Karlsruhe Institute of Technology

Master's Thesis

DNN classification and forecasting of simulated energy consumption and charging behavior for electric vehicles on the island of la Réunion

Jiacheng Yao

Handover Date: 30.05.2016

Reviewers:	Prof. Alexander Waibel
	Prof. José Juan Hernández-Cabrera
Supervisors:	Dr. Jochen Wendel
	Dr. José Évora
	Dr. Kevin Kilgour

Department of Informatics
Karlsruhe Institute of Technology

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen, als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe.

Ort, den Datum

Zusammenfassung

Als eine der wichtigsten Aufgaben auf Gebiet von kunstlicher Intelligenz, gilt Klassifizierung bis jetzt noch als ein hochaktuelles Thema in Forschung und Industrie. Zum anderen stellt Zeitreihenprognose eine eher größere Herausforderung dar und kommt zur Anwendung in Statistik, Finanz, Physik und vielen anderen Forschungsfelder.

Um das erste der zwei erwähnten Problemen zu lösen, zahlreiche Untersuchungen sind durchgeführt worden über die letzten paar Jahrzehnte. Darunter haben die tiefen neuronalen Netze (DNN) sich hervorragend bewährt und symbolisieren den neusten Stand der Technik. Zur Zeitreihenprognose ist Autoregressive Integrated Moving Average weithin beherrschend verwendet worden. Aber es weisen viele theoretische und empirische Literaturen darauf, dass die Integration von verschiedenen Modellen zu einem besseren Vorhersagegenauigkeit führen könnte.

In dieser Masterarbeit bemühen wir uns, ein Klassifizierung- und Prognosesystem für die Fahrmuster, Ladeverhalten sowie den gesamte Energieverbrauch von millionen Autos auf die virtuelle Insel der La Réunion. Darüber hinaus werden mehrere unterschiedliche Arten von Neuronalen Netzen analysiert und verwendet zur Zweickmäßigkeitsprüfung. Beim Teil der Prognose wird ein hybridisiertes Modell implementiert, das ARIMA Modell und bidirectionales Long short-term Memory kombiniert.

Außerdem, um das Problem des Energiemangels zu angehen, wird ein Priorisierungsschema entwickelt, das entscheiden kann, welche Autos vor anderem geladen werden sollten bei überbelegten Ladestationen.

Abstract

Classification is a fundamentally crucial task in the field of artificial intelligence and remains one of the hottest topics in research and industry community. Forecasting, on the other hand, is even more challenging and is of a far greater importance in statistics, finance, physics and numerous other branches of study.

To tackle the first of the two tasks above, countless studies have been conducted throughout the last couple of decades. Among all those different approaches in the previous studies, deep neural networks (DNN) have shown the greatest performance and achieved state-of-the-art results in a wide range of machine learning tasks. As for forecasting, even though linear models such as autoregressive integrated moving average (ARIMA) models are the most dominant and widely used, several findings have indicated both theoretically and empirically that integration of different models can, in many cases, yield a better predictive performance.

In this thesis, we endeavor to build a classification and forecasting system for the driving patterns, charging behaviors and the total energy consumption of millions of electric vehicles on the island of la Réunion. Different neural network structures and methodologies are utilized and tested to examine their suitability for the specific tasks. For the forecasting part, a hybridized model combining ARIMA model and bi-directional long short-term memory is implemented.

Moreover, to address the problem of energy resource scarcity, a prioritization scheme is developed, which determines which vehicles should be charged prior to the others when the charging stations are overcrowded.

Acknowledgments

First of all, I would like to express my utmost gratitude to my supervisors, Jochen Wendel at European Institute for Energy Research, José Évora at SIANI of Universidad de Las Palmas de Gran Canaria and Kevin Kilgour at KIT Computer Science faculty. Without Jochen's guidance and support in the last several months, this thesis would have gone on an entirely different direction. José has shown enormous patience and helped me greatly by making the dataset robustly available and providing deep insight into the tasks. Kevin has also been kind enough to share his precious time and keen knowledge on the construction and fine-tuning process of the neural networks. In the meantime, Enrique Kremers and Francisco Marzabal have given me assistance in better understanding the simulation mechanism as well.

