection and diagnosis of faults (Chiang et al., 2004; Kulkarni et al., 2005; Mahadevan & Shah, 2009).Another solution to the problem of fault diagnosis is combining the results from the PCA algorithm in combination with a Cumulative Sum (CUSUM). This has showed to be a viable option to solve the detection problem of difficult to observe disturbances   due that the cumulative sum of values over time will be able to detect minor changes in the process variables that cannot be detected without using the CUSUM operation (M. Bin Shams et al., 2011).

Table 6. Previous results for the detection of difficult to observe faults with CUSUM-PCA

	CUSUM-PCA	
Fault	Statistics	$ARL_{o.c}$ (hrs)
3	LCS	127.05
9	SCS	8.2
15	SCS	41
3	T^2	102.4
9	T^2	276.05
15	T^2	89.65
3 & 15	T^2	41.3

This technique was applied for the detection of faults of difficult to observe faults and depicts the Average Run Length (ARL) to detect these faults in hours using a combination of CUSUM and PCA. Results are extracted from: **CUSUM-PCA** (Mohamed Bin Shams et al., 2010).

Table 7. Previous results for the detection of faults with different statistical techniques.

Fault	FDA	PLS	TPLS	MPLS	PCA	DPCA	CVA	CVA-ICA	SVM	ICA	CVA-KDE
1	100	99.88	99.88	100	99.88	99.88	100	100	99.8	99.59	99.75
2	98.75	98.63	98.88	98.88	98.75	99.38	99	100	98.6	7.08	99.5
3	7	14.25	24.25	18.75	12.88	12.25	1.4	91.79	N/A	16.14	73.03
4	100	99.5	100	100	100	100	100	100	99.6	95.42	99.88
5	100	33.63	100	100	33.63	43.25	100	100	100	100	99.88
6	100	100	100	100	100	100	100	100	100	100	99.88
7	100	100	100	100	100	100	100	100	100	100	99.88
8	98.13	97.88	98.5	98.63	98	98	98.4	99.37	97.9	97.29	96.63
9	6.25	14.5	23.5	18.75	8.38	12.88	0.7	95.79	N/A	7.29	92.26
10	87.13	82.63	91	91.13	60.5	72	90.1	99.58	87.6	91.25	96.63
11	73.38	78.63	86.13	83.25	78.88	91.5	80.5	100	69.8	79.17	99.38
12	99.75	99.25	99.63	99.88	99.13	99.25	100	100	99.9	98.54	99.5
13	95.63	95.25	96.13	95.5	95.38	95.38	96	100	95.5	97.5	96.13
14	100	100	100	100	100	100	100	100	100	99.79	99.88
15	12.63	23	29.88	23.25	14.13	19.75	9.7	97.9	N/A	11.46	99.5
16	83.25	68.38	90.75	94.28	55.25	67.38	91.6	99.37	89.8	80.41	99.13
17	96.63	94.25	96	97.13	95.25	97.25	97.6	100	95.3	90.42	98.13
18	90.75	90.75	91.88	91.25	90.5	90.88	90.8	98.74	90	84.58	99.25
19	87.88	26	82.88	94.25	41.13	87.25	98.1	100	83.9	75.83	99.88
20	81.88	62.75	78.38	91.5	63.38	73.75	91.3	99.58	90	93.96	97.63
21	52.75	59.88	66.38	72.75	52.13	61	65.8	N/A	52.8	N/A	N/A

All these results are for the detection of faults. Results are extracted from: **FDA** (Yin et al., 2012), **PLS/TPLS/MLPS** (Yin et al., 2012), **DPLS** (Odiowei & Yi, 2010), **PCA** (Chiang et al., 2001; Lau et al., 2013; Mahadevan & Shah, 2009; Rato & Reis, 2013; Stefatos & Hamza, 2010; Yin et al., 2014), **DPCA** (Chiang et al., 2001; Mahadevan & Shah, 2009; Odiowei & Yi, 2010; Rato & Reis, 2013; Stefatos & Hamza, 2010; Yang et al., 2012; Yin et al., 2014), **CVA** (Chiang et al., 2001; Odiowei & Yi, 2010), **CVA-ICA** (Yang et al., 2012), **ICA** (Stefatos & Hamza, 2010; Yang et al., 2012; Yin et al., 2014), **DICA** (Stefatos & Hamza, 2010), **CVA-KDE** (Odiowei & Yi, 2010; Yang et al., 2012), **SVM** (Chiang et al., 2004; Mahadevan & Shah, 2009).

2.3.3 Previous Detection and Diagnosis Approaches based on Deep Learning

Deep learning network models can also serve as a fault monitoring tool. Different deep learning approaches have been applied for the detection and diagnoses of disturbances in the Tennessee Eastman simulation. As with the statistical methods, each reviewed method have shown different results for the detection and diagnosis problem of the 20 faults contained in the simulation. In the case of deep learning methods, as with statistical methods, faults 3, 9 and 15 have been classified as difficult to observe due to the closeness of the responses in the measured variables (Lv, Wen, Bao, & Liu, 2016; Xie & Bai, 2015; Zhao, Sun, & Jin, 2018). Usually, to solve the problem of detection and diagnosis problem the data can be used as either static or dynamic

When approaching the data as static the input vectors of data at different time steps are considered as independent vectors. One of the first applications of neural networks in a chemical plant was developed to detect and diagnose faults in the Syschem plant, which contains 19 different faults (Hoskins, Kaliyur, & Himmelblau, 1991). In this work a three layered network with backpropagation was implemented resulting in valuable associations between system faults and input vectors providing good detection and classification of fault. In 2015 Xie and Bai proposed deep neural networks as solution for the diagnosis problem in the Tennessee Eastman simulation with some modifications (Xie & Bai, 2015). In their work they proposed the application of a hierarchical deep neural network, which is basically a model that separates the faulty data into different clusters and depending on the cluster to which newly acquired data belong a diagnosis can be produced. The classification results obtained by this method are promising for many observable faults and the hierarchical structure improves the classification rates from simple deep neural networks. In 2016 Lv et al. proposed implementing a stacked sparse auto encoder (SSAE) structure to a deep neural networks in order to extract important features from the input in order to improve the diagnosis problem in the Tennessee Eastman simulation (Lv et al., 2016). The diagnosis results applying this deep learning technique showed improvements, when compared to typical statistical techniques, for most observable faults; however, in the case of difficult to observe faults as 3, 9 and 15 the classification accuracy continue to be unsatisfactory. In addition, a comparison between a deep stacked networks (DSN) and deep stacked auto encoders (SAE) for the detection of faults was also studied (Chadha & Schwung, 2018).

While with static data we treat the input data vectors collected at each time interval as independent vectors for dynamic data the time dimension is explicitly consider by inputting the data in the form of matrices where the columns correspond to the sequential lagged variables measured at different time intervals (Li, Olson, & Chester, 1990). Recently, Long Short Term Memory units have been applied to the Tennessee Eastman Problem for the diagnosis of faults (Zhao et al., 2018). The article based the results in a dynamical model with LSTM units to learn the dynamical behavior from sequences of features of three time steps and the application of batch normalization to enhance convergence. This model is able to enhance the classification rates of most observable faults and slightly improves the results of the difficult to observe faults.

Table 8. Previous results for the detection and diagnosis of faults with deep learning techniques.

					DL - LSTM	
Fault Case	DSN	SAE	DL-SSAE	DL-HIER	(FP = 0.1)	(FP = 0.2)
1	90.8	77.6	100	97.7	100	100
2	89.6	85	99.75	97.2	100	100
3	14.4	79.4	N/A	N/A	68	90
4	47.6	56.6	100	99.6	100	100
5	31.6	76	98.88	92.8	100	100
6	91.6	82.8	100	99.5	100	100
7	91	80.6	100	99.5	100	100
8	90.2	83	97.88	62.3	94	95
9	16.3	50.6	N/A	N/A	60	88
10	63.2	75.3	99.25	81.8	90	95
11	54.2	75.9	89.25	52.5	87	93
12	87.8	83.3	99.75	55.3	90	95
13	85.5	83.3	99.75	50.4	88	95
14	89	77.8	95.13	69.5	100	100
15	26.7	55.5	N/A	N/A	62	94
16	74.8	78.3	99.5	77.9	81	84
17	83.3	78	99.75	71.4	93	98
18	82.4	83.3	99.5	84.3	93	97
19	52.4	67.7	96.75	97.2	100	100
20	44.1	77.1	99.38	73.4	83	91
21	31	16.7	N/A	N/A	38	73

It is important to denote that in the case of DL-LSTM we based this results in the ROC curves and drew lines in false positive rates to get the classification. Results are extracted from: **DSN&SAE** for detection (Chadha & Schwung, 2018), **DL-SSAE** for classification (Lv et al., 2016), **DL-HIER** for classification (Xie & Bai, 2015), **DL-LSTM** for classification (Zhao et al., 2018).

2.3.4 Different Control Schemes in the Tennessee Eastman Simulation

As explained in the previous section the Tennessee Eastman Simulation, has been widely applied for several applications of which one of the most important one is the testing of novel control schemes. Nowadays, three different control schemes for the TEP has been made available in the Tennessee Eastman Matlab simulator. These three modes have been made available in recent releases of the TEP called MultiLoop_mode3, MultiLoop_mode1 and MultiLoop_ Skoge _mode1 (Bathelt et al., 2015).

MultiLoop_mode3 and MultiLoop_mode1 where developed by Lawrence Ricker and the Simulink model of the control strategy is fully described in the paper "Decentralized control of the Tennessee Eastman Challenge Process" (Ricker, 1996). It is important to consider that both of these models were developed using decentralized control strategies. The decentralized control scheme is based on partitioning the whole plant into little sub-units, with one controller is sued to control each sub-unit. The key difference between Mode 3 and Mode 1 lies is that they consider different operating conditions and process initial values. These differences require the use of different control loops to ensure good control performance.

On the other hand, the MultiLoop_ Skoge _mode1 Simulink model of the control strategy is described in the paper “Self-Optimizing control of a large-scale plant: The Tennessee Eastman process” (Larsson et al., 2001). The control scheme applied on this mode introduces an idea by Morari et al (Morari, Arkun, & Stephanopoulos, 1980) and further developed by Skogestad (Skogestad, 2000) named “self-optimizing control”. This idea involves the selection of a set of controlled variables, which if controlled at constant setpoints will result in a near to optimal process operation (Skogestad, 2000). In general, the self-optimizing control approach provides information about the controlled and manipulated variables that are important for the control of the system

Chapter 3
Fault Detection and Classification using Static Data

3.1 Overview

In this chapter, statistical monitoring models using static data are applied to the Tennessee Eastman Process for fault detection and classification problems. This chapter covers two principal detection and diagnosis models: Principal Component Analysis (PCA) and Artificial Neural Networks (ANN). For the purpose of fault diagnosis with PCA we combined the structure of PCA with Support Vector Machine (SVM) to differentiate between different faults. PCA was considered for comparison since it is the most widely used approach to detection and diagnosis in both academic studies and industrial practice. Fault diagnosis with static data refers to the fact that only data available at a given time interval is used to diagnose a fault at that time interval. Therefore, the information contained in the vector of data at the particular time interval at which the fault is being diagnosed it only accounts for the correlation between variables but it does not account for dynamic correlations between the current and past time intervals. The use of dynamic data where dynamic correlations are considered is presented separately in Chapter 4.

The classification rates and false alarm rates resulting from models trained with static data will be tested with the 20 different in a fault diagnosis problem together. In addition, based on these results, a hierarchical scheme is proposed for both PCA and ANN models to assess whether this hierarchical static data based structure is able to improve the fault diagnosis results, as well as the false alarm rates. In addition, the robust performance of the PCA and ANN models is tested by comparing the results for different control schemes available for the TEP simulator.

3.2 Definitions and Methods

Process monitoring algorithms involve two objectives: fault detection and fault classification. Fault detection is usually viewed as a binary classification problem that is used to assess whether the process is normal or faulty. Typically, fault detection is trained by first analyzing the normal process information, which is then used to generate statistical limits of normal operation. Therefore, when a particular fault occurs the algorithm will detect the abnormal behavior by assessing whether the data is out of the normal range of operation. On the other hand, fault classification is a more complex multi-class classification problem where data is associated to different classes and where each class correspond to a particular fault. Hence, fault classification algorithms infer from different correlations between the variables whether a specific fault is occurring at any particular time. Following the above, in the current study we are merging the concepts of detection and classification into a single algorithm by adding another class which represent the normal operation condition. Thus by treating normal operation as an additional class, fault detection can be accomplished by simply assessing whether the data corresponds to normal operation or not.

In addition, in this research a differentiation is made between two types of faults occurring in the TEP problem: observable and difficult-to-observe. Observable faults are considered as fault that have discernible symptoms as compared to the noise considered in the TEP case study and have been classified with a good degree of accuracy using conventional fault detection or diagnosis techniques, e.g. multivariate statistical algorithms. On the other hand, the difficult to observe faults are defined as faults which symptoms are very similar to other faults, they exhibit a low signal to noise ratio and they have been reported as difficult to classify with conventional statistical techniques.

3.2.1 Process monitoring with Principal Component Analysis

As explained in the previous chapter, Principal Component Analysis (PCA) is an unsupervised learning data driven technique based on orthogonal transformations for simplifying a multivariate data set by reducing its dimensionality while conserving the most relevant information (Hotelling, 1933; Pearson, 1901). The PCA is a data reduction technique, given a data matrix $\mathbf{X}$ composed of n observations and p random variables, that computes a set of linear combinations k, where $p > k$**,** that explains the highest percentage of the total variation in the observations. This is computed by finding the sample covariance matrix $\mathbf{S}$ of matrix $\mathbf{X}$, which is equivalent to the calculation of the eigenvalue decomposition of $\boldsymbol{S}$. Accordingly, this equation is computed as follows: $\boldsymbol{S}$

$$\boldsymbol{S} = \frac{1}{\mathrm{n}-1}\mathbf{X}^{\mathrm{T}}\mathbf{X} = \mathbf{PD}\boldsymbol{P}^{\mathrm{T}} \tag{71}$$

Where **P** is an orthogonal matrix which columns represents the corresponding eigenvector of **S** and the diagonal elements of **D** represents the real non-negative values of the eigenvalues. Since PCA is a data reduction technique it results in a lower dimensional space where $\mathrm{k} < \mathrm{p}$ and the loading vectors' set is of dimension:

$$\mathbf{P}_k = (\mathrm{v}_1, \mathrm{v}_2 \dots \mathrm{v}_\mathrm{k}) \in \mathbf{R}^{\mathrm{pxk}} \tag{72}$$

Following the above a lower-dimensional space projection of X, can be described by the following score matrix:

$$\mathbf{Y} = \mathbf{P}_\mathrm{k}^\mathrm{T}\mathbf{X}_\mathbf{i}, \mathrm{i} = 1, \dots \mathrm{n} \tag{73}$$

The score space , which is the projection of the score matrix onto the p dimensional space, is

$$\widehat{\mathbf{X}} = \mathbf{T}\mathbf{P}^\mathrm{T} \tag{74}$$

Then, the residual matrix, which describes the remaining differences between the reduced dimensional space projected and the actual input, is

$$\mathbf{E} = \mathbf{X} - \widehat{\mathbf{X}} \tag{75}$$

PCA can be used for fault detection by using measures that quantify the statistically significant differences between the normal states and the faulty states. Particularly, there are two statistical measures that can be applied as fault detection techniques when applying PCA: $\mathbf{T}^2$ and $\mathbf{Q}$.

The $\mathbf{T}^2$ statistic is computed as

$$\mathbf{T}_\mathrm{i}^2 = \mathbf{x}_\mathrm{i}^\mathrm{T}\mathbf{P}_\mathrm{k}\mathbf{D}^{-1}\mathbf{P}_\mathrm{k}^\mathrm{T}\mathbf{x}_\mathrm{i} \tag{76}$$

where $\mathbf{x}$ is an observation vector of a process variable, $\mathbf{P}$ is the orthogonal matrix and $\mathbf{D}$ is a diagonal matrix which diagonal elements are the eigenvalues of the original **X**. The Hotelling's $\mathbf{T}^2$ statistics test represents the variation of the samples from the mean. On the other hand, $\mathbf{Q}$ statistis, also known as $\mathbf{Q}$ residual, measures the variation between the data samples representation in the k reduced projected space and each sample. $\mathbf{Q}$ residuals are computed as follows:

$$\mathbf{Q}_\mathrm{i} = \mathbf{r}_\mathrm{i}^\mathrm{T}\mathbf{r}_\mathrm{i} \tag{77}$$

Where r_i is the residual vector that is calculated using the following formula

$$\mathbf{r}_\mathrm{i} = (\mathbf{I} - \mathbf{P}_\mathrm{K}\mathbf{P}_\mathrm{k}^\mathrm{T})\mathbf{x}_\mathrm{i} \tag{78}$$

PCA is applied in this study as a space reduction technique that explains the variance of a set of variables through a projection onto linear combinations of these variables. Usually, the first few principal components, which explained the largest variation, are used for monitoring thus neglecting the principal components with small contributions. Accordingly, assessing the relative importance of the number of principal components to be selected for process monitoring performance is crucial for correct detection and classification.

3.2.2 Fault classification using Support Vector Machine

SVM is a supervised binary classification technique, which goal is to project the data onto linear hyperplanes to separate between pre-defined classes, given a dataset matrix $\mathbf{X}$ and a labels matrix $\mathbf{Y}$ (Cortes & Vapnik, 1995). Usually, the matrix $\mathbf{X}$ is represented by mxn, where m is the number of observations and n the number of variables. On the other hand, each row in the labels matrix **Y** is represented as either 1 or -1 to differentiate between the situation where the data belongs or does not belong to the class respectively. Given that SVM uses linear hyperplanes to differentiate between classes, we can write it as

$$\mathrm{f(x)} = \bar{\boldsymbol{w}} \cdot \bar{\boldsymbol{x}}_i + \boldsymbol{b} \tag{79}$$

Where $\bar{\boldsymbol{w}}$ and $\boldsymbol{b}$ are parameters that determine the position and orientation of the linear hyperplane. Furthermore, based on the values of the matrix $\mathbf{Y}$, SVM theory dictates that a canonical hyperplane is defined which permits defining the following constraint

$$\mathbf{y}_\mathrm{i}(\bar{\mathbf{w}} \cdot \bar{\mathbf{x}}_\mathrm{i} + \mathbf{b})-1 = 0 \tag{80}$$

Then, for the purpose of finding an optimal separation hyperplane, the distance between the two classes is represented as follows:

$$\rho(\bar{\mathbf{w}}, \mathbf{b}) = \frac{\mathbf{2}}{\|\bar{\mathbf{w}}\|} \tag{81}$$

After some mathematical manipulations it is possible to show that the optimal hyperplane that maximizes the distance between the two classes can be obtained by solving the following minimization problem subject to constraint (80):

$$\min \frac{1}{2} \|\bar{\mathbf{w}}\|^2 \tag{82}$$

Then, this constrained optimization problem can be formulated with Lagrange multipliers as follows:

$$\mathbf{l} = \frac{1}{2}\|\bar{\mathbf{w}}\|^2 - \sum \boldsymbol{\alpha_i}[\mathbf{y}_\mathrm{i}(\bar{\mathbf{w}} \cdot \bar{\mathbf{x}}_\mathrm{i} + \mathbf{b}) - 1] \tag{83}$$

Where α_i represent the Lagrange multipliers. Then, to find the optimal solution to the problem : the optimality conditions are as follows:

$$\frac{\partial \mathbf{l}}{\partial \mathbf{w}} = \bar{\mathbf{w}} - \sum \boldsymbol{\alpha_i}\mathbf{y}_\mathrm{i}\bar{\mathbf{x}}_\mathrm{i} = 0 \quad \therefore \quad \mathbf{w} = \sum_\mathrm{i}^\mathrm{m} \boldsymbol{\alpha_i}\mathbf{y}_\mathrm{i}\bar{\mathbf{x}}_\mathrm{i} \tag{84}$$

$$\frac{\partial \mathbf{l}}{\partial \mathbf{b}} = -\sum \boldsymbol{\alpha_i}\mathbf{y}_\mathrm{i} = 0 \quad \therefore \quad \sum_\mathrm{i}^\mathrm{m} \boldsymbol{\alpha_i}\mathbf{y}_\mathrm{i} = 0 \tag{85}$$

From (83), (84) and (85) it is possible to obtain:

$$\mathbf{l} = \sum_{\boldsymbol{k}}^\mathrm{m} \boldsymbol{\alpha_k} - \frac{1}{2}\sum_\mathrm{i}^\mathrm{m}\sum_\mathrm{j}^\mathrm{m} \boldsymbol{\alpha_i\alpha_j y_i y_j \bar{x}_i \cdot \bar{x}_j} \tag{86}$$

From this equation, it can be noticed that the solution to the optimization problem depends only on the dot product of pairs of samples $\boldsymbol{\bar{x}_i \cdot \bar{x}_j}$. On the other hand, depending on the problem under consideration, it may be impossible to linearly separate between classes. To address highly nonlinear problems, the theory of Kernel function was developed (Hofmann, Schölkopf, & Smola, 2008), that is based on the transformation of the feature vector into a higher dimensional space, which can be expressed as follows:

$$\mathrm{K}(\boldsymbol{\bar{x}_i}, \boldsymbol{\bar{x}_j}) = \emptyset(\boldsymbol{\bar{x}_i}) \cdot \emptyset(\boldsymbol{\bar{x}_j}) \tag{87}$$

Where ∅ is considered the feature mapping. Then, the separation hyperplane can be re-calculated using these Kernel transformations. Some of the most common Kernel functions are listed in table 9.