I would also thank Alexandru Nichersu, Alexander Simons and other colleagues, who might not have been directly involved with this work but yet still showed immense support.

I am also forever indebted to my loved ones, who have granted me unconditional love throughout the entire process and beyond.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 Motivation

In the last couple of centuries the world has undergone fundamental changes with the advent of industrial and technological revolution. One of the most significant byproduct of those changes is that human beings are consuming energy resources at an ever-increasing pace. For this very reason, many estimate that non-renewable energy reserves could be completely depleted in the near future[ST09]. To avoid this catastrophic possibility, it has been argued that sustainability by increasing the usage of renewable energy and environment-friendly technology is of utmost importance[AB07].

One option to achieve the purpose of sustainability is the transition from fossil fuel vehicles to electric cars[TT13]. Compared to conventional internal combustion engine automobiles, they possess many obvious advantages. For example, electric cars produce significantly less noise, emit no tailpipe pollutants and reduce total greenhouse gas emission by a large margin. Economically, their running costs and other energy related costs are not subject to the prices of fossil fuels, which are not only extremely volatile but also is, in the long term, prone to increase as the fossil fuel reserve dries out. Recently, as their driving range is expanding with help of more powerful and reliable batteries, electric vehicles are drawing more and more attention and a larger share of the automobile market is also to be expected.

However, new problems arise with the massive introduction of renewable energy sources (RES) and the transition from fossil fuel vehicles to electric vehicles. Unlike traditional energy sources, renewable energy such as wind can have great range of fluctuation due to changes in environment condition. This will lead to unstable energy production and energy demand will in turn be more difficult to be satisfied. Therefore, when introducing RES and electric vehicles, it is imperative to provide smart power grid management. For example, we need to figure out the daily, monthly and yearly development of energy consumption, spot the rush hours and come up with adaptive power supply policies so that the power grid system can be more reliable and stable regardless of fluctuations in the environment.

1.2 Contributions

In this thesis, we utilize the electromobility model created by Kremers et. al[Tor+15]. to generate a simulated island full of electric vehicles. The model has been thoroughly discussed by Kremers et.al. in their original paper[Tor+15]. The dataset generated from the model will be discussed in Chapter 4 in detail. The objective of the thesis is to construct a classification and forecasting system, which can:

- classify the driving patterns and charging behaviors of the vehicles,
- make prediction on the total energy consumption of all the cars on the island,
- foretell how one singular vehicle will behave based on historical knowledge present.

In addition, a prioritization scheme is developed, which can output the order in which the vehicles should be charged at the charging stations. This will be instrumental in rush hours when the charging stations are overcrowded and also in other situations when resources are limited and a more delicate coping method is needed.

Other contributions are made in the system construction process. Specifically, state-of-the-art neural networks are utilized in both classification and forecasting parts. One dimensional convolutional neural network is constructed for the classification task. In the forecasting part, a hybridized methodology combining bi-directional long short-term memory and autoregressive integrated moving average (ARIMA) model is developed to achieve a better predictive performance.

Furthermore, the classification, forecasting and the prioritization models can be utilized for analyzing and predicting energy consumption in a complex power grid and can serve as crucial components to construct smart demand side management (DSM) which can flatten peak demand and allow for efficient and flexible energy usage[Évo14].

1.3 Thesis Structure

The remainder of this thesis is structured as follows:

Chapter 2 and Chapter 3 serve the purpose of introducing the theoretical foundations of this work. Fundamental methodologies in neural network research community will be reviewed in Chapter 2, starting from the most basic feed-forward neural networks to the newest bi-directional long short-term memory networks. Afterwards, Chapter 3 reviews the related approaches in the field of time series forecasting.

Chapter 4 starts by describing the problems of interest in this thesis and inspecting the electromobility dataset. After that, the methodologies adopted in this work will be introduced, which is divided into three parts, classification, time series forecasting and the prioritization scheme.

The evaluation of the utilized methodologies can be found in Chapter 5. Extensive experiments are conducted and their results are visualized and analyzed.