Table 9. Kernel Functions Expressions

Name	Expression
Linear	$\mathrm{K}(\boldsymbol{\bar{x}_i}, \boldsymbol{\bar{x}_j}) = \boldsymbol{\bar{x}_i \cdot \bar{x}_j}$
Polynomial	$\mathrm{K}(\boldsymbol{\bar{x}_i}, \boldsymbol{\bar{x}_j}) = (1 + \frac{\boldsymbol{\bar{x}_i \cdot \bar{x}_j}}{\sigma^2})^{\boldsymbol{n}}$
Radial Basis Function (RBF)	$\mathrm{K}(\boldsymbol{\bar{x}_i}, \boldsymbol{\bar{x}_j}) = \exp(-\frac{\|\bar{x}_i \cdot \bar{x}_j\|^2}{\sigma^2})$

Even though Support Vector Machines (SVM) where first designed as a binary classification technique, several studies have been able to extend the SVM properties to multiclass classification problems. To extend the SVM binary classification problem into a multiclass classification approach two different methods have been proposed: “one-against-all” or “one-against-one”. The “one-against-all” method is based on assembling k SVM models, where k is the number of classes. Then, the k classifiers are trained by using the information of each class against the information of all the other classes (L. Bottou et al., 1994). On the other hand, the “one-against-one” method is developed by assembling $k(k\text{-}1)/2$ classifiers, where each one of these classifiers is obtained by comparing the data between two different classes. After training all the classifiers, scores can be developed that define to which class each set of data belongs to (Ulrich, 1999). During this study the “one-against-one” strategy was applied since it is simpler to implement. In addition, there have been studies to investigate the efficacy of each of these methods and they have concluded that the “one-against-one” method is usually the better strategy (Hsu & Lin, 2002).

3.2.3 Fault diagnosis with Artificial Neural Networks

In this study the structure of the neural network is developed by dividing the training in two different parts: a stacked auto encoder neural network that will help compressing the data into a smaller space followed by softmax classifier layer which will provide the final classification result. It should be noticed that the purpose of the auto-encoder is twofold: i-compressing the raw data into a smaller space and ii- it provides weight values that are used as initial guesses for further refined training of the autoencoder+softmax structure. Since the training of the final network is based on a highly nonlinear optimization procedure, the use of good initial guesses as provided by the autoencoder are crucial for the success of the final training procedure. It has been reported in the literature, that auto encoders have been found to help to boost the classification accuracy as well as loss function (Geoffrey E. Hinton et al., 2006).

A stacked auto encoder is an unsupervised type of networks which consist of stacking several layers of neurons and forcing the output to replicate the input as best as possible as explained in the previous chapter. The data compression achieved by the autoencoder serves primarily to denoise the input data. As mentioned in the previous chapter the auto-encoder is composed of an encoding part and a decoding part. However, after training, the decoding part is left out while the encoding part, that is used to compress the data and denoise the inputs, is kept for further use in combination with a softmax based classification layer. In addition, in order to accelerate the training of the autoencoder and avoid the covariate shift problem mentioned in the previous chapter, a batch normalization is applied between the layers of the autoencoder.

In order to use the autoencoder for classification the decoder is removed and the auto-encoder is connected to a classification layer. After combining the auto-encoder with the classification layer, the weights of the encoder may be left constant from the previous training operation or they may be fine-tuned together with the classification layer. Usually, when solving classification problems, it is possible to apply sigmoidal or softmax functions as activation functions in the output layer. Softmax activation function are able to handle multiclass classification problems by outputting set of binary values, where each configuration in this binary code represents a certain class. In addition, softmax functions provide probability distributions that can inform us on the confidence level of the prediction. Training the classification model involves a supervised learning procedure, where for each data vector applied for training or testing a class is assigned thus allowing the neural network model to learn to detect different patterns corresponding to different classes.

3.3 Results and Discussions

3.3.1 Experiment and Data Setup

To demonstrate the effectiveness of the theory presented above, we generate training and test data from the Tennessee Eastman Process (TEP) simulation. The TEP control scheme used for the main part of the research is described as “Multiloop_mode3”. The data extracted from the TEP simulation has been extracted from the normal state and 20 different faults, which consists of 52 different variables generated with a sampling interval of 36 seconds. It should be noticed that during the testing of our methods the normal state was considered as a different class and hence 21 different classes are considered for diagnosis (classification). In addition, we have eliminated the effect of short initial transients by avoiding using the first hour of the simulation. Each different fault is run for 72 hours, generating 7,100 samples for each state. This data is divided between training and testing data, where we include the first 4,100 samples as training data and the rest for testing. This results in 21 different classes in total that sums up to 81,859 training samples and 59,179 testing data. We should point out that the discrepancy between these numbers is due to the fact that when implementing faults 6 and 8 the simulation became unstable after a period of time (~6 h) due to the inherent unstable behavior of the reactor in response to the onset of fault 6. The problems related to Fault 6 have been reported in previous studies and control overrides have been proposed to avoid the resulting instability (Larsson et al., 2001).

3.3.2 Static PCA-SVM vs Static ANN

In this section the results obtained for fault diagnosis using different statistical tools and ANN will be analyzed and compared. First, the results of Principal Component Analysis (PCA) are presented. Then, the relationship between the number of PCA loadings used and classification performance is discussed. Finally, the classification performance and the false alarm rates obtained with Support Vector Machine (SVM), PCA -SVM and Artificial Neural Networks (ANN) based on static data will be compared.

When applying PCA to the TEP dataset we combine all the training data of the different disturbances into one data matrix X composed of $\mathrm{n}=86{,}100$ observations and $\mathrm{p}=53$ random variables. PCA will compute a set of linear combinations k, where $p > k$, that explains the highest % of the total variation in the observations. Figure 15 shows the relationship between the number of PCA loadings and the variance explained in the dataset. As it can be observed in Figure 15, as we increase the number of principal components that are considered in the PCA model the variance captured by the model increases. However, ince the TEP simulation contains a certain % of noise in all the measurements and we are interested in filtering the noise from the data, the number of principal components in the final PCA model should be selected so as it results in improvements in classification rates. Hence, in order to determine the number of principal components to be used, we compared the classification results obtained from different combinations of principal components with SVM. Figure 16 shows the comparison between the classification rates resulting from models based on SVM only and on the combination of SVM with PCA. The comparison of explained variance among the models with different number of principal components is shown in Figure 15. It is evident from this figure that beyond 35 principal components there is no further improvement towards explaining the variance in the data. For example, considering PCA models with 15, 25 and 35 principal components resulted in the explanation of 83%, 94% and 98.6% of the total variance respectively. Figure 16 indicates that for almost all the faults, we obtain higher classification rates when 98.6% of the dataset variances is explained by 35 principal components (linear relationships). Figure 16 also shows that the results obtained from the combination of PCA and SVM provides significantly better classification as compared to a model that is based solely on SVM. Hence, it is concluded that the ability of PCA to denoise the raw data through data compression is very useful. Therefore, in the following sections of this chapter a model that combines PCA with SVM is used and 35 principal components are considered for the PCA component of the model. Figure 17 shows the false alarm rates for each different class using the previous methods. Consistent with the classification results, the model that is solely based on SVM results in higher false alarm rates for many of the faults as compared to the rates obtained with the combines PCA+SVM model.

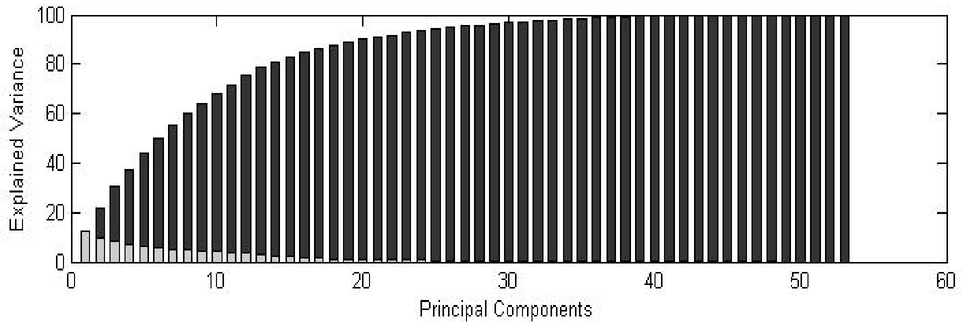


Figure 15. Illustration of PCA loadings and explained variance. (white) Represents the explains variation by each loading. (black) Accumulation of variance explained.

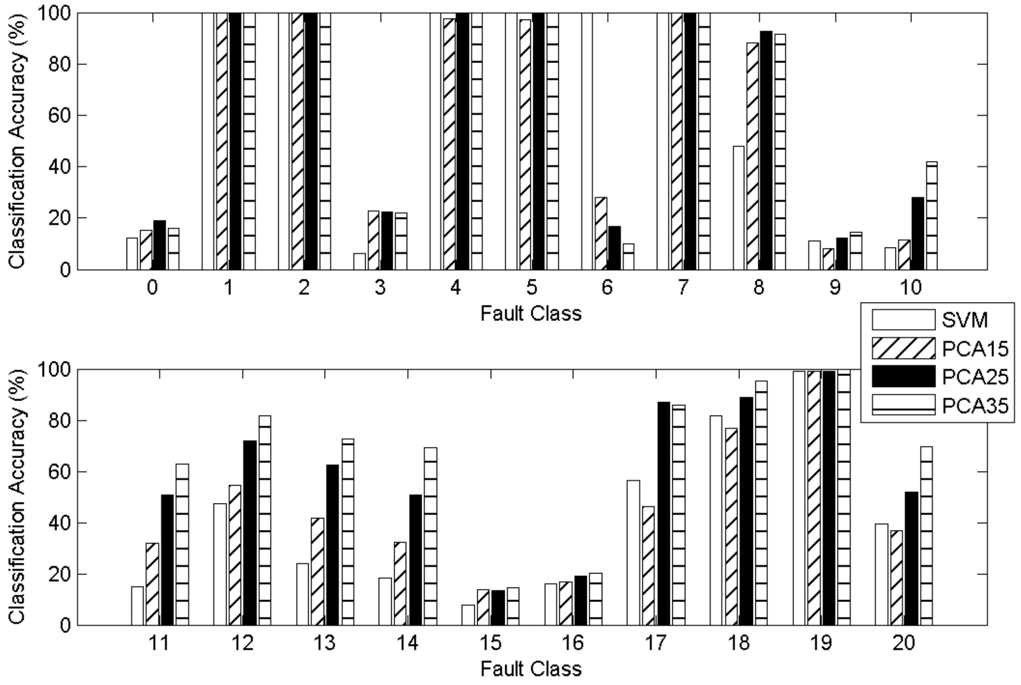


Figure 16. Classification accuracy obtained in TEP depending on the number of loadings

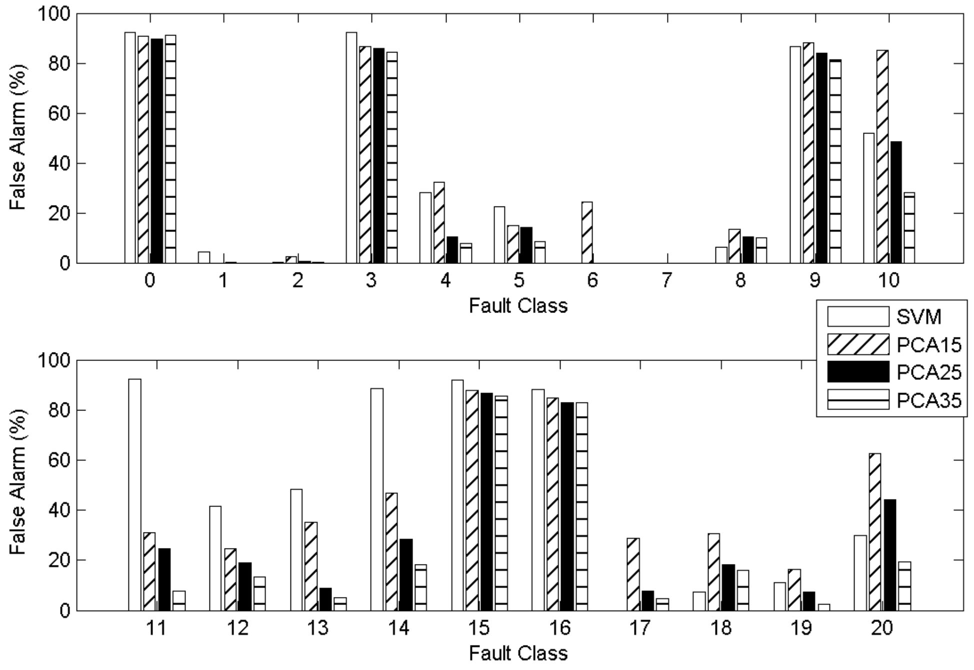


Figure 17. False alarm rates obtained in TEP depending on the number of loadings

Subsequently an ANN model composed of an auto-encoder and a softmax layer was developed and compared to the model based on SVM and to the model based on the combination of PCA and SVM. As explained above the autoencoder was learned first through an unsupervised learning procedure and the resulting weight values were used as initial guesses for re-training the weights of the final model that consists of the autoencoder connected to the softmax classification layer.

Figure 18 shows the classification rates obtained by different statistical approaches for fault diagnosis, as well as for the ANN. From this figure, it can be observed that the ANN is able to provide better classification results than either the SVM or the combined PCA+SVM models. Good classification performance is obtained in most of the observable faults, with the exception of fault 16. However, for the faults (0,3,9,15) that are reported as difficult to observe in previous studies it is possible to observe that also with the techniques developed here the results are still poor. In the average, the classification accuracies are 49.7%, 67.3% and 73.8% for SVM, PCA+SVM and ANN respectively. Figure 19 shows the false alarm rates obtained with the different models. From this figure it can be observed that the false alarm rates have a consistent trend with the classification rates' results. Accordingly, the false alarm rates are particularly high for the difficult to observe faults. However, we can observe that the false alarm rates obtained with the ANN model are still significantly lower (better) that with the SVM, and SVM+PCA models.

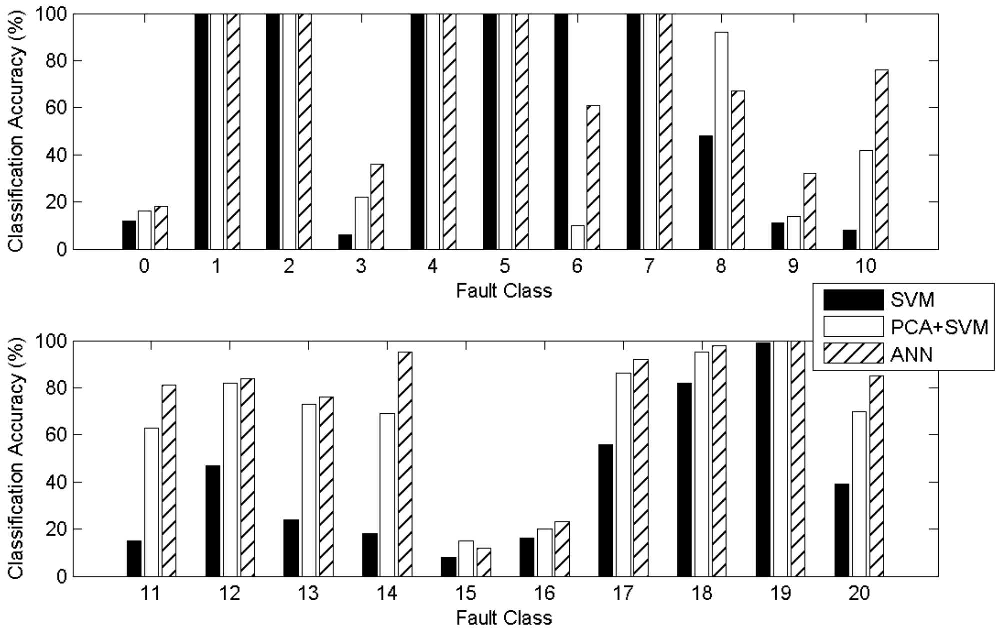


Figure 18. Comparison of the classification accuracy of SVM, PCA+SVM and ANN obtained in TEP

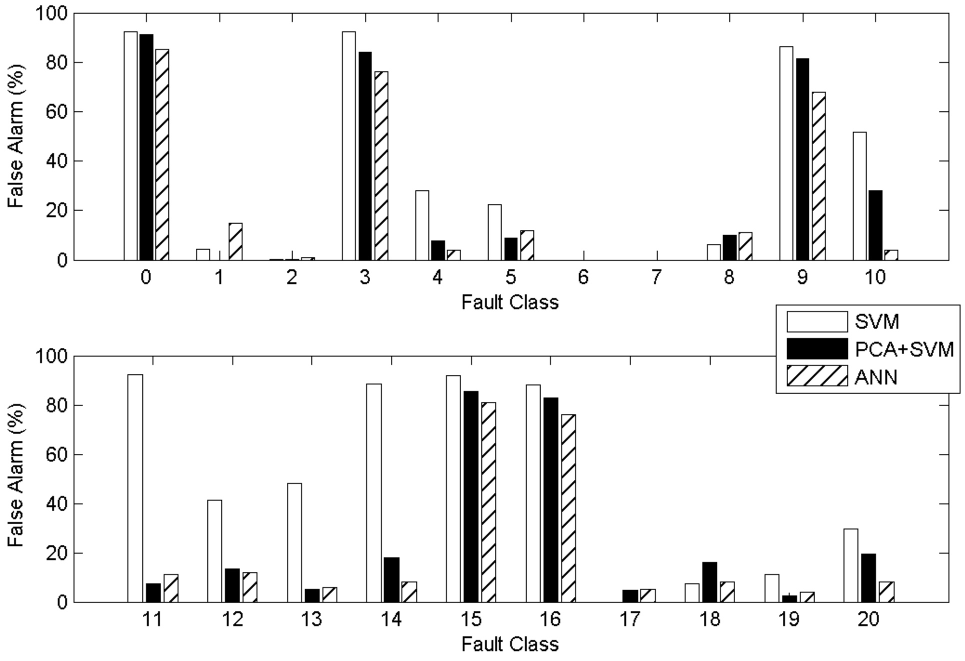


Figure 19. Comparison of the false alarm rated of SVM, PCA+SVM and ANN obtained in TEP

3.3.3 Hierarchal PCA-SVM vs Hierarchal ANN

To address the low classification rates obtained for the difficult-to-observe faults a hierarchal strategy is proposed here. First, a rationale for the use of a hierarchal structure will be provided. Then, the results with and without the hierarchical structure for all the proposed models will be compared, ie. SVM, PCA+SVM and ANN.