We conclude this thesis with Chapter 6, which summarizes the work and discusses the future prospects.

2 Neural Networks

As this work is based mostly on neural networks, it is only fair that we begin it with a review of the most relevant methodologies of neural networks, from the most simplistic *perceptron*[Ros57] to the state-of-the-art *bi-directional long short-term memory*[GS05].

Research in the field of neural networks took flight in the year of 1943, when McCulloch and Pitts introduced the first model of artificial neurons in [MP]. Ever since then, the principal motivation has not yet changed, namely to emulate the structure and the computational process of the human brain. This is inspired by the observation that the human brain is far more superior to any von Neumann computer when facing numerous cognitive tasks, despite its comparatively low speed of serial computation. Therefore, the main differentiating factor is not the processing speed, but the organization of the processing.

So how does the human brain perform information processing? The oversimplified answer would be *parallelism, adaptability* and *self-organization.* The human brain contains approximately $10^{11} - 10^{12}$ elementary nerve cells called *neurons.* This biological neural network is essentially a collection of *interconnected* neurons that compute and generate impulses. Each neuron, connected to 1000 other neurons on average, can be *activated* by inputs from elsewhere and can stimulate other neurons as well. Thanks to the vast number of neurons, the complex interconnections and the *parallel* way in which simple operations are carried out simultaneously, the human brain can cope with complex cognitive tasks very quickly. Furthermore, as a person grows older and gains more and more experience each passing day, the brain also makes adaptations on its own by assimilating the new knowledge or perspectives and re-organizing the structure of the neural network accordingly.

2.1 Perceptron

To emulate the biological neuron network in the human brain, Frank Rosenblatt, an American psychologist, proposed the *perceptron* algorithm in [Ros57] at the Cornell Aeronautical Laboratory. In a nutshell, a perceptron is a linear classifier, the architecture of which is that of Figure 1. Each input vector $\mathbf{x}$ is of the same dimension, here $\mathbf{x} = (1, x_1, x_2, \ldots, x_m)$. Moreover, each input vector is associated with a target output value t. An activation function f takes the variable $\mathbf{x}$ as input and generates a binary output y through the following formula:

$$y = f(\sum_{i=0}^{m} w_j x_j) = f(\mathbf{w}^T \mathbf{x}), \tag{2.1}$$

where $\mathbf{w}$ is the weight vector, also $(m+1)-$dimensional. The training set is denoted by $D = \{(x^{(i)}, t^{(i)}), i = 1, \ldots, N\}$. The error criterion is denoted by $E(\mathbf{w}) = \frac{1}{2}\sum_{x \in X}(t_x - y_x)^2$.

The training procedure of the perceptron algorithm consists of the following steps:

(i) Parameter initialization and selection. this includes the weights $\mathbf{w}$, the learning rate η and the threshold γ.

(ii) Iterate:

- calculate the current output: $y(t) = f(\mathbf{w}^T(t)\mathbf{x})$.
- update $\mathbf{w}$ with the delta rule: $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$, where $\Delta\mathbf{w} = -\eta\nabla E(\mathbf{w})$.

The following theorem as proven by Rosenblatt et.al. in 1962[JJ62] guarantees the convergence of the perceptron algorithm:

Theorem 2.1 (Perceptron Convergence Theorem)**.** *If* $D = \{(x^{(i)}, t^{(i)}), i = 1, \ldots, N\}$ *describes a linearly separable dichotomy, then the fixed-increment perceptron algorithm terminates after a finite number of weight updates.*

Originally, after the perceptron algorithm was introduced, much attention was drawn to this simple yet powerful learning approach. However, in 1962, Minsky and Papert in [MP69] demonstrated that perceptrons do not possess the ability to learn an XOR function. After this publication, research on artificial neural networks fell into a period of recession.

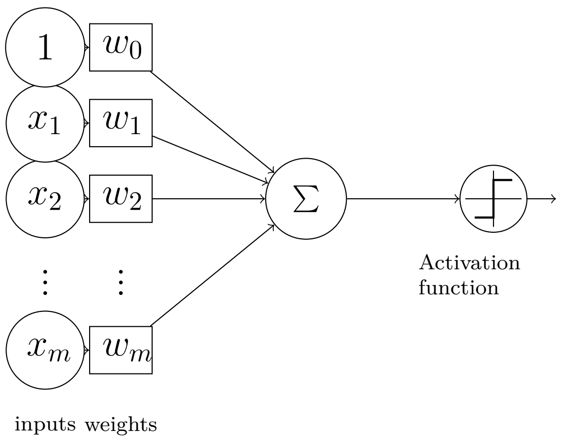

Figure 1: A simple (single-unit) perceptron[Ros57]

2.2 Feedforward Neural Networks

In comparison to perceptrons, feedforward neural networks (FNN) consist of a input layer, at least one hidden layer and a output layer. Thanks to the more complicated topology, FNN is able to solve learning tasks that are not linearly separable. The architecture of a FNN is displayed in Figure 2. As we can see, the information passes only in one direction, from the input layer via hidden layers to the output layer.

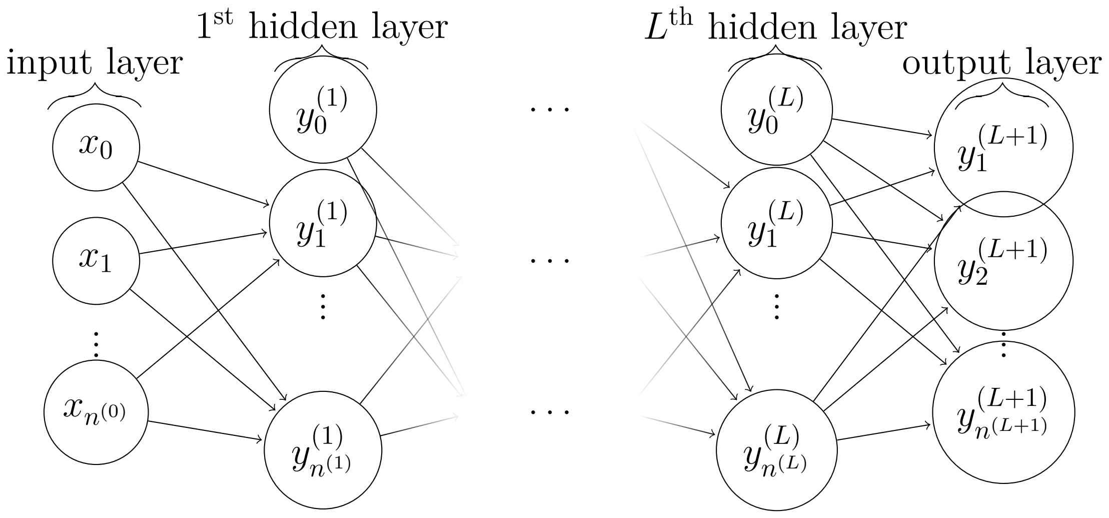

Figure 2: Network graph of a $(L+1)$-layer perceptron with $n^{(0)}$ input units and $n^{(L+1)}$ output units. The l^{th} hidden layer contains $n^{(l)}$ hidden units

As in the perceptron algorithm, an activation function is needed when we pass the information down to the next layer and want to determine whether the neuron should be activated or not. There are many shapes of activation functions available, respectively suitable for various kinds of tasks. For FNN, the most commonly used activation functions are the *sigmoid* function $\sigma(x)$, depicted in Figure 3, and the *hyperbolic tangent function* $\phi(x)$ in Figure 7. Their general forms are as follows:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.2}$$

$$\phi(x) = \frac{2}{1+e^{-2x}} - 1 \tag{2.3}$$

The major characteristics of those two functions is that they are nonlinear, differentiable and their input will be mapped into a certain area. Take sigmoid function for example. Its output is "squashed" between 0 and 1. For specific learning tasks like classification, this feature is quite desirable. However, for other tasks, like time series forecasting, a linear activation function is the most suitable choice, as we want the normal, non-normalized values as output in this case.