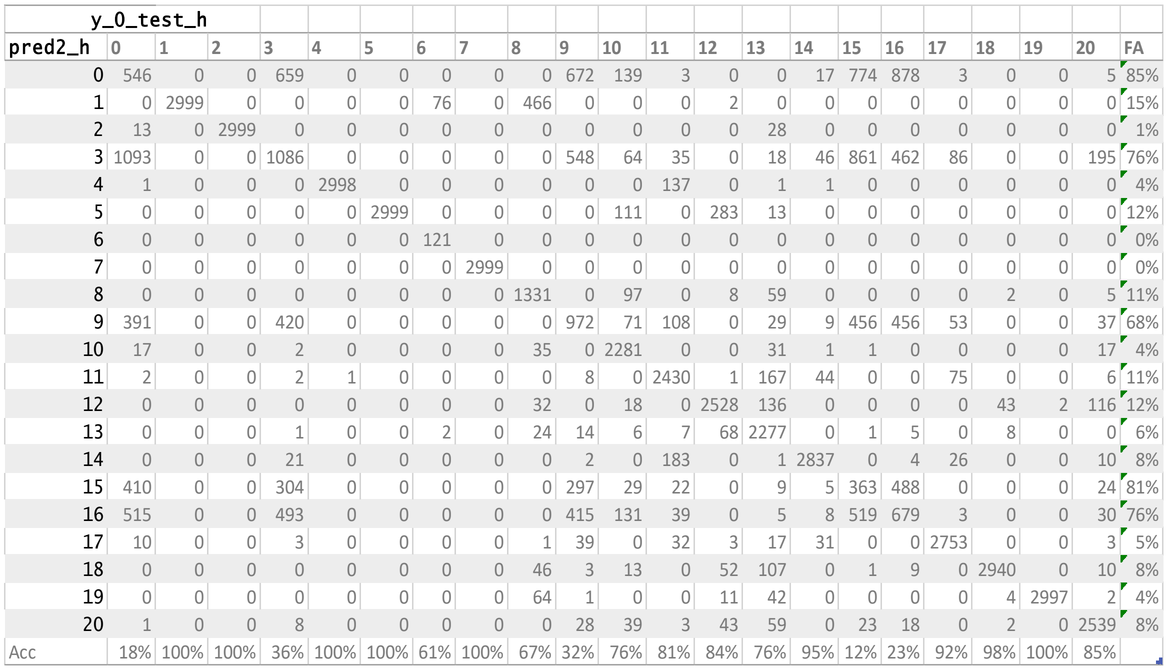
The rationale behind this hierarchical structure derives from the confusion matrix of the classification results of the previous section illustrated in table 10. Table 10 indicates that the classification rates are good for almost every observable fault. However, when we focus on the classification performance of the difficult-to-observe faults we find that the neural network is not able to discern between the information of these faults. This is a common problem when we are dealing with data sets corresponding to different faults that is too similar one to another. This similarity result in large miss-classification of faults as also corroborated by the results in the confusion matrix.

The proposed hierarchical structure addresses the similarities of data sets corresponding to different faults by focusing the classification model on the smaller group of faults that are difficult to classify. This is somewhat analogous to the use of weighted least squares based models where part of the outputs are weighted preferentially with respect to others. Similar preferential weighting could have been done with a single one ANN based model but isolating the problem by finding separate models for the easier to identify faults and for the difficult-to-identify faults is a simpler, easier to calibrate approach. Accordingly, we separate the data related to the difficult-to-identify faults and these data is then normalized anew based on the range of values for each variable obtained for these faults. This re-normalization is able to increase the variance of the data corresponding to the difficult-to-identify faults as compared to their relative variance when these data were embedded together with the rest of the faults, i.e. with the easier faults. This re-normalization, enhances the fault classification. After normalizing the data, another ANN, SVM and PCA+SVM models are trained for the few faults included in the class of the difficult-to-observe faults. The proposed hierarchical structure for separate identification of easier and difficult to identify faults is shown in Figure 20.

Figure 21 compares the classification rates for the different faults as obtained with the SVM, PCA+SVM and ANN models. Figure 21 shows that the relative superiority of the

ANN classification algorithm with respect to the SVM and PCA+SVM algorithms is still maintained. On the other hand, it is evident from Figure 21 that the classification accuracy for the difficult to observe faults when using the hierarchical structure improves as compared to the models that do not use this structure. This can be further seen in Figure 22, where we compare the classification rates obtained with ANN applying a simple all-to-all structure and the hierarchical structure. When we compare the overall average classification rate of the hierarchical structure to the original non-hierarchical model that consider all the faults together we can observe a slight improvement of 2.2% in the average classification rate: 76.0% with the hierarchical structure versus 73.8% without the hierarchical structure.

Table 10. Confusion matrix of the classification results for TEP for an all-to-all structure.



y_0_test_h																					
pred2_h	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	FA
0	546	0	0	659	0	0	0	0	0	672	139	3	0	0	17	774	878	3	0	0	5	85%
1	0	2999	0	0	0	0	76	0	466	0	0	0	2	0	0	0	0	0	0	0	0	15%
2	13	0	2999	0	0	0	0	0	0	0	0	0	0	28	0	0	0	0	0	0	0	1%
3	1093	0	0	1086	0	0	0	0	0	548	64	35	0	18	46	861	462	86	0	0	195	76%
4	1	0	0	0	2998	0	0	0	0	0	0	137	0	1	1	0	0	0	0	0	0	4%
5	0	0	0	0	0	2999	0	0	0	0	111	0	283	13	0	0	0	0	0	0	0	12%
6	0	0	0	0	0	0	121	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0%
7	0	0	0	0	0	0	0	2999	0	0	0	0	0	0	0	0	0	0	0	0	0	0%
8	0	0	0	0	0	0	0	0	1331	0	97	0	8	59	0	0	0	0	2	0	5	11%
9	391	0	0	420	0	0	0	0	0	972	71	108	0	29	9	456	456	53	0	0	37	68%
10	17	0	0	2	0	0	0	0	35	0	2281	0	0	31	1	1	0	0	0	0	17	4%
11	2	0	0	2	1	0	0	0	0	8	0	2430	1	167	44	0	0	75	0	0	6	11%
12	0	0	0	0	0	0	0	0	32	0	18	0	2528	136	0	0	0	0	43	2	116	12%
13	0	0	0	1	0	0	2	0	24	14	6	7	68	2277	0	1	5	0	8	0	0	6%
14	0	0	0	21	0	0	0	0	0	2	0	183	0	1	2837	0	4	26	0	0	10	8%
15	410	0	0	304	0	0	0	0	0	297	29	22	0	9	5	363	488	0	0	0	24	81%
16	515	0	0	493	0	0	0	0	0	415	131	39	0	5	8	519	679	3	0	0	30	76%
17	10	0	0	3	0	0	0	0	1	39	0	32	3	17	31	0	0	2753	0	0	3	5%
18	0	0	0	0	0	0	0	0	46	3	13	0	52	107	0	1	9	0	2940	0	10	8%
19	0	0	0	0	0	0	0	0	64	1	0	0	11	42	0	0	0	0	4	2997	2	4%
20	1	0	0	8	0	0	0	0	0	28	39	3	43	59	0	23	18	0	2	0	2539	8%
Acc	18%	100%	100%	36%	100%	100%	61%	100%	67%	32%	76%	81%	84%	76%	95%	12%	23%	92%	98%	100%	85%	

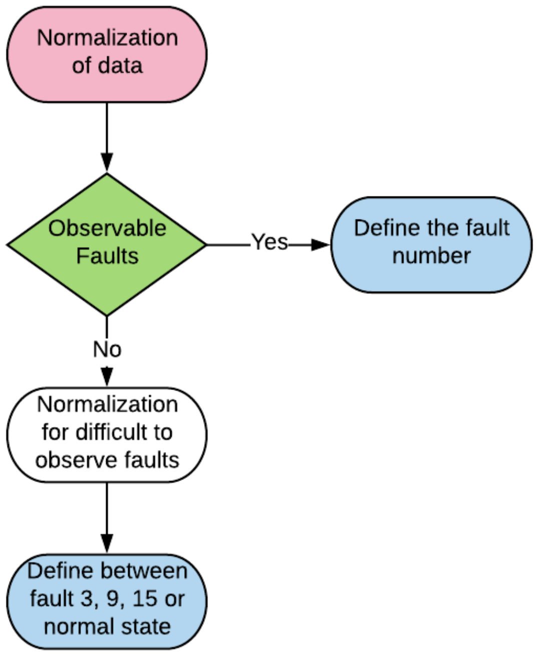

Normalization of data
Observable Faults
Yes
Define the fault number
No
Normalization for difficult to observe faults
Define between fault 3, 9, 15 or normal state

Figure 20. Hierarchical structure for partitioning of classes in the TEP

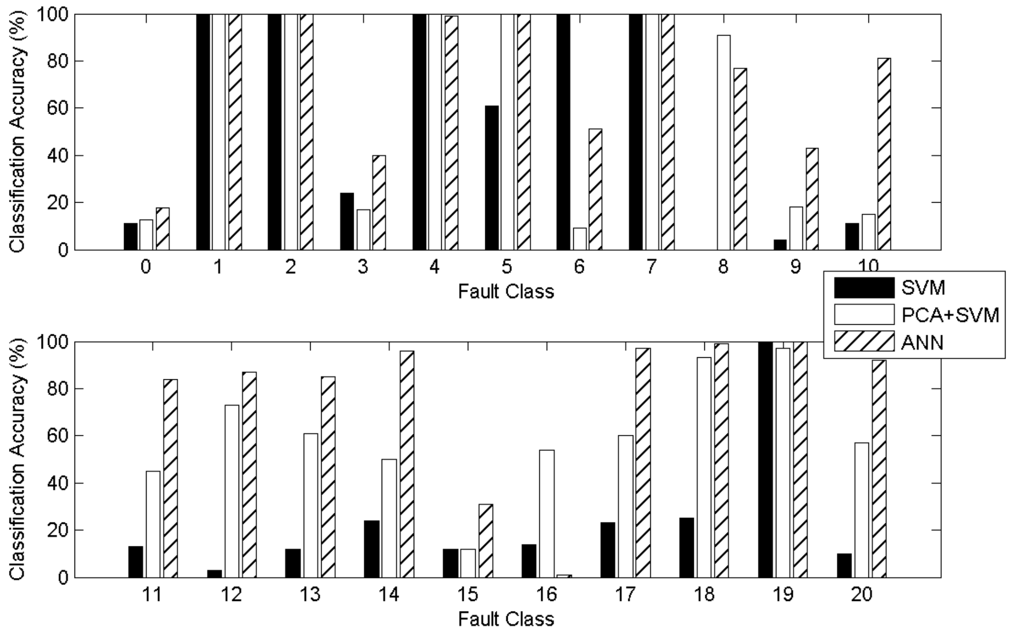


Figure 21. Comparison of the classification accuracy of SVM, PCA+SVM and ANN using hierarchical structure

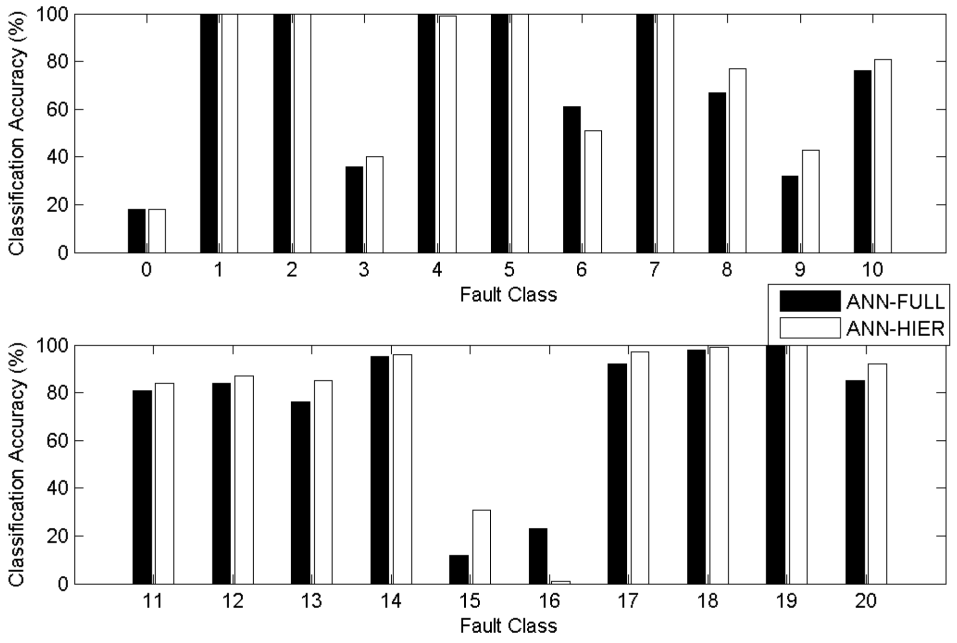


Figure 22. Comparison between the classification accuracy of ANN using an all-to-all structure and hierarchical structure

3.3.4 Comparison between different modes of Tennessee Eastman Simulation

In this section we analyze and compare the results obtained for fault classification using the three different control strategies contained in the Tennessee Eastman Simulation. In order to conduct a fair comparison between the different modes the same network structure was applied to the 3 modes of operation. Comparing the classification rates for the 3 modes is important for assessing the robustness of the algorithm to cope with changes in operating conditions since the modes involve different conditions. Both classification and false alarm rates were calculated

In order to be consistent with the previous study the same data extraction and data treatment as before were applied for this study. The results obtained for the classification and false alarm rates of the different TEP modes are shown in Figure 23 and Figure 24 respectively. Even though, the operating conditions may be different and the control schemes may change between the different modes similar trends are observed to the ones observed in the previous section that considered only one mode of operation. Accordingly, the classification rates for the easier to identify faults are still very high, while the faults that were difficult to identify in the original model are still difficult to discern in the new two modes. The average classification accuracies are: 73.8%, 71.1%, and 76.6% for mode3, mode 1 and mode1Skoge respectively.

An important point to consider when comparing mode3 to mode1 and mode1Skoge, is that there is a high level of confusion between the normal state and the fault 16 thus resulting in classification rates of either 0 or 16 to be 0%. Carefully inspection of the data associated to these two classes indicated a high level of similarity of the data associated to fault 16 and the normal state. A thorough dynamic investigation using different dynamic scenarios was conducted in Simulink but we were unsuccessful in finding a scenario that allowed distinguishing between fault 16 and the normal state. This led to the conclusion that in the simulation files for mode1 and mode1Skoge fault 16 is not active.

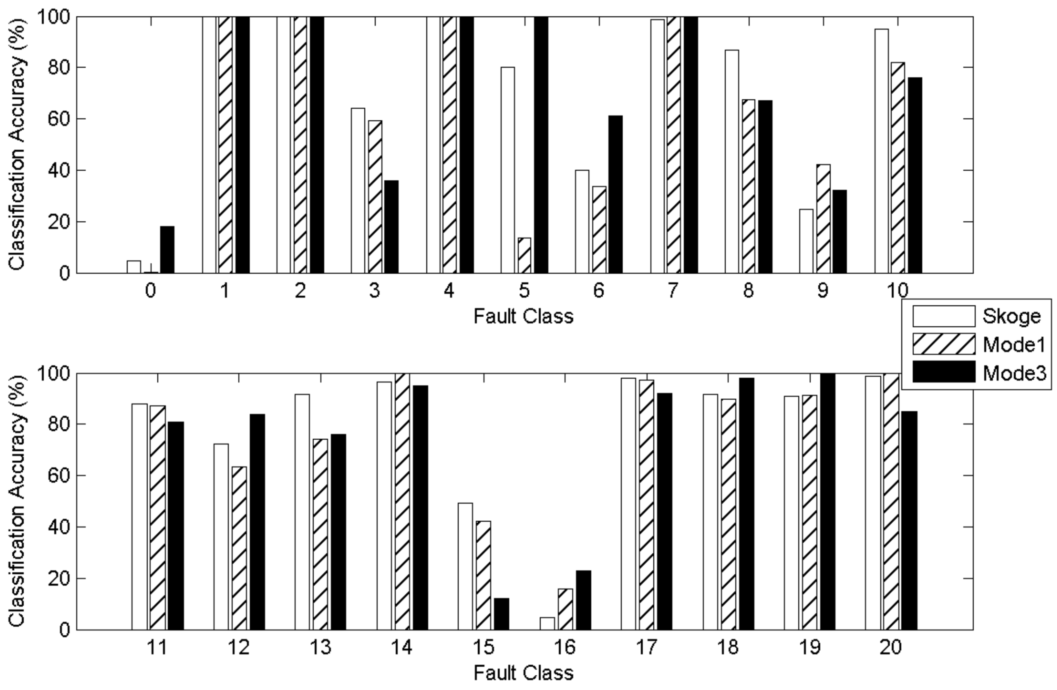


Figure 23. Comparison between the classification accuracy of ANN using the different control schemes

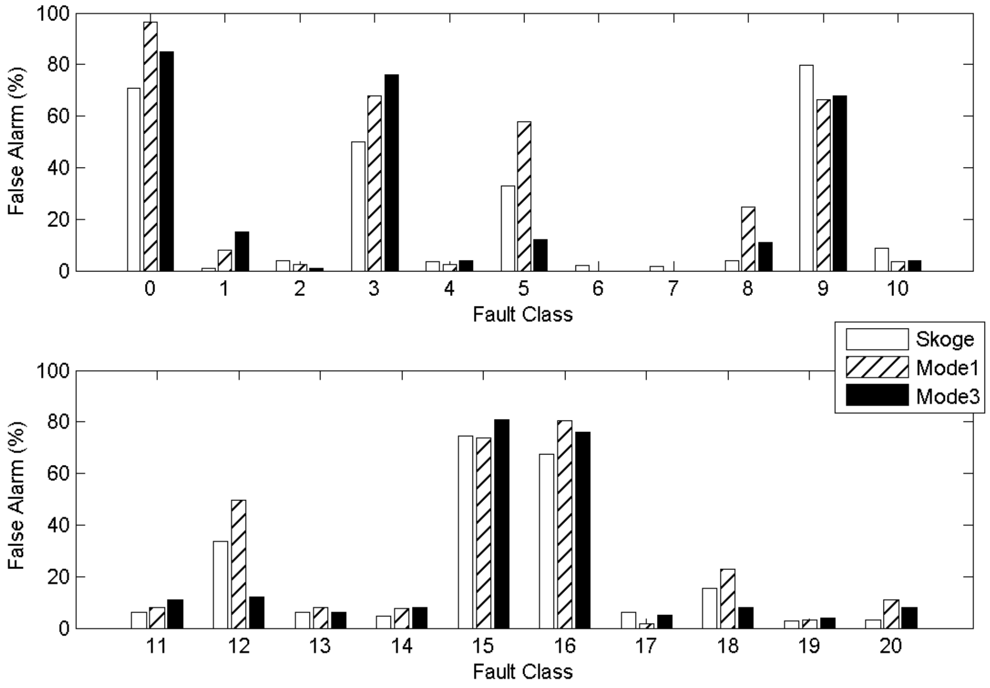


Figure 24. Comparison between the false alarm rates of ANN using the different control schemes

3.4 Conclusions of Chapter 3

In this chapter we compare the accuracy of deep learning models (ANN) to statistical models (SVM and PCA+SVM) to classify faults based on static information. It was to clearly demonstrated in the presented studies that neural networks provide better performance that the statistical models and they are a powerful tool for fault classification, due to their abilities to capture nonlinear behavior. The overall classification average before the hierarchical structure was implemented for SVM was 49%, for PCA was 67.3% and ANN 73.8%. In addition, a hierarchical structure was developed whereby splitting faults with similar characteristics and then reclassifying them results in good improvements in the classification accuracy. In the case of ANN, the improvement of using a hierarchical structure, compared to an all-to-all structure, was of +3% in average classification accuracy. Furthermore, we applied this same study to several different control scheme modes available for the Tennessee Eastman Simulator. These additional studies show similar results to those obtained with the original operating mode. Thus, good classification is obtained for all easy to observe faults and the performance is still poor for the difficult to observe faults. The average classification accuracies obtained with the hierarchical structure are similar for the 3 modes of operation: 73.8%, 71.1%, and 76.6% for mode3, mode1, and mode1Skoge respectively.

Chapter 4

Fault Detection and Classification using Dynamic Data

4.1 Overview

In this chapter, statistical monitoring models are generated based on dynamic data of the Tennessee Eastman Process for fault detection and classification. The key difference between this chapter and the previous one is that here current and past data is considered together as inputs to the models and thus dynamic correlations in the data are considered. This chapter covers two principal detection and diagnosis models that are specifically suited for capturing dynamic correlations: Dynamic Principal Component Analysis (DPCA) and Recurrent Neural Networks (RNN). For the purpose of fault diagnosis with DPCA a DPCA compression model combined with a Support Vector Machine (SVM) based model is used to differentiate between different faults. Although this model combines DPCA and SVM, for simplicity it is referred in this chapter as the DPCA model. On the other hand, the RNN model is a particular dynamic neural network structure that captures dynamic correlations in the data. Similar to the previous chapter, a hierarchical scheme is developed to improve the classification results by developing separate models for the easier and more difficult to observe faults. In addition, for specific faults that are difficult to observe due to low signal to noise ratio a novel approach using a pseudorandom signal is implemented in order to enhance the classification rates. This structure is shown to increase the classification accuracy of both easier to observe and difficult to observe faults. To test the robustness of the proposed classification models, we will compare them for different operating modes available for the TEP simulator.

4.2 Definitions and Methods

During this section, the concepts of number of samples and number of lags could be considered as synonyms. However, in the case of number of lags, to be consistent with notations used in other DPCA studies it is used to describe the number of lagged vectors of variables stacked in the observation matrix used as input to a DPCA model. Similarly, in the case of number of samples, it will be used in RNN models to describe the number of continuous observations used for the transformation of the data matrix into a 3 dimensional version.

4.2.1 Dynamic Principal Component Analysis (DPCA) – SVM

In order to understand the rationale for using dynamic data to enhance fault classification, the main differences in information contained in static and dynamic data are explained. On one hand, static systems are only able detect correlations between variables at a particular time instance and to relate these instant correlated data to the status of the fault at that instance. On the other hand, dynamic systems will detect correlation between variables as well as correlations with past values. It is evident that upon occurrence of the fault the measured variables do not change significantly as compared to the noise levels, then the dynamic fault will not be identified until the differences in the signal become significant as compared to the noise. In addition two faults may have similar static correlations but it is likely that they will exhibit exactly the same dynamic correlations. Furthermore, highly nonlinear processes are expected to exhibit different dynamic behavior around different operating variables. For all these reasons, the consideration of dynamic data is expected to enhance fault classification. Dynamic Principal Component Analysis (DPCA) was developed as an extension of the PCA monitoring method to take into account dynamical correlations between variables in the observation matrix (Chiang et al., 2001; Ku et al., 1995).

Accordingly, the observation matrix is extended by stacking lagged information into a matrix that it is also referred to as a Hankel matrix. Mathematically, the lagged observation matrix is defined as follows:

$$\mathbf{X}(\boldsymbol{l}) = \begin{bmatrix} \boldsymbol{x_t^T} & \boldsymbol{x_{t-1}^T} & \cdots & \boldsymbol{x_{t-l}^T} \\ \boldsymbol{x_{t-1}^T} & \boldsymbol{x_{t-2}^T} & \cdots & \boldsymbol{x_{t-l-1}^T} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{x_{t+l-n}^T} & \boldsymbol{x_{t+l-n-1}^T} & \cdots & \boldsymbol{x_{t-n}^T} \end{bmatrix} \tag{88}$$

where $\mathrm{x_t^T}$ is defined as the observation vector in time instance t, l is the lag number and n is the number of observations. In principle, the subsequent procedure of data compression by DPCA is the same as for PCA. Thus, after stacking the observation matrix with lagged information vectors a decomposition into principal components is effected followed by the projection of the data onto a lower dimensional space. Following this compression step a multivariate autoregressive (ARX) model based on the compressed data can be established (Ku et al., 1995). On the other hand a DRNN (Deep Recurrent Neural Network) model is investigated that is based on the same amount of past information as determined by the number of lags considered for the DPCA models. The description of the DRNN model is detailed in the following section.

4.2.2 Fault Diagnosis with Deep Recurrent Neural Networks (DRNN)

Although as shown in previous studies and in the previous chapter neural networks are a powerful tool for detection and classification of faults they may still lacking accuracy when the data used for detection is not informative enough. For example, static data used in the previous chapter may not be sufficient for accurate classification, especially for the difficult to observe faults, since they ignored dynamic correlations among the inputs. Therefore, it is important to understand that the consideration of dynamic instead of static information only may result in improved classification performance. Consequently, to study the effect of considering dynamic data on classification rate, we formulated recurrent neural networks, a tool for modelling nonlinear dynamic processes.

The general structure of the dynamic neural network model is somewhat similar to the one used for the fault classification problem with static data. However, when using dynamic data, the basic neuron has to be capable of receiving and correlating data collected at different time intervals. The two most common neural network units to regress dynamic data are: Long-Short Term Memory (LSTM) cells and Gated Recurrent Units (GRU). Thus, the basic model structure used for static data in the previous chapter where the network is based on the connection of a stacked auto encoder and a softmax classifier remains the same but the basic neural units of the auto-encoder are now modified to account for dynamic inputs.

Thus, in the case, the stacked auto encoder will consist of several layers of recurrent units stacked together. It is important to denote that we applied batch normalization to the transformed inputs after each inner layer of the auto-encoder, in order to decrease the learning time of the training algorithms. The data employed in this current study is the same data as used in the previous chapter. However, in the current case instead of using a data vector with the instantaneous measured variables, current and past data are organized into a matrix to be inputted through the recurrent network. It is important to denote that there are two ways to structure the context, the middle layer, in an auto encoder structure for recurrent networks. The first option is to structure the context as a vector, where the data matrix is transformed by the encoder into a context vector that will contain the important information and then transform it back into a matrix for presentation to the decoder. The second option is by applying the context as a matrix, where we only reduce the variable size, without reducing the dimension of the matrix. Through several testing we decided that the second option was the more viable one since the lost information due to the dimension reduction was detrimental to the accuracy. As before, after training of the auto-encoder-decoder, the decoder part is ignored and the remaining encoder part is connected to a soft-max based classification layer. As before to accelerate the training of the autoencoder and to avoid the covariate shift problem batch normalization is applied to the intermediate inputs in between the recurrent layers of the autoencoder.

In contrast with the case that used static data in the current case since the auto encoder is built to return a matrix as its output we need to implement another layer that will transform this output matrix into a vector for further use in the classification layer. This intermediate layer necessary to transform the output matrix into a vector is also based recurrent neural units. The classification model that is made of the combination of the auto-encoder connected to a softmax classification layer is trained by a supervised learning procedure where for each data vector applied for training or testing a particular class is provided. Following this supervised learning procedure, the network learns to differentiate dynamic patterns corresponding to the different classes. As in the previous chapter we can also here keep the weights obtained from trained of the auto-encoder or we can fine tune these weights in the supervised learning procedure that is used to train the classification model, i.e. the model made of the auto-encoder connected to the softmax classification layer.

4.2.3 Pseudo Random Signal for Fault Diagnosis

Classic fault detection theories establish as undetectable faults those that result in measurable signals whose amplitudes are smaller than the noise magnitude along the quantification interval. In the case of the Tennessee Eastman several faults have been considered difficult to detect due to their inherent small signal to noise ratio. Therefore, in some cases we might need to induce signals with bigger amplitude in order to be able to detect these faults. The problem of undetectable faults is particularly acute in the TEP problem since several fault must be inferred from variables that are part of closed loop control loop. Variables that are controlled in closed loop inherently exhibit small variation thus making it difficult to estimate occurrence of faults from such variables. For example, when certain variable in the system approximately reach steady state only small signal changes may be observed which may be of the order of the noise. Thus, in order to create signals with larger amplitude that will be more informative about the occurrence of a fault, we rely on process-identification methods for fault detection (Isermann, 2006). Following system identification theory, it is known that by applying external input changes at certain point of the process it is possible to obtain information about the behavior of the process (Ljung, 1987). Therefore, with the goal of creating fault related signals with larger amplitudes we propose injecting excitation signals at different points of control loops, e.g. adding an excitation signal to the set-points, especially into the loops that involve variables related to the difficult to detect faults.

4.3 Results and Discussions

As in the previous chapter, we continue to differentiate between two types of faults: observable and difficult to observe. Observable faults will be considered fault that have discernible symptoms and have been well classified with conventional fault detection or diagnosis techniques. On the other hand, difficult to observe faults are those that generate changes in measured variables that are similar for different faults and thus are difficult to classify with conventional statistics techniques. It is important to emphasize that all the model used for classification in this section are able to capture dynamic correlation among the data.

4.3.1 Experiment and Data Setup

To demonstrate the effectiveness of the proposed methods, we generate training and test data from the Tennessee Eastman Process (TEP) simulation. As before, the TEP control scheme used for the initial phase of the current case study the operating mode is the one referred in the TEP simulator as “Multiloop_mode3”. The data extracted from the TEP simulation has been extracted for operation at the normal state and for 20 different faults. At each sampling interval 52 different variables are assumed to be measured. Each vector of measurements can be acquired with a sampling interval of 36 seconds. It should be noticed that during the testing of our methods we used the normal state as a different class and hence 21 different classes are considered for classification. In addition, we have eliminated the effect of short initial transients by avoiding using the first hour of the simulation. Each different fault occurrence of the simulation is run for 72 hours, generating 7,100 samples for each fault. The data is then divided between training and testing data sets, where the first 4,100 samples are used as training data and the rest I used for testing. This results in 21 different classes in total that sums up to 81,859 training samples and 59,179 testing data. It is important to consider that the number of training and testing samples may slightly vary depending on the data transformation needed for each model used here for classification, i.e. DPCA or RNN.

For the DPCA the data matrix is constructed by concatenating the observation vectors with the respective lagged observations where different number of lags were considered for comparison in the case study. After developing the lagged observation matrix, we search for dynamical correlations in the data through DPCA, which functions as a dynamical feature extraction method. Then, the resulting features are fed to an SVM for fault classification. On the other hand, in order to feed the information through a RNN model the observation vector is transformed into a 3D data matrix, instead of a 2D matrix. This data transformation from a 2D, defined as batch size and variables, to a 3D structure is done by adding the time samples as another dimension. Thus, the transformed data matrix has 3 dimensions defined as: batch size, number of time samples and variables. Figure 25 shows an illustration of this data transformation.

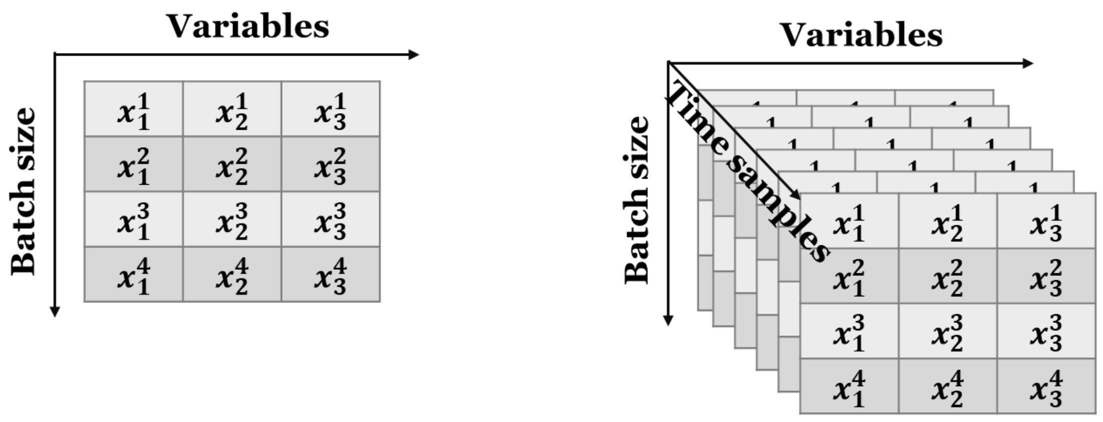


Figure 25. Illustration of data transformation for RNN.

4.3.2 Dynamic PCA vs Recurrent Neural Network

In this section we analyze and compare results obtained for fault diagnosis using the DPCA and RNN based models. First, we show and analyze the results obtained for the classification performance and the false alarm rates obtained with the Recurrent Neural Networks (RNN) for dynamical data for the three modes of the simulation available. In addition, we study the effect of increasing the number of time samples for all different modes in the Tennessee Eastman simulation based on the hypothesis that using an increasing number of lags will enhance classification accuracy. The results of the RNN are then compared with the classification results obtained with the Dynamic Principal Component Analysis (DPCA) based model. In addition we compare the classification performance with two main gated recurrent units LSTM and GRU.

, For comparing the RNN and DPCA classification models we first defined the number of samples we will be using for both studies. In the case of DPCA the number of lags used in the observation matrix is the key parameter. We compare the models resulting that use 5, 15 and 25 lags. It is important to understand that DPCA is a data reduction technique usually used for fault detection. However, in order to make it into a classification problem the output features from the DPCA model are fed into an SVM model that is used for final classification. Figure 26 shows the classification accuracies of DPCA combined with SVM for 5, 15 and 25 lags. From this figure it is found that with 5 and 15 lags similar classification results are obtained. However, with 25 lags the classification accuracy decreases. A possible explanation for this decrease is that as the number of lags increases the data exhibit increasing variability due to noise and nonlinearity that cannot be properly captured by the linear DPCA based extraction model This is somewhat comparable with the training of autoregressive models that as the number of lagged basis functions become larger the model becomes more prone to overfitting of noise and therefore it has lower predictive accuracy.

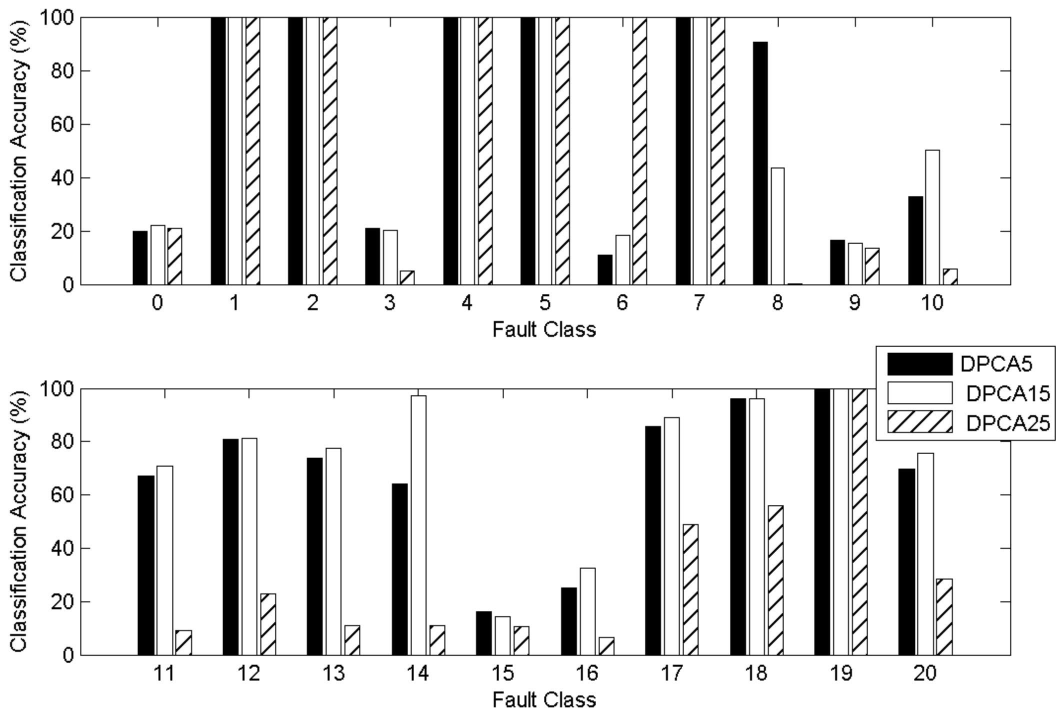


Figure 26. Classification accuracy obtained with DPCA in TEP depending on the number of lags.

Therefore, in order to further compare the results of DPCA with RNN, we choose the best model which in this case was obtained with 15 lags. Figure 27 shows the classification rates obtained by the DPCA model and the RNN that uses LSTM neural units. From this figure, we observe that the RNN provides better classification than DPCA. The RNN provides consistent better classification than the DPCA model for all the faults that are considered to be easier to observe. Furthermore, the RNN is also able to outmatch DPCA in classification accuracy for all the difficult-to-observe faults (0,3,9,15). However, in the case of these faults the results are still poor despite the fact that richer dynamic information is used. Figure 28 shows the false alarm rates obtained by the two models The trends observed form this figure are consistent with the classification rates where RNN provide lower false alarm rates for most faults but false alarm rates are still high with both methods for the difficult to observe faults.

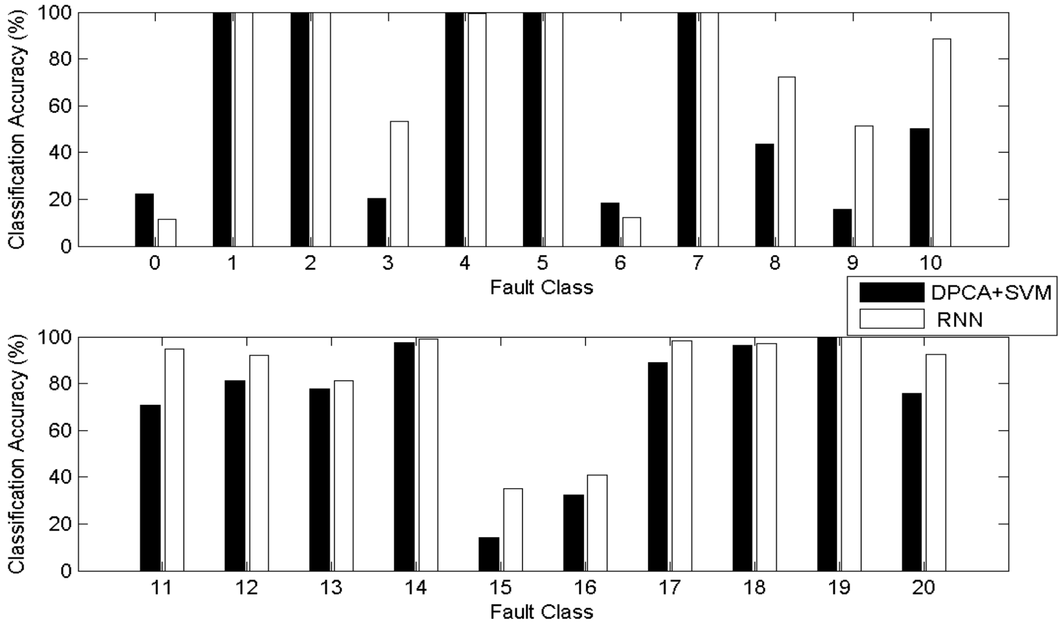


Figure 27. Comparison of classification accuracy of DPCA+SVM and RRN in TEP.