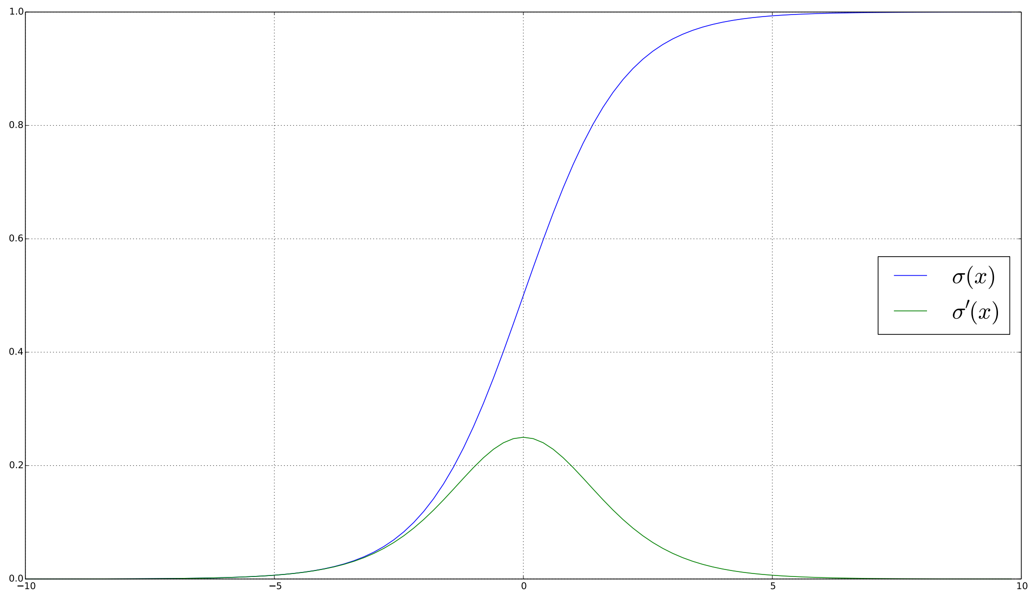

Figure 3: Sigmoid and its first order derivative

Furthermore, when facing the challenge of classification for $K > 2$ classes, the so-called *softmax* function is often applied at the output layer of FNNs. Softmax function normalize the output values to be between 0 and 1 and has the effect of making the output value of the most likely class to be close to 1 and the rest 0. Just like the sigmoid function or the hyperbolic tangent function, the softmax function is differentiable. The general form of the softmax function is as follows:

$$\phi(a_j) = \frac{e^{a_j}}{\sum_k e^{a_k}} \tag{2.4}$$

The process of adjusting the weights in the neural network and producing the correct outputs for the inputs is called *training*. For feedforward neural networks, this is achieved with the *backpropagation algorithm*[RHW86].

In the training process of the FNN with backpropagation algorithm, we start by initializing the weights with small random values. Afterwards, each propagation can be decomposed in the following four steps:

(i) Feed-forward computation,

(ii) Backpropagation to the output layer,

(iii) Backpropagation to the hidden layers,

(iv) Weight updates.

The four steps are to be repeated until the error criterion $E(\mathbf{w})$ has converged, which is defined as follows:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{x \in X} \sum_{k \in outputs} (t_{kx} - o_{kx})^2 \tag{2.5}$$

The weight w_{ji} from input i of node j is updated again with the delta rule: $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$, where $\Delta w_{ji} = -\eta \frac{\partial E_x}{\partial w_{ji}}$. This time, different from the update method in the perceptron algorithm, we need to consider two cases for the weight update.

If the weight w_{ji} is between the last hidden layer and the output layer, then $\Delta w_{ji} = -\eta \frac{\partial E_x}{\partial w_{ji}} = \eta (t_{jx} - o_{jx}) o_{jx} (1 - o_{jx}) x_{ji}$. Otherwise, $\Delta w_{ji} = -\eta \frac{\partial E_x}{\partial w_{ji}} = \eta o_j (1 - o_j) \sum_{k \in Downstream(j)} \delta_k w_{kj} x_{ji}$.