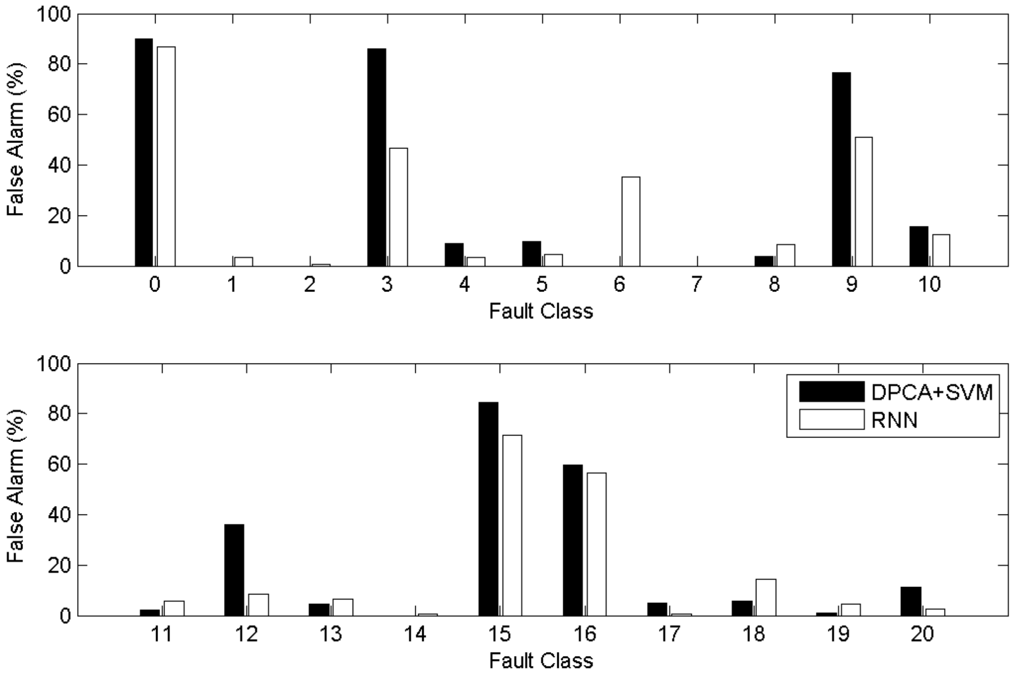


Figure 28. Comparison of false alarm rates of DPCA+SVM and RRN in TEP.

In order to improve the classification accuracy with RNN the consideration of a longer time horizon of data into the model was investigated. It was hypothesized that by including additional past information, the nonlinear RNN model can better capture the nonlinear behavior of the process while not increasing the overfitting of noise in the data Consequently, we study the effect of the number of samples presented to the RNN. Figure 29 shows the different classification accuracies obtained with models that assumed different number of samples. We can observe that by increasing the number of samples we are able to increase the classification accuracies of some faults. In particular, we can observe a drastic increase in the classification accuracy of fault 16 possibly indicating the ability of the network to capture the nonlinear dynamic behavior associated to this fault. In addition, the effect of increasing the number of samples can be observed by analyzing the average classification accuracy for the different number of samples, where we obtained 79.8%, 80.3%, 81% and 84% for the RNN with 5, 15, 25 and 100 number of samples respectively. On the other hand, for the difficult to observe faults the classification accuracy for any of these faults was still low. In addition, from Figure 30 which shows the false alarm rates, it can be observe very high false alarm rates for the difficult to observe faults.

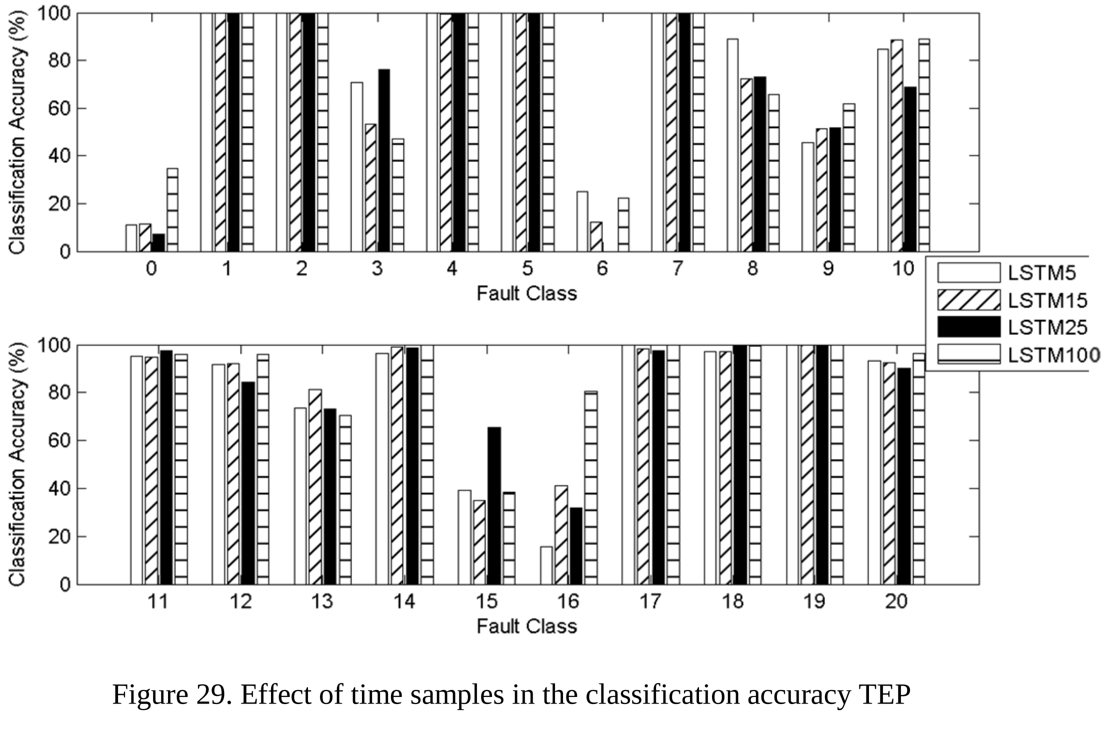


Figure 29. Effect of time samples in the classification accuracy TEP

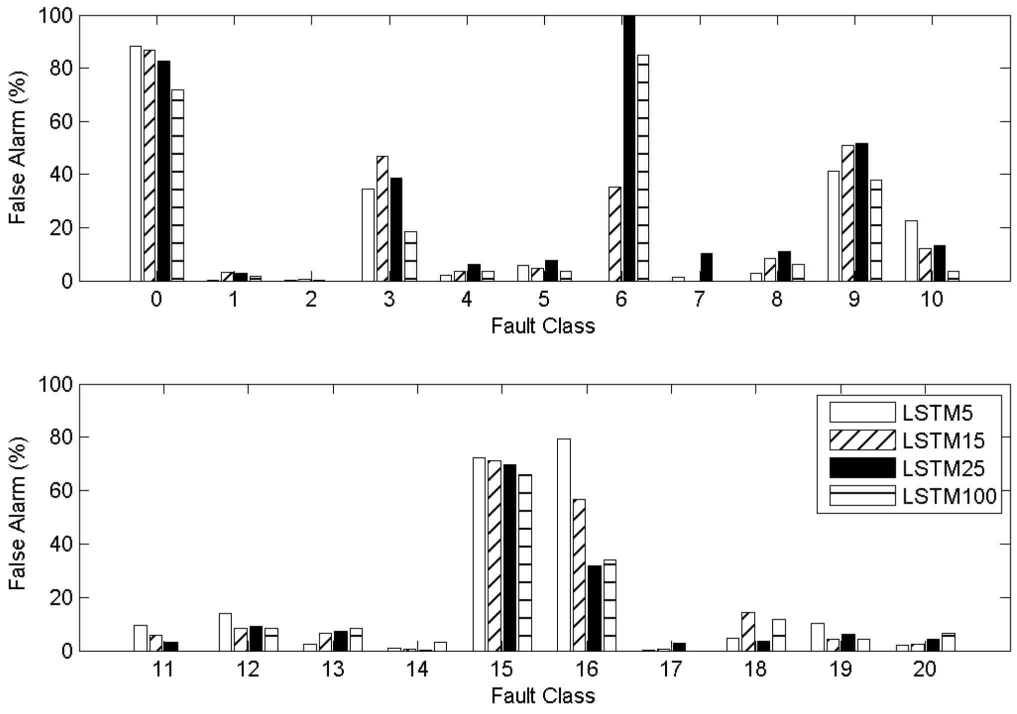


Figure 30. Effect of time samples in the false alarm rates in TEP

Following the hypothesis that different neural units may affect the classification accuracy the performance of the two main different recurrent units: LSTM and GRU were used to design classification models. Therefore, a classification model with GRU units was developed that used the same general structure that was used with LSTM, i.e. a combination of a stacked encoder connected to a softmax classification layer. Figure 31 and Figure 32 shows the results for the classification accuracy and the false alarm rates respectively when applying in the network GRU's as units. By inspecting the figures, we can perceive the same trends as with LSTM's, where we are able to identify easily between the observable faults and as we increase the number of samples we are able to increase the classification rates. However, when it comes to the difficult to observe faults the classification rates are still low and the false alarm rates are relatively high. To quantitatively compare the results obtained with the LSTM and GRU, the average classification rates were calculated as follows 79.8%, 80.3%, and 81% for LSTM and 77.4%, 78.2%, and 79.1% for GRU. Thus, the network based on LSTM performs slightly better for this particular study but this result may be problem dependent.

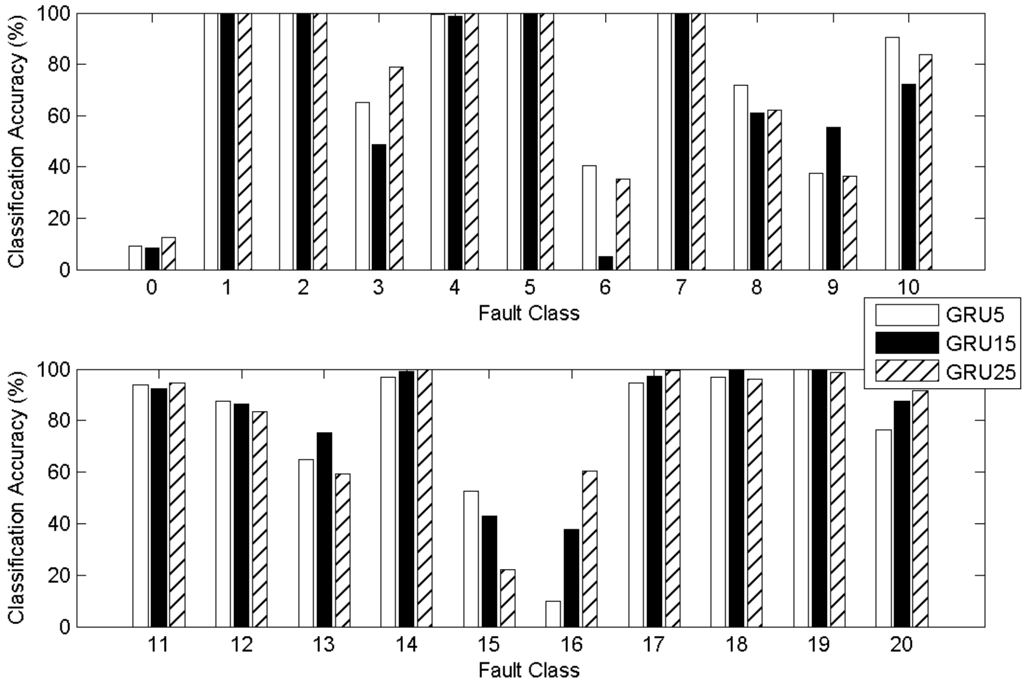


Figure 31. Effect of time samples in the classification accuracy in the TEP applying GRU's

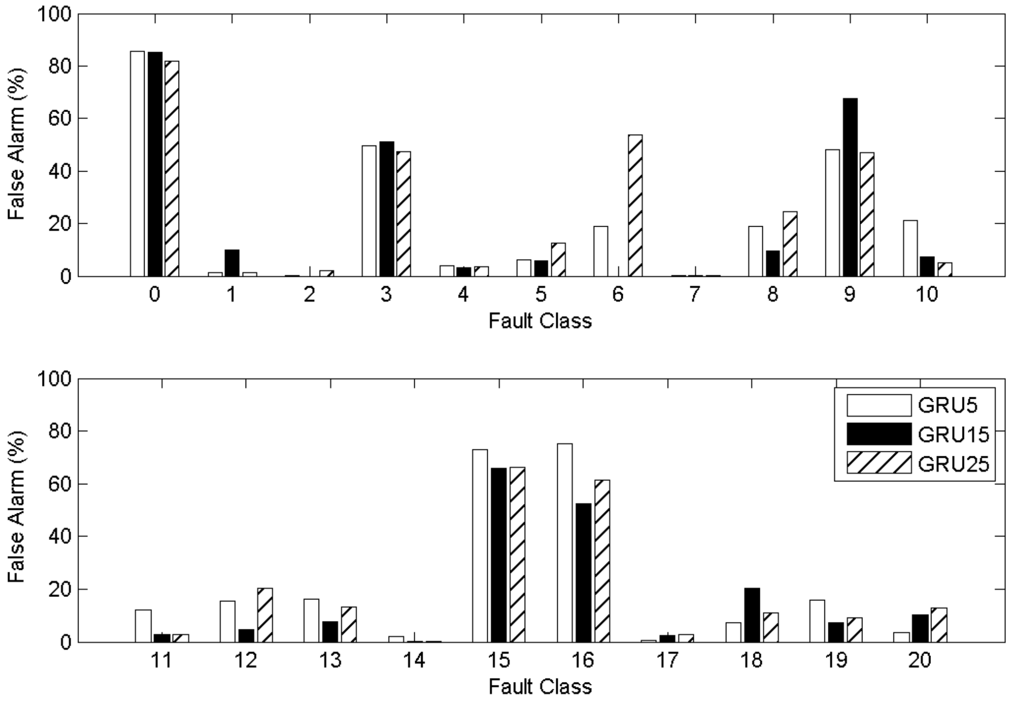


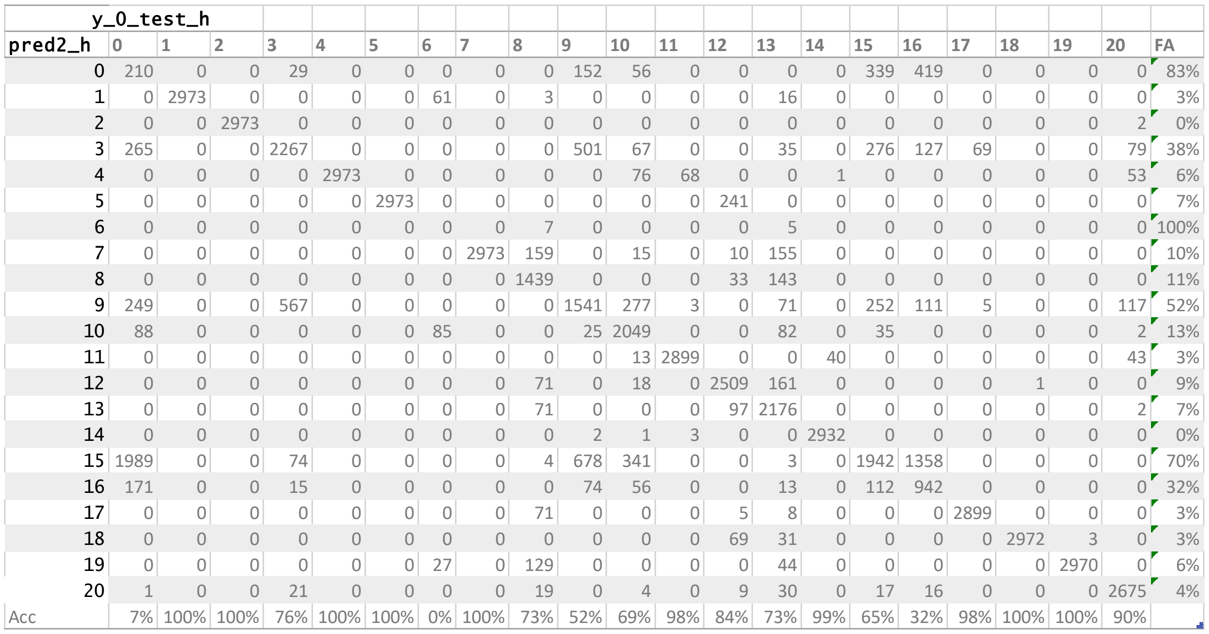
Figure 32. Effect of time samples in the false alarm rates in the TEP applying GRU's

4.3.3 Hierarchal Recurrent Neural Network

In this section we analyze and compare results obtained for fault classification while using a hierarchal partition of the faults as done with the static data based models presented in Chapter 3. The classification and false alarm rates obtained with the hierarchical structure with an RNN model are calculated and compared to the RNN model without a hierarchical structure.

As done with the models developed for static data, we look again at the confusion matrix of the classification problem with the RNN that uses LSTM as the neural unit, showed in table 11. Again, it can be observed from the confusion matrix that the classification rates are good for almost every observable fault. However, the confusion related to the difficult to observe faults is still very high. Therefore, the same hierarchical structure showed in Figure 22 was applied here to the RNN model that uses LSTM as its neural unit. The goal was to enhance the classification accuracy by using separate models for the easier to observe and difficult to observe faults respectively.

Table 11. Confusion matrix of the classification results with RNN for TEP for an all-to-all structure with 25 number of samples.



y_0_test_h																					
pred2_h	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	FA
0	210	0	0	29	0	0	0	0	0	152	56	0	0	0	0	339	419	0	0	0	0	83%
1	0	2973	0	0	0	0	61	0	3	0	0	0	0	16	0	0	0	0	0	0	0	3%
2	0	0	2973	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0%
3	265	0	0	2267	0	0	0	0	0	501	67	0	0	35	0	276	127	69	0	0	79	38%
4	0	0	0	0	2973	0	0	0	0	0	76	68	0	0	1	0	0	0	0	0	53	6%
5	0	0	0	0	0	2973	0	0	0	0	0	0	241	0	0	0	0	0	0	0	0	7%
6	0	0	0	0	0	0	0	0	7	0	0	0	0	5	0	0	0	0	0	0	0	100%
7	0	0	0	0	0	0	0	2973	159	0	15	0	10	155	0	0	0	0	0	0	0	10%
8	0	0	0	0	0	0	0	0	1439	0	0	0	33	143	0	0	0	0	0	0	0	11%
9	249	0	0	567	0	0	0	0	0	1541	277	3	0	71	0	252	111	5	0	0	117	52%
10	88	0	0	0	0	0	85	0	0	25	2049	0	0	82	0	35	0	0	0	0	2	13%
11	0	0	0	0	0	0	0	0	0	0	13	2899	0	0	40	0	0	0	0	0	43	3%
12	0	0	0	0	0	0	0	0	71	0	18	0	2509	161	0	0	0	0	1	0	0	9%
13	0	0	0	0	0	0	0	0	71	0	0	0	97	2176	0	0	0	0	0	0	2	7%
14	0	0	0	0	0	0	0	0	0	2	1	3	0	0	2932	0	0	0	0	0	0	0%
15	1989	0	0	74	0	0	0	0	4	678	341	0	0	3	0	1942	1358	0	0	0	0	70%
16	171	0	0	15	0	0	0	0	0	74	56	0	0	13	0	112	942	0	0	0	0	32%
17	0	0	0	0	0	0	0	0	71	0	0	0	5	8	0	0	0	2899	0	0	0	3%
18	0	0	0	0	0	0	0	0	0	0	0	0	69	31	0	0	0	0	2972	3	0	3%
19	0	0	0	0	0	0	27	0	129	0	0	0	0	44	0	0	0	0	0	2970	0	6%
20	1	0	0	21	0	0	0	0	19	0	4	0	9	30	0	17	16	0	0	0	2675	4%
Acc	7%	100%	100%	76%	100%	100%	0%	100%	73%	52%	69%	98%	84%	73%	99%	65%	32%	98%	100%	100%	90%	

This can be seen in Figure 33, where can observe the classification rates obtained by the hierarchical structure with 25 and 100 number of samples. In addition, Figure 35 shows a comparison between the classification rates obtained for 100 number of samples when using an all-to-all structure and using the hierarchical structure. Overall, by developing the hierarchical structure, we are able to increase the average classification rates of both structures from 80.6% to 84.1% and from 84.0% to 87.0% respectively. However, most importantly by doing the hierarchical structure we are able to classify correctly 2 faults that are generally considered difficult to observe: fault 3 and 9. The hierarchical structure is able to improve the classification accuracy in both of these faults by 43.0% and 17.2% respectively for the case of 100 number of samples and by 8.7% and 23.4% for 25 number samples. In the case of the normal state and the fault 15 we are able to see an increase in their classification rate but it is still very low. The false alarm rates with the hierarchical structure are shown in Figure 34.

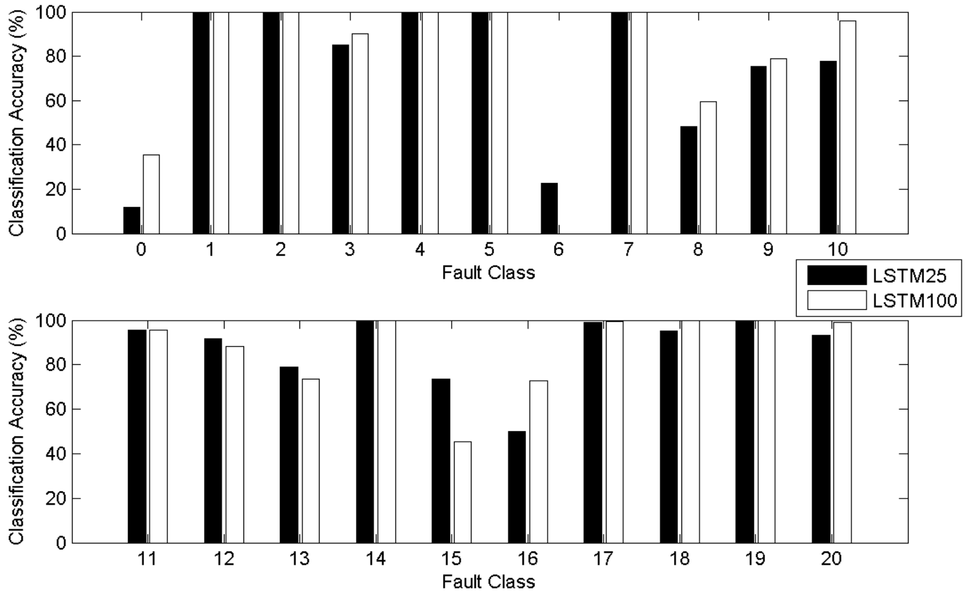


Figure 33. Classification accuracy of RNN applying a hierarchical structure for 25 and 100 number of samples.

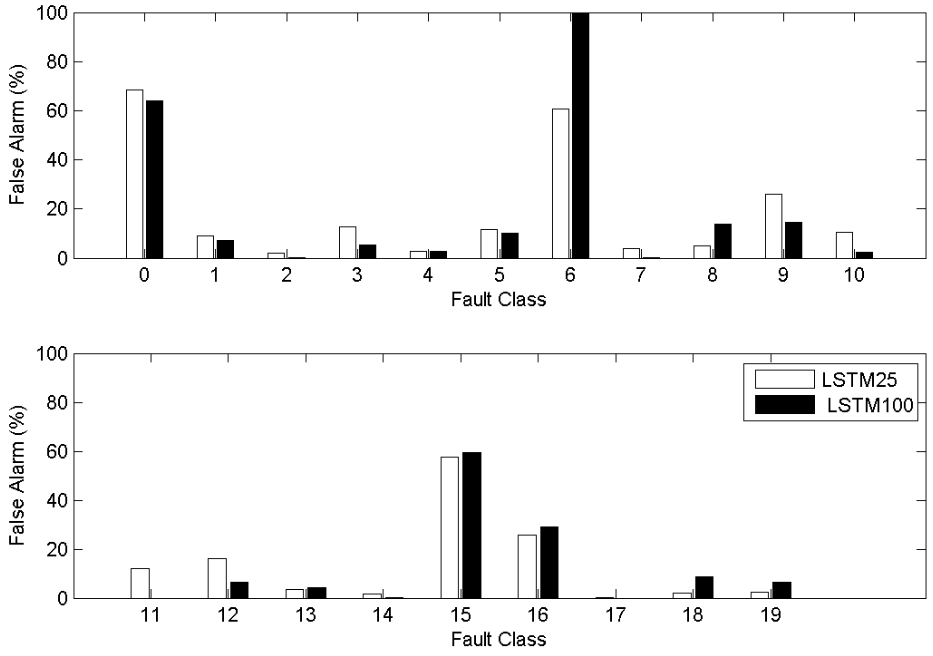


Figure 34. False alarm rates of RNN applying a hierarchical structure for 25 and 100 number of samples.

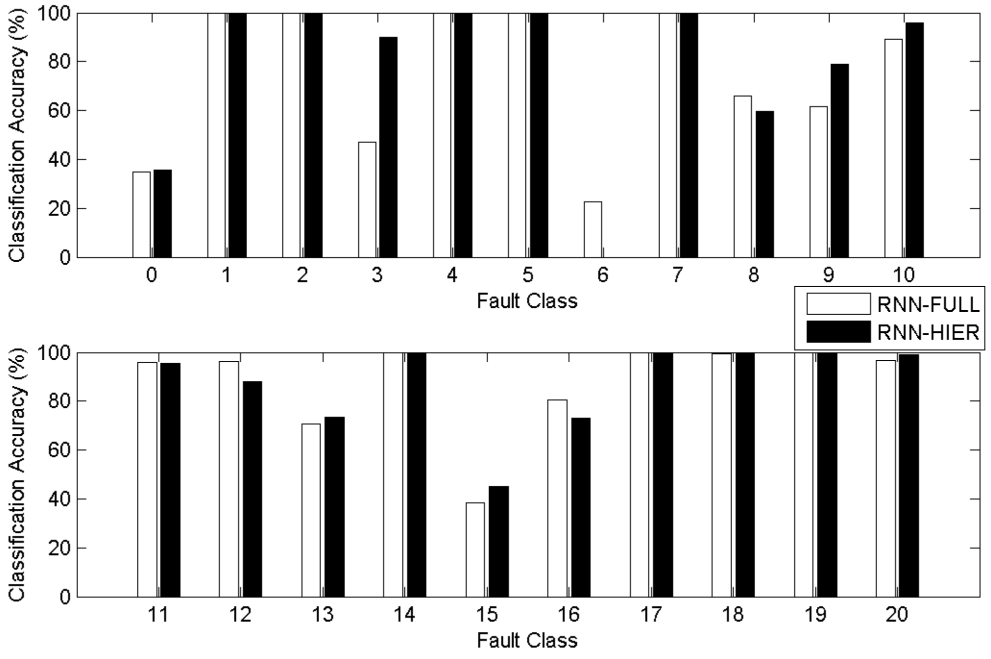


Figure 35. Comparison between the classification accuracy of RNN using an all-to-all structure and hierarchical structure with 100 number of samples.

4.3.4 Application of Pseudo Random Signal for Diagnosis Results

The last section showed that the use of a hierarchical structure can significantly enhance the classification accuracy but the resulting model is still unable to diagnose correctly between the normal state and fault 15. Therefore, in this section we implemented a pseudo random signal, as a disturbance, to a setpoint in the Tennessee Eastman Simulation to generate sufficient excitation so as to overcome low signal to noise ratios in signals associated to fault 15.

The reasoning behind the implementation of this pseudo random signal is that, even with the renormalization, we can observe from the confusion matrix a high level of confusion between the normal state and the fault 15. Consequently, following a process identification approach we add an external signal in order to increase the signal-to-noise ratio of signal/s that can be used for distinguishing fault 15 from the normal state. Towards this goal a pseudo random signal was designed and added as a disturbance in a process setpoint. In a practical implementation since it may be detrimental to vary a set-point with an added PRBS signal this signal could be introduced sporadically to test the occurrence of the fault. Thus, the proposed PRBS will not be used on a continuous basis. Analyzing the fault 15, which corresponds to stiction in the condenser cooling water valve, we introduced the additional excitation in a setpoint related to the condenser control. The nearest setpoint that could excite the condenser control loop was selected to be the separator temperature. The added set-point signal was implemented by running the simulation and implementing the signal at different times and data were collected while the signal was being introduced. This pseudo random signal can be observed in Figure 36. The data was then inputted through a new neural network model that was specifically designed to differentiate between the normal state and fault 15. Therefore, a new level was added to the hierarchical structure given before as shown in Figure 37. The results obtained by implementing this pseudo random signal can be observed in Figure 38, which is a comparison between the results obtained applying only the hierarchical structure and with a PRS with 100 number of samples. Figure 38 shows a dramatic increase in the classification rates of the normal state to 88.7% and fault 15 up to 76.4%.

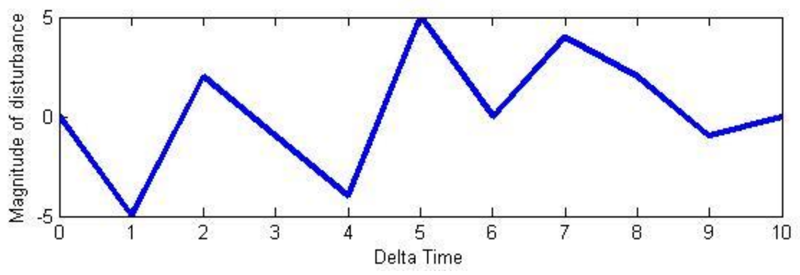


Figure 36. Pseudo random signal implemented in the Separator Temperature

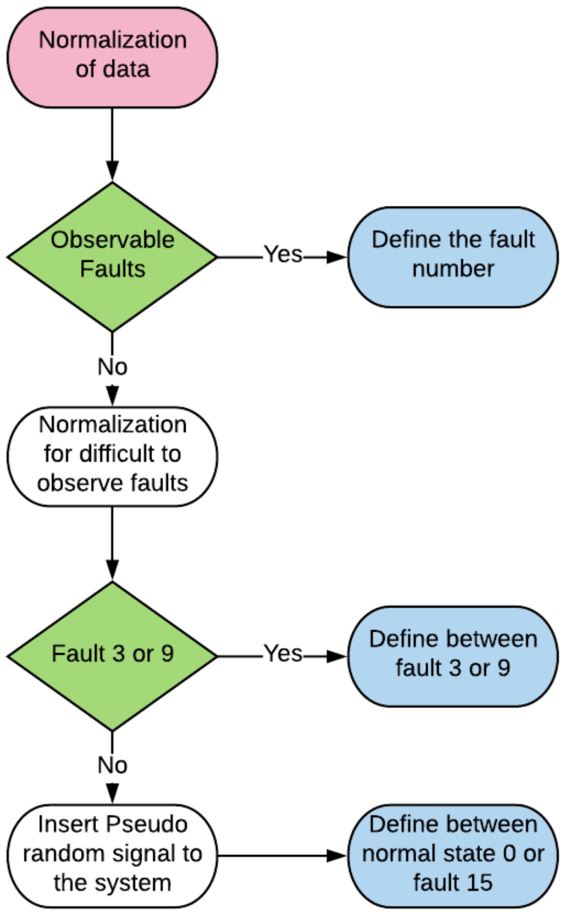

Normalization of data
Observable Faults
Yes
Define the fault number
No
Normalization for difficult to observe faults
Fault 3 or 9
Yes
Define between fault 3 or 9
No
Insert Pseudo random signal to the system
Define between normal state 0 or fault 15

Figure 37. Final Hierarchical structure in the TEP for mode3 with PRS

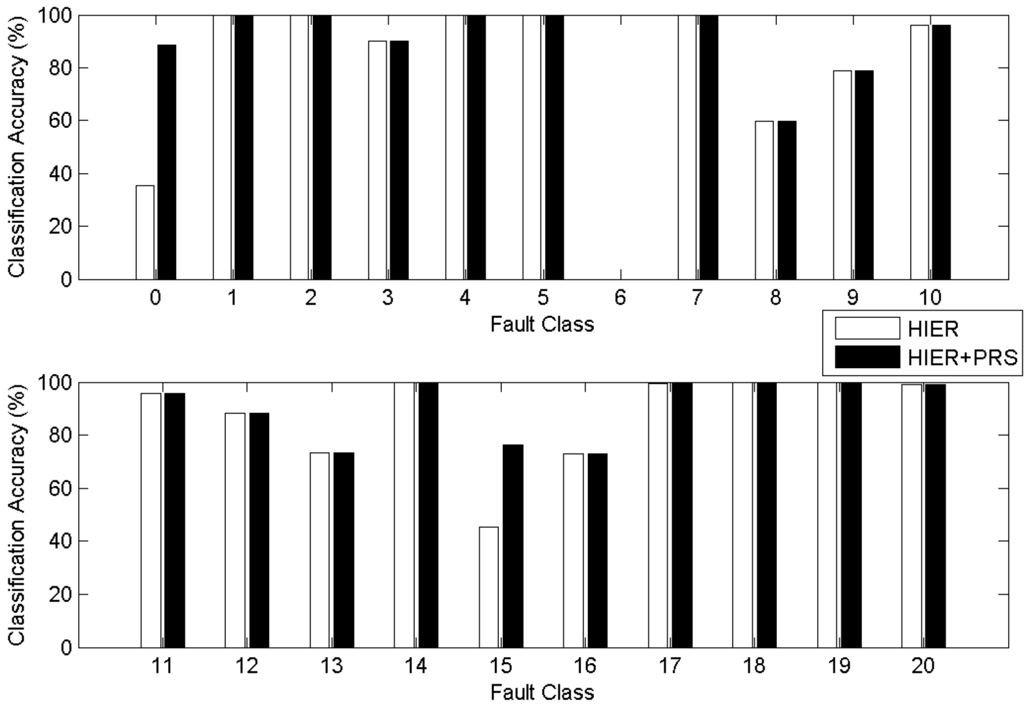


Figure 38. Comparison between the classification accuracy of RNN applying only the hierarchical structure and with a PRS with 100 number of samples.

4.3.5 Comparison Different Modes of Tennessee Eastman Simulation

In this section we analyze and compare results obtained using dynamical data for fault classification with the RNN algorithm for the three different operating mode used in the Tennessee Eastman Simulation. We will show the results for the remaining two modes: mode1 and mode1Skoge. In order to be able to compare between the different modes we applied the same network structure to every mode. Both the classification rates as well as the false alarm rates are obtained and compared for the different modes.

For consistency with the previous study we applied the same data extraction and data treatment as before. One important thing to consider, is that for mode1 and mode1Skoge there is a limitation with fault 16 that cannot be distinguished from fault 16 since the data is almost identical for the two variables. Therefore, the data from fault 16 was not used for the following study.

The results obtained for the effect of the number of samples considered in the models on the classification accuracy for TEP mode1 and mode1Skoge are shown in Figure 39 and Figure 41. The corresponding false alarm rates are shown in Figure 40 and Figure 42. Even though, the operating conditions may be different and the control schemes may change between the different modes similar trends are observed for both modes. Good classification rates are obtained for both modes for the easier to detect faults. However, the classification rates are low for the difficult to observe faults. As before, for both operating modes, increasing the number of samples in the models results in higher classification rates. In the case of mode1 we obtained average classification accuracies of 79.8%, 80.5%, 81.6% and 84.3% respectively as we increase the number of samples from 5, 15, 25 and 100 respectively. On the other hand, for mode1Skoge we obtained average classification accuracies of 83.4%, 84.6%, 85.4% and 87.3% for 5,15, 25 and 100 samples respectively. Figure 43 compares the classification accuracy for each class in each of the three different control schemes.

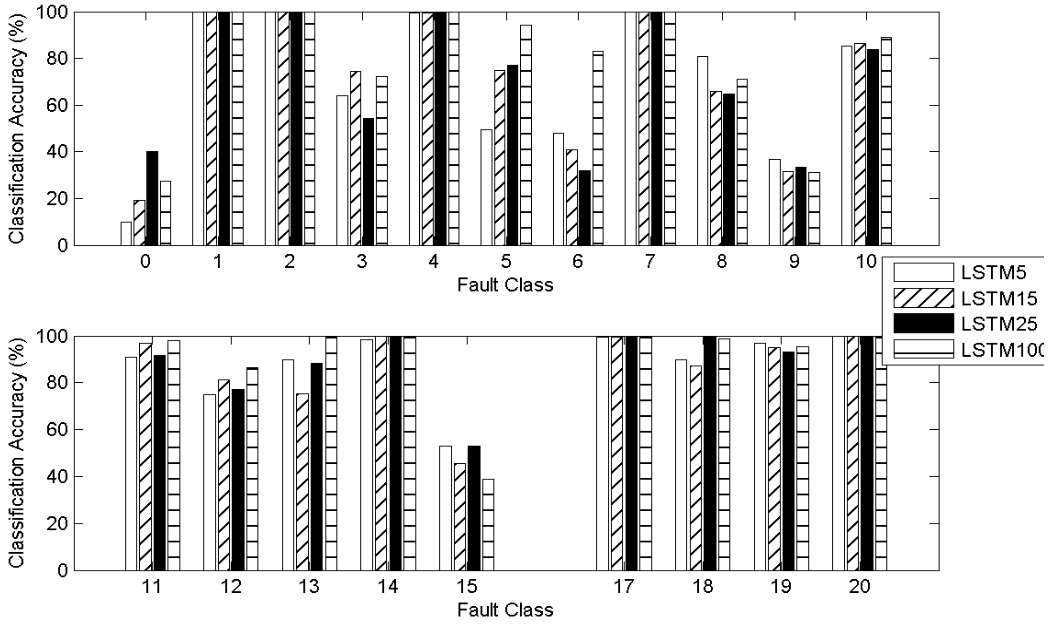


Figure 39. Effect of time samples in the classification accuracy TEP mode1

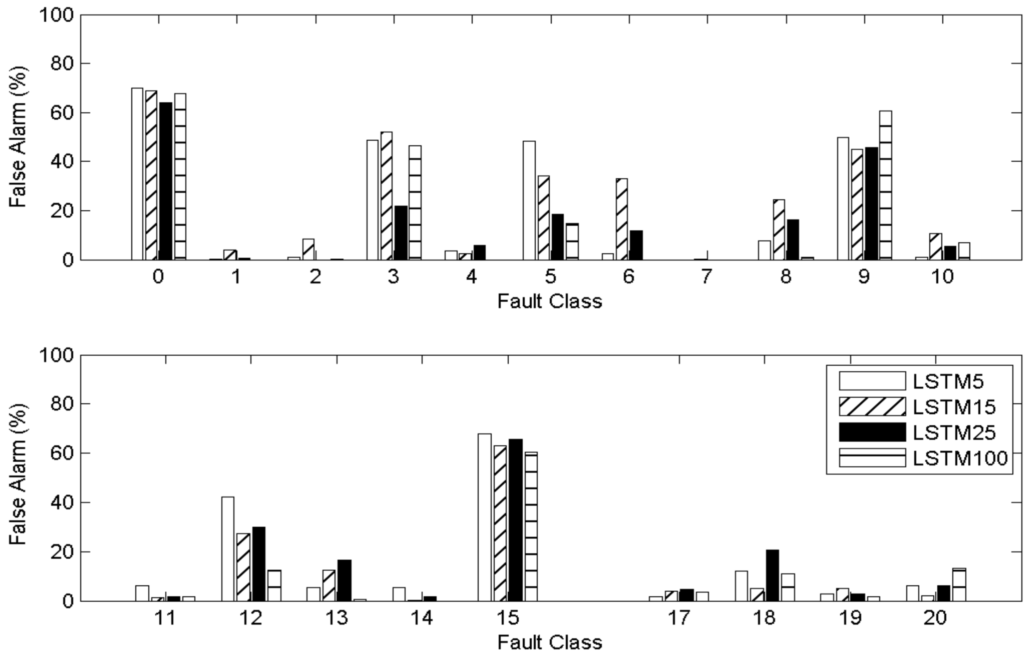


Figure 40. Effect of time samples in the false alarm rates TEP mode1

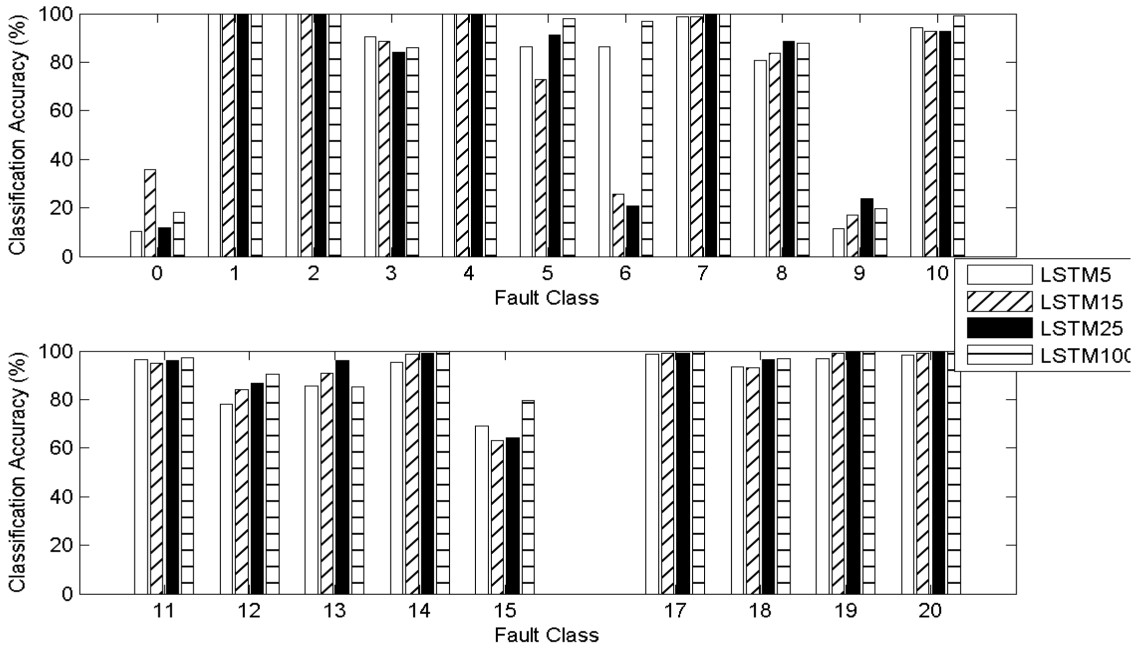


Figure 41. Effect of time samples in the classification accuracy TEP mode1Skoge

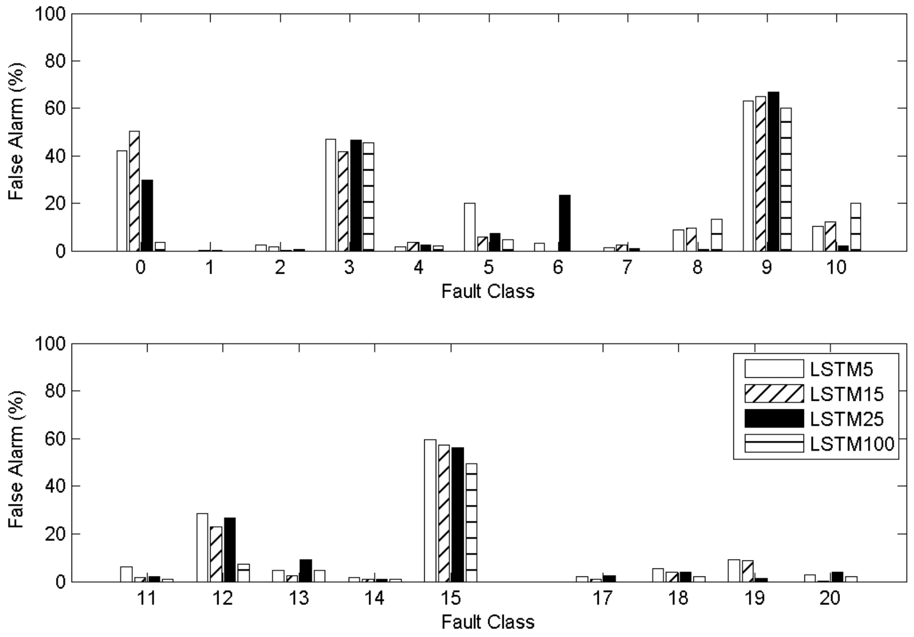


Figure 42. Effect of time samples in the false alarm rates TEP mode1Skoge

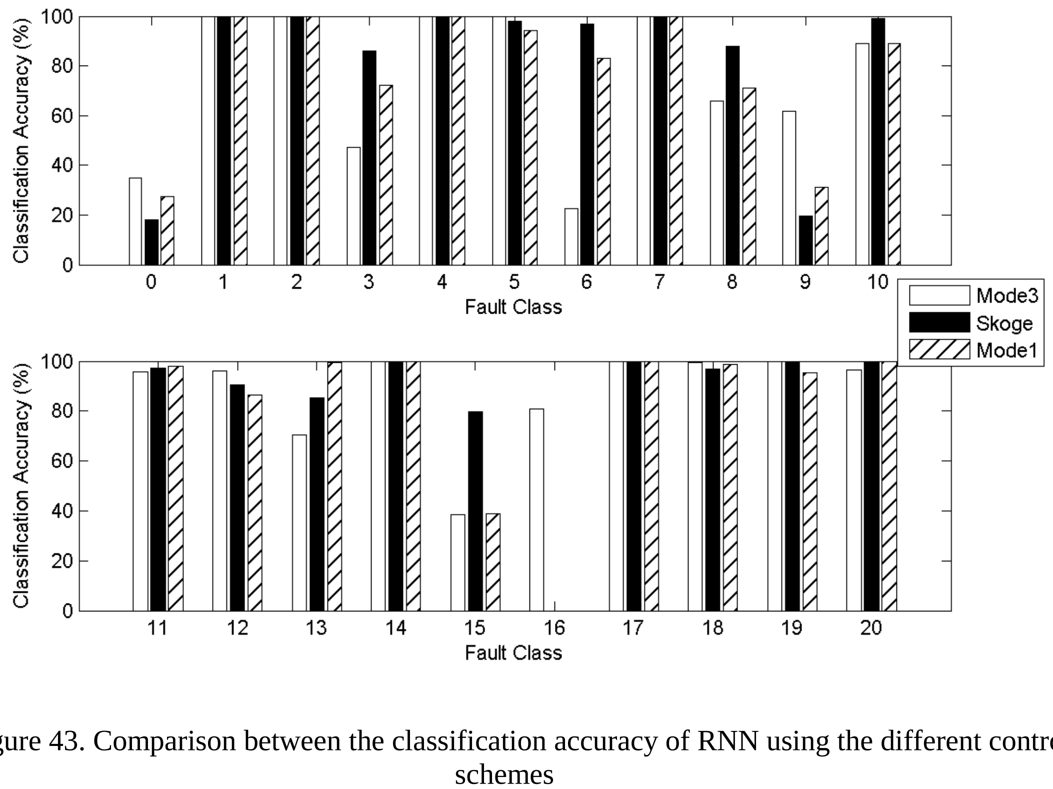


Figure 43. Comparison between the classification accuracy of RNN using the different control schemes

Since the classification accuracy in the difficult to observe faults was still low for both operating modes of the TEP, we applied the techniques proposed above to enhance the classification for mode3.Thus, first the hierarchical structure was applied to enhance the classification of the difficult to observe faults. However, in this case the results in the hierarchical structure showed almost the same results as with the original non-hierarchical model structure.

Subsequently, the introduction of PRS excitation signals was considered. However, it should be remembered that in the case of mode3 we were able to detect fault 3 and 9 using the hierarchical structure, which lead us to think that only applying an excitation to the system through the implementation of a pseudo random signal to a setpoint related to fault 15 would not be enough for the other operating modes of the TEP. Consequently, a study was conducted to identify the most suitable points for injecting excitation signals to identify faults 3 and 9. Fault 3 and 9 turn out to be closely related faults since they are both disturbances in the D feed temperature, but fault 3 is a step disturbance while fault 9 is a random disturbance. Consequently, we decided to add another pseudo random signal, which is shown in Figure 44 to the D feed ratio, in order to create a suitable excitation. The new hierarchical structure for the mode1 and mode1Skoge is shown in Figure 45. After developing this pseudorandom signal, we added both signals to the process as shown in Figure 36 and Figure 44 and applied this signal at different times during the simulation.

The results obtained for the classification rates are shown in Figure 46 and Figure 47 for the difficult to observe faults in mode1 and mode1Skoge respectively. It is found that by applying both of these signals the classification rates of faults 3 and 9 are significantly improved. For example, by inserting this pseudo random signal in mode1 we are able to improve the classification accuracy of these faults by 16.4% and 34.1% with respect to the full structure. On the other hand, in mode1Skoge the classification accuracy of these faults are improved by only 8% and 22.3% with respect to the full structure.

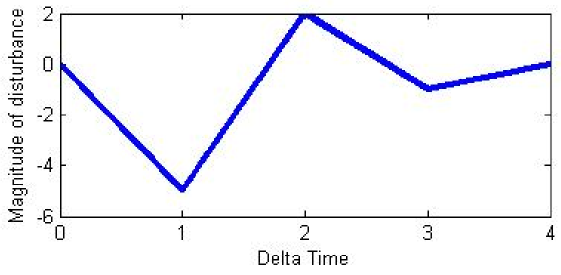


Figure 44. Pseudo random signal implemented in the D feed ratio

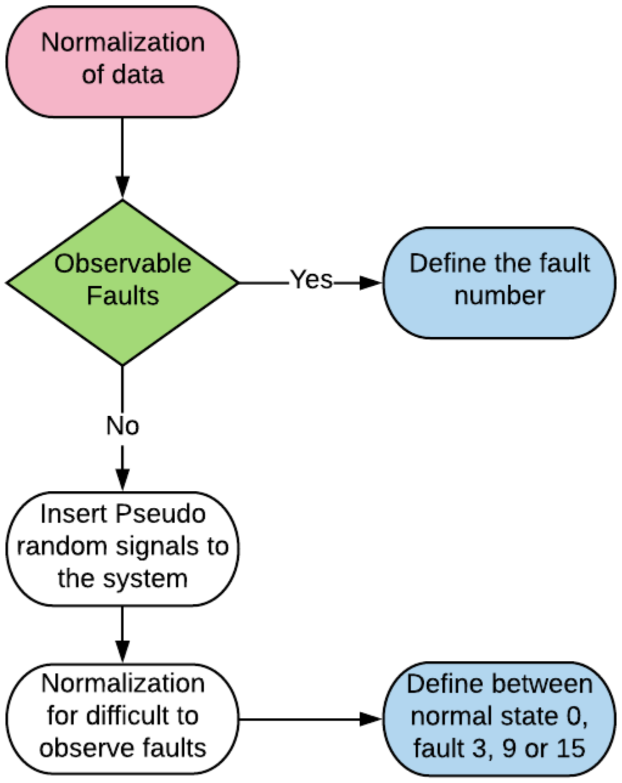

Normalization of data
Observable Faults
Yes
Define the fault number
No
Insert Pseudo random signals to the system
Normalization for difficult to observe faults
Define between normal state 0, fault 3, 9 or 15

Figure 45. Hierarchical structure for partitioning of classes in the TEP for mode1 and mode1Skoge

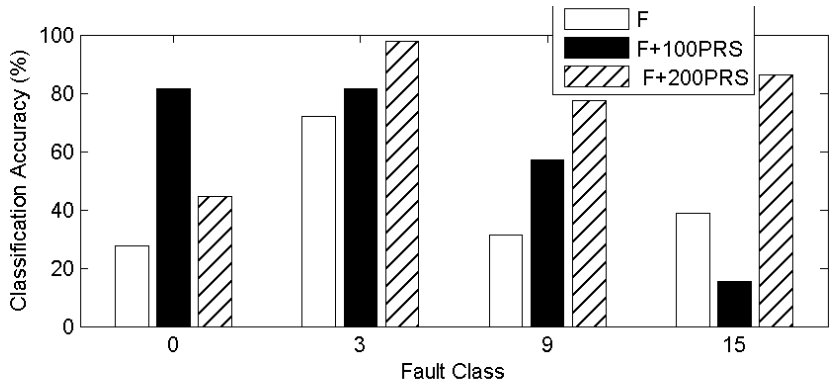


Figure 46. Comparison between classification accuracy of RNN, in mode1, applying a full structure (F), hierarchical structure and PRS with 100 number of samples (F+100PRS), and hierarchical structure and PRS with 200 number of samples (F+200PRS).

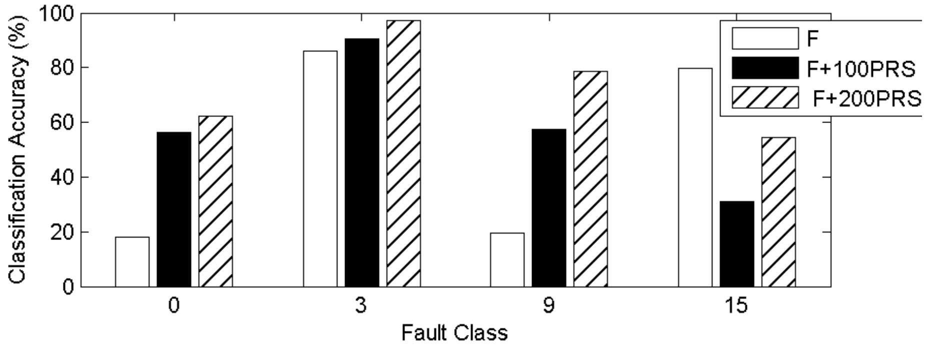


Figure 47. Comparison between classification accuracy of RNN, in mode1Skoge, applying a full structure (F), hierarchical structure and PRS with 100 number of samples (F+100PRS), and hierarchical structure and PRS with 200 number of samples (F+200PRS).

4.4 Conclusions of Chapter 4

In this chapter we compared the accuracy of deep learning techniques to statistical techniques when using dynamical information. Recurrent neural networks were shown to be an effective tool for fault classification due to their ability to capture nonlinear dynamic behavior. We showed that the classification averages can be enhanced depending on the length of the time horizon of past data fed to the RNN, where we obtained 79.8%, 80.3%, 81% and 84% for the RNN with 5, 15, 25 and 100 number of samples respectively. In addition, for mode3, a hierarchical model structure was developed, which was effective for classifying some faults that are very difficult to observe by other methods: fault 3 and 9 by 43.0% and 17.2% respectively for the case of 100 number of samples. Thus, by splitting the classification model into a model dealing with the easier to detect fault and another model that classify the more difficult to detect faults the classification accuracy can be improved. In addition, the introduction of external excitation to the system through the employment of pseudorandom signals has been shown to help increasing the classification accuracies of difficult to observe faults, specifically increasing the classification rates of the normal state to 88.7% and fault 15 up to 76.4%. The methods to enhance the classification accuracy were further tested for two additional operating modes available in the TEP simulator resulting in similar improvements to the classification accuracy.

Chapter 5 Final Remarks and Future Work

5.1 Concluding remarks

In a large chemical plant, the quick identification of faults and disturbances is crucial in order to keep the process running at optimal condition. This work studied the application of deep learning as a way to increase the detection and classification of faults in a chemical benchmark simulation plant, the Tennessee Eastman Problem (TEP). The TEP simulation contains 20 different disturbances that were used during this study to make the classification problem. This work proposes the division of these faults in two categories: observable and difficult to observe. The difference between these two categories lies in the signal-to-noise ratio, where the former result in variable changes with good signal-to-noise ratio with discernible symptoms whereas the later exhibit low signal-to-noise ratio with similar symptoms to other disturbances. Specifically and as previously reported by other researchers the difficult to observe faults in the TEP are fault 3, fault 9, fault 15 and the normal state.

First, the accuracy of deep learning models (ANN) and statistical models (SVM and PCA+SVM) was compared to classify faults based on static information. By comparing ANN with classical statistical tools for fault detection and diagnosis such as PCA (Principal Component Analysis), we demonstrated the inherent ability of neural networks to explain nonlinear behavior. The overall classification average obtained in an all-to-all structure was for SVM 49%, for PCA 67.3% and for ANN 73.8%. The superior results obtained with ANN is directly related to the fact that ANN uses nonlinear activation functions, which are able to capture nonlinear behavior of the system. In addition, a hierarchical structure was developed were faults with similar characteristics were assigned to subgroups and then the individual faults were re-classified within the subgroup resulting in an overall classification average improvement of +3% for ANN. Additionally, this same study was applied to the different control scheme modes available for the Tennessee Eastman Simulator, showing similar results to those obtained with the original operating mode, obtaining 73.8%, 71.1%, and 76.6% for mode3, mode1, and mode1Skoge respectively. Similar trends between the three modes were found, where we obtained good classification accuracies most faults. However, the classification accuracies for the difficult to observe faults was deficient.

Following the results obtained by the static study, it was hypothesized that the addition of dynamic information may improve classification accuracy. Initially, a comparison between deep learning techniques to statistical techniques when using dynamical information was developed. A comparison of Dynamic PCA to Recurrent Neural Networks (RNN) with 15 number of time intervals showed an overall classification average of 69.6% and 80.3% for DPCA and RNN respectively. Thus, as for the study with static data, it was shown that RNN is an effective tool for fault classification due to their ability to capture nonlinear dynamic behavior. Afterwards, the study showed that the classification averages can be enhanced depending on the length of the time horizon of past data fed to the RNN, where we obtained 79.8%, 80.3%, 81% and 84% for the RNN with 5, 15, 25 and 100 number of samples respectively. However, most of the improvement in classification improvements occurred for the observable faults. Therefore, a hierarchical model structure was developed, which was effective for classifying some faults that are very difficult to observe by other methods: the classification of fault 3 and 9 improved by 43.0% and 17.2% respectively for the case of 100 number of samples. The explanation behind these improvements is that by splitting the classification model into a model dealing with the easier to detect fault and another model that classify the more difficult to detect faults and re-normalizing the variables within each group we are able to increase the signal-to-noise ratio of these faults thus leading to better classification. It could be argued that similar results could have been obtained by designing a much larger network that encompasses both models of the hierarchical structure but this will result in a very high computational cost.

Also, despite the classification improvements for fault 3 and 9, it was found that the normal state and the fault 15 were still highly confounded thus resulting in poor differentiation between the normal state and Fault 15. To address this problem external excitation was added to the system through the injection of pseudorandom signals, increasing the classification rates of the normal state to 88.7% and fault 15 up to 76.4%. Thus, this additional excitation improved the signal to noise ratio in the responses resulting from these faults leading to better classification results. Lastly, the methods proposed to enhance fault classification were further tested for two additional operating modes available in the TEP simulator resulting in similar improvements to the classification accuracy.

5.2 Future work

Based on the studies carried out during this research, the following recommendations for future work are proposed:

1. Optimal structure selection of the deep learning algorithm: Apply a grid search that will be able to infer the best combination of number of samples, number of layers, number of nodes and values of regularization terms in the RNN to be used for classification.
2. Find theoretical limits for classification accuracy: Continue to study the effect of the number of samples, due that, theoretically, there should be a number of sample size where the classification accuracy should stop improving.
3. Optimal design of external excitation signals for fault classification: Study the effect of implementing different kinds of excitation signals in into the process order to increase the observability of the difficult to observe faults. As well, we should study the optimal time required to obtain a good accuracy when applying an excitation signal. In addition, it would be valuable to observe how the classification performance changes as we vary the location of the excitation signal across the process.
4. Optimal integration of hierarchical structure and external excitation: Analyze if the hierarchical structure can improve the results after inserting the excitation signal. This can be applied as an initial tool to differentiate between the two kinds of difficult to observe faults, where we usually obtained a lot of confusion between the normal state and fault 15 and between fault 3 and fault 9.
5. Further studies on the effect of change in operating mode on fault classification accuracy: Develop more comparison studies between the different control schemes of the TEP simulation and the reasons of discrepancies in the fault detection and diagnosis of fault in the different modes.
6. Consideration of additional faults: Extend this study by adding the 8 additional disturbances proposed by Ricker in the new TEP simulation update (Bathelt et al., 2015).

Bibliography

Aurelio Ranzato, M., Poultney, C., Chopra, S., Cun, Y. L., Ranzato, M., Poultney, C., … Cun, Y. L. (2007). Efficient Learning of Sparse Representations with an Energy-Based Model. *Advances in Neural Information Processing Systems*, *19*(1), 1137–1144. Retrieved from http://papers.nips.cc/paper/3112-efficient-learning-of-sparse-representations-with-an-energy-based-model%5Cnhttp://papers.nips.cc/paper/3112-efficient-learning-of-sparse-representations-with-an-energy-based-model.pdf

Bathelt, A., Ricker, N. L., & Jelali, M. (2015). Revision of the Tennessee eastman process model. *IFAC-PapersOnLine*, *28*(8), 309–314. https://doi.org/10.1016/j.ifacol.2015.08.199

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy Layer-Wise Training of Deep Networks. *Advances in Neural Information Processing Systems*. https://doi.org/citeulike-article-id:4640046

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning Long Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166. https://doi.org/10.1109/72.279181

Bottou, L. (1991). Stochastic Gradient Learning in Neural Networks. *Proceedings of Neuro-Nımes*.

Bottou, L. (1998). Online Learning and Stochastic Approximations. In *Online Learning in Neural Networks*.

Bottou, L., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Jackel, L. D., … Vapnik, V. (1994). Comparison of classifier methods: a case study in handwritten digit recognition. *Proceedings of the 12th IAPR International Conference on Pattern Recognition (Cat. No.94CH3440-5)*, *2*, 77–82. https://doi.org/10.1109/ICPR.1994.576879

Cao, Y., & Samuel, R. T. (2016). Dynamic latent variable modelling and fault detection of Tennessee Eastman challenge process. *Proceedings of the IEEE International Conference on Industrial Technology*, *2016–May*(March), 842–847. https://doi.org/10.1109/ICIT.2016.7474861

Chadha, G. S., & Schwung, A. (2018). Comparison of deep neural network architectures for fault detection in Tennessee Eastman process. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 1–8. https://doi.org/10.1109/ETFA.2017.8247619

Chiang, L. H., Kotanchek, M. E., & Kordon, A. K. (2004). Fault diagnosis based on Fisher discriminant analysis and support vector machines. *Computers and Chemical Engineering*, *28*(8), 1389–1401. https://doi.org/10.1016/j.compchemeng.2003.10.002

Chiang, L. H., Russell, E. L., & Braatz, R. D. (2000). Fault diagnosis in chemical processes using Fisher discriminant analysis, discriminant partial least squares, and principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, *50*(2), 243–252.

Chiang, L. H., Russell, E. L., & Braatz, R. D. (2001). Fault detection and diagnosis in industrial systems. *Time*. https://doi.org/10.1109/DSN.2009.5270324

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. https://doi.org/10.3115/v1/D14-1179

Chung, J. (2014). Gated Recurrent Neural Networks on Sequence Modeling, 1–9.

Chung, J. (2015). Gated Feedback Recurrent Neural Networks, *37*.

Clevert, D.-A., Unterthiner, T., & Hochreiter, S. (2015). Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs), 1–14. https://doi.org/10.3233/978-1-61499-672-9-1760

Comon, P. (1994). Independent component analysis, A new concept? *Signal Processing*, *36*(3), 287–314. https://doi.org/10.1016/0165-1684(94)90029-9

Cortes, C., & Vapnik, V. (1995). Support-Vector Networks, *297*, 273–297.

Dong, J., Zhang, K., Huang, Y., Li, G., & Peng, K. (2015). Adaptive total PLS based quality-relevant process monitoring with application to the Tennessee Eastman process. *Neurocomputing*, *154*, 77–85. https://doi.org/10.1016/j.neucom.2014.12.017

Downs, J. J., & Vogel, E. F. (1993). A plant-wide industrial process control problem. *Computers and Chemical Engineering*, *17*(3), 245–255. https://doi.org/10.1016/0098-1354(93)80018-I

Du, Y., & Du, D. (2018). Fault Detection using Empirical Mode Decomposition based PCA and CUSUM with Application to the Tennessee Eastman Process. *IFAC-PapersOnLine*, *51*(18), 488–493. https://doi.org/10.1016/j.ifacol.2018.09.377

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, *12*, 2121–2159.

https://doi.org/10.1109/CDC.2012.6426698

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

Fisher, R. A. (1936). The use of multiple measures in taxonomic problems. *Annals of Eugenics*. https://doi.org/10.1111/j.1469-1809.1936.tb02137.x

Glorot, X., & Bordes, A. (2011). Deep Sparse Rectifier Neural Networks, *15*, 315–323.

Goodfellow, I., Yoshua, B., & Aaron, C. (2016). Deep Learning. *Deep Learning*. https://doi.org/10.1016/B978-0-12-391420-0.09987-X

Herculano-Houzel, S. (2009). The human brain in numbers: a linearly scaled-up primate brain. *Frontiers in Human Neuroscience*, *3*(November), 1–11. https://doi.org/10.3389/neuro.09.031.2009

Hinton, G. E., Osindero, S., & Teh, Y.-W. (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. R. (2006). Reducing the dimensionality of data with neural networks. *Science*. https://doi.org/10.1126/science.1127647

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors, 1–18. https://doi.org/arXiv:1207.0580

Hochreiter, J. (1991). Untersuchungen zu dynamischen neuronalen Netzen. *Master's Thesis, Institut Fur Informatik, Technische Universitat, Munchen*.

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

Hofmann, T., Schölkopf, B., & Smola, A. J. (2008). Kernel methods in machine learning. *Annals of Statistics*. https://doi.org/10.1214/009053607000000677

Hoskins, J. C., Kaliyur, K. M., & Himmelblau, D. M. (1991). Fault diagnosis in complex chemical plants using artificial neural networks. *American Institute of Chemical Engineers Journal*, *37*(1), 137–141.

Hotelling, H. (1933). Analysis of a complex of statistical variables into Principal Components. Jour. Educ. Psych., 24, 417-441, 498-520. *The Journal of Educational Psychology*, *24*, 417–441.

Hsu, C. W., & Lin, C. J. (2002). A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, *13*(2), 415–425. https://doi.org/10.1109/72.991427

Ioffe, S., & Szegedy, C. (2015). Batch Normalization. In *International conference on machine learning*. https://doi.org/10.1007/s13398-014-0173-7.2

Isermann, R. (2004). Model-Based Fault Detection and Diagnosis - Status and Applications. *IFAC Proceedings Volumes*, *37*(6), 49–60. https://doi.org/10.1016/S1474-6670(17)32149-3

Isermann, R. (2006). *Fault-diagnosis systems: An introduction from fault detection to fault tolerance*. *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. https://doi.org/10.1007/3-540-30368-5

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks*, *1*(4), 295–307. https://doi.org/10.1016/0893-6080(88)90003-2

Jaeger, H. (2010). The " echo state " approach to analysing and training recurrent neural networks – with an Erratum note 1. *GMD Report*, (148), 1–47. https://doi.org/citeulike-article-id:9635932

Jiang, B., Huang, D., Zhu, X., Yang, F., & Braatz, R. D. (2015). Canonical variate analysis-based contributions for fault identification. *Journal of Process Control*, *26*, 17–25. https://doi.org/10.1016/j.jprocont.2014.12.001

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization, 1–15. https://doi.org/http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503

Ku, W., Storer, R. H., & Georgakis, C. (1995). Disturbance detection and isolation by dynamic principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, *30*(1), 179–196. https://doi.org/10.1016/0169-7439(95)00076-3

Kulkarni, A., Jayaraman, V. K., & Kulkarni, B. D. (2005). Knowledge incorporated support vector machines to detect faults in Tennessee Eastman Process. *Computers and Chemical Engineering*, *29*(10), 2128–2133. https://doi.org/10.1016/j.compchemeng.2005.06.006

Larimore, W. E. (1990). Canonical variate analysis in identification, filtering, and adaptive control. In *29th IEEE Conference on Decision and Control*. https://doi.org/10.1109/CDC.1990.203665

Larsson, T., Hestetun, K., Hovland, E., & Skogestad, S. (2001). Self-Optimizing Control of a Large-Scale Plant : The Tennessee Eastman Process, 4889–4901. https://doi.org/10.1021/ie000586y

Lau, C. K., Ghosh, K., Hussain, M. A., & Che Hassan, C. R. (2013). Fault diagnosis of Tennessee Eastman process with multi-scale PCA and ANFIS. *Chemometrics and Intelligent Laboratory Systems*, *120*, 1–14. https://doi.org/10.1016/j.chemolab.2012.10.005

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Lee, J.-M., Qin, S. J., & Lee, I.-B. (2006). Fault detection and diagnosis based on modified independent component analysis. *AIChE Journal*. https://doi.org/10.1002/aic.10978

Li, R., Olson, J. H., & Chester, D. L. (1990). Dynamic fault detection and diagnosis using neural networks. *Proceedings 5th IEEE International Symposium on Intelligent Control 1990*, *4*(1), 1169–1174. https://doi.org/10.1109/ISIC.1990.128602

Ljung, L. (1987). *System Identification Theory for User*. *Prentice Hall*. https://doi.org/10.1016/0005-1098(89)90019-8

Lv, F., Wen, C., Bao, Z., & Liu, M. (2016). Fault diagnosis based on deep learning. *2016 American Control Conference (ACC)*, (2), 6851–6856. https://doi.org/10.1109/ACC.2016.7526751

Mahadevan, S., & Shah, S. L. (2009). Fault detection and diagnosis in process data using one-class support vector machines. *Journal of Process Control*, *19*(10), 1627–1639. https://doi.org/10.1016/j.jprocont.2009.07.011

McCulloch, W. S., & Pitts, W. (1943). A Logical Calculus of the Idea Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, *5*, 115–133. https://doi.org/10.1007/BF02478259

Mika, S., Ratsch, G., Weston, J., & Scholkopf, B. (1999). Fisher discriminant analysis with kernels. *Neural Networks for Signal Processing IX, 1999. Proceedings of the 1999 IEEE Signal Processing Society Workshop*. https://doi.org/10.1109/NNSP.1999.788121

Minsky, M., & Papert, S. (1969). *Perceptrons*. *Perceptrons.*

Morari, M., Arkun, Y., & Stephanopoulos, G. (1980). Studies in the synthesis of control structures for chemical processes: Part I: Formulation of the problem. Process decomposition and the classification of the control tasks. Analysis of the optimizing control structures. *AIChE Journal*, *26*(2), 220–232. https://doi.org/10.1002/aic.690260205

Nair, V., & Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning*, (3), 807–814. https://doi.org/10.1.1.165.6419

Nimmo, I. (1995). Adequately address abnormal situation operations. *Chemical Engineering Progress, 91 (9),* 36–45.

Odiowei, P. E. P., & Yi, C. (2010). Nonlinear Dynamic Process Monitoring Using Canonical Variate Analysis and Kernel Density Estimations. *Industrial Informatics, IEEE Transactions On*, *6*(1), 36–45. https://doi.org/10.1109/TII.2009.2032654

Pearson, K. (1901). LIII. *On lines and planes of closest fit to systems of points in space*. *Philosophical Magazine Series 6, 2*(11), 559–572. https://doi.org/10.1080/14786440109462720

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, *4*(5), 1–17. https://doi.org/10.1016/0041-5553(64)90137-5

Prechelt, L. (1998). Automatic early stopping using cross validation: Quantifying the criteria. *Neural Networks*, *11*(4), 761–767. https://doi.org/10.1016/S0893-6080(98)00010-0

Qin, S. J. (2009). *Data-driven Fault Detection and Diagnosis for Complex Industrial Processes*. *IFAC Proceedings Volumes* (Vol. 42). IFAC. https://doi.org/10.3182/20090630-4-ES-2003.00184

Rashid, M. M., & Yu, J. (2012). Hidden Markov Model Based Adaptive Independent Component Analysis Approach for Complex Chemical Process Monitoring and Fault Detection. *Industrial & Engineering Chemistry Research*, *51*(15), 5506–5514. https://doi.org/10.1021/ie300203u

Rato, T. J., & Reis, M. S. (2013). Fault detection in the Tennessee Eastman benchmark process using dynamic principal components analysis based on decorrelated residuals (DPCA-DR). *Chemometrics and Intelligent Laboratory Systems*, *125*, 101–108. https://doi.org/10.1016/j.chemolab.2013.04.002

Ricker, N. L. (1996). Decentralized control of the Tennessee Eastman Challenge Process. *Journal of Process Control*, *6*(4), 205–221. https://doi.org/10.1016/0959-1524(96)00031-5

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in …. *Psychological Review*, *65*(6), 386–408. https://doi.org/10.1037/h0042519

Rosenbloom, P. (1956). The method of steepest descent. *Proceedings of Symposia in Applied Mathematics, Vol. 6*.

Ruder, S. (2016). An overview of gradient descent optimization algorithms, 1–14. https://doi.org/10.1111/j.0006-341X.1999.00591.x

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. https://doi.org/10.1038/323533a0

Samuel, R. T., & Cao, Y. (2015). Kernel canonical variate analysis for nonlinear dynamic process monitoring. *IFAC-PapersOnLine*, *28*(8), 605–610. https://doi.org/10.1016/j.ifacol.2015.09.034

Shamir, O. (2003). Stochastic Gradient Descent for Non-smooth Optimization : Convergence Results and Optimal Averaging Schemes, 1–13.

Shams, M. Bin, Budman, H., & Duever, T. (2010). Fault detection using CUSUM based techniques with application to the Tennessee Eastman process. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, *9*(PART 1), 109–114. https://doi.org/10.3182/20100705-3-BE-2011.0009

Shams, M. Bin, Budman, H., & Duever, T. (2011). Finding a trade-off between observability and economics in the fault detection of chemical processes. *Computers and Chemical Engineering*. https://doi.org/10.1016/j.compchemeng.2010.04.006

Skogestad, S. (2000). Plantwide control : the search for the self-optimizing control structure, *10*.

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*, 1929–1958. https://doi.org/10.1214/12-AOS1000

Stefatos, G., & Hamza, A. Ben. (2010). Dynamic independent component analysis approach for fault detection and diagnosis. *Expert Systems with Applications*, *37*(12), 8606–8617. https://doi.org/10.1016/j.eswa.2010.06.101

Stubbs, S., Zhang, J., & Morris, J. (2012). Fault detection in dynamic processes using a simplified monitoring-specific CVA state space modelling approach. *Computers and Chemical Engineering*, *41*, 77–87. https://doi.org/10.1016/j.compchemeng.2012.02.009

Tesauro, G. (1992). Practical Issues in Temporal Difference Learning. *Machine Learning*, *8*(3–4), 257–277. https://doi.org/10.1007/BF00992697

Touretzky, D. S., & Hinton, G. E. (1985). Symbols among the neurons: details of a connectionist inference architecture. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, *1*, 238–243.

Ulrich, H. G. K. (1999). Pairwise classification and support vector machines. In *Advances in kernel methods*.

Vapnik, V. N. (1998). Statistical Learning Theory. *Interpreting*. https://doi.org/10.2307/1271368

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th International Conference on Machine Learning - ICML '08*, 1096–1103. https://doi.org/10.1145/1390156.1390294

Widrow, B. (1960). An Adaptive "Adaline" Neuron Using Chemical "Memistors." *Stanford Electronics Laboratories Technical Report*.

Wold, H. (1985). Partial Least Squares. In *Encyclopedia of Statistical Sciences*. https://doi.org/10.1017/CBO9781107415324.004

Xie, D., & Bai, L. (2015). A Hierarchical Deep Neural Network for Fault Diagnosis on Tennessee-Eastman Process. *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, 745–748. https://doi.org/10.1109/ICMLA.2015.208

Yang, Y., Chen, Y., Chen, X., & Liu, X. (2012). Multivariate industrial process monitoring based on the integration method of canonical variate analysis and independent component analysis. *Chemometrics and Intelligent Laboratory Systems*, *116*, 94–101. https://doi.org/10.1016/j.chemolab.2012.04.013

Yin, S., Ding, S. X., Haghani, A., Hao, H., & Zhang, P. (2012). A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark Tennessee Eastman process. *Journal of Process Control*, *22*(9), 1567–1581. https://doi.org/10.1016/j.jprocont.2012.06.009

Yin, S., Ding, S. X., Xie, X., & Luo, H. (2014). A review on basic data-driven approaches for industrial process monitoring. *IEEE Transactions on Industrial Electronics*, *61*(11), 6414–6428. https://doi.org/10.1109/TIE.2014.2301773

Zemel, R. S., & Hinton, G. E. (1994). Developing Population Codes By Minimizing Description Length. *Neural Computation*, *7*, 11–18. Retrieved from http://www.cs.toronto.edu/~fritz/absps/nips93.pdf

Zhang, Y. (2009). Enhanced statistical analysis of nonlinear processes using KPCA, KICA and SVM. *Chemical Engineering Science*, *64*(5), 801–811. https://doi.org/10.1016/j.ces.2008.10.012

Zhao, H., Sun, S., & Jin, B. (2018). Sequential Fault Diagnosis Based on LSTM Neural Network. *IEEE Access*, *6*, 12929–12939. https://doi.org/10.1109/ACCESS.2018.2794765