As the backpropagration algorithm is based on the gradient descent, which is generally a slow process and takes a long time to converge, many researches have been conducted in the effort to speed up the training process. A detailed review can be found in Chapter 8: Fast Learning Algorithms of [Roj96]. An overview of some algorithms to improve convergence speed of feedforward neural networks is provided below:

(i) Stochastic gradient descent (SGD), which has been discussed in detailed in [Bot12] and many other publications. It is a simplified version of gradient descent algorithm and generally results in faster convergence.

(ii) AdaGrad[DHS11], first proposed in 2011, has gained considerable popularity since its publication. It is also a variant of SGD algorithm and can converge very fast on convex error surfaces.

(iii) RMSprop[TH12] is the modification of AdaGrad algorithm, which introduces a decaying factor.

Additionally, since neural networks are prone to the overfitting problem, which means that the network has learned well enough with the training set but fails to generalize well later on when facing unseen data. This problem can also be handled with help of cross-validation and some other more advanced techniques. Recently, neural networks with dropout training[Sri+14] are favored to address this issue, which allows the network to randomly drop units along with their connections during the training process. This has proven to significantly reduce overfitting and many state-of-the-art findings have been obtained as a result.

2.3 Convolutional Neural Networks

Traditional feedforward neural networks have already been used and had some successes in vision-related machine learning subfields. However, as shown in Figure 2, the neurons in a feedforward neural network are fully connected with each other. This full connectivity gives rise to problems like curse of dimensionality and poor scalability to high resolution images. To tackle these issues, researchers turn to biological visual cortex for inspiration. In 1968, Hubel et. al. discovered that "the cortex is seen as a system organized vertically and horizontally in entirely different ways. In the vertical system (in which cells lying along a vertical line in the cortex have common features) stimulus dimensions such as retinal position, line orientation, ocular dominance, and perhaps directionality of movement, are mapped in sets of superimposed but independent mosaics. The horizontal system segregates cells in layers by hierarchical orders, the lowest orders (simple cells monocularly driven) located in and near layer IV, the higher orders in the upper and lower layers."[HW68] All in all, convolutional neural networks (CNN) are a way to exploit the spatial information enclosed in natural images. As follows, the fundamental building blocks in a CNN are presented.

2.3.1 Convolution Operation

Convolution, in the most abstract sense, between functions f and g, is written $f * g$ in literature. It is given by:

$$\begin{aligned}(f * g)(t) &\equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \\ &= \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau\end{aligned} \tag{2.6}$$

A convolution on an image I, which is viewed as a matrix with dimension of $n_1 \times n_2$, with a filter K is defined by:

$$(I * K)_{r,s} := \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} K_{u,v} I_{r+u,s+v} \tag{2.7}$$

in which the filter K is also a matrix:

$$K = \begin{pmatrix} K_{-h_1,-h_2} & \dots & K_{-h_1,h_2} \\ \vdots & K_{0,0} & \vdots \\ K_{h_1,-h_2} & \dots & K_{h_1,h_2} \end{pmatrix}. \tag{2.8}$$

It should be noted that special care needs to be taken towards the borders of the image. The following filter, named as the *discrete Gaussian filter*[FP02], could be used for smoothing:

$$\left(K_{G(\sigma)}\right)_{r,s} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{r^2+s^2}{2\sigma^2}\right) \tag{2.9}$$

When it comes to complicated tasks, several types of layers serving different purposes are usually stacked[CMS12; KSH12]. A most simplistic convolutional network (for CIFAR-10 classification challenge[Kri09] as example) could consist of the following five sorts of layers:

(i) *Input layer (INPUT)* takes in the raw pixel values from the dataset, in CIFAR-10's case with width of 32, height of 32 and three color channels R, G, B.

(ii) *Convolutional layer (CONV)* computes the output of the convolution operation, which in turn will be used as input for the next layer (usually a rectification layer).

(iii) *Rectified Linear Units (ReLU)* or the rectification layer applies an elementwise activation function.

(iv) *Pooling layer (POOL)* performs a downsampling operation on the previous layer and result in a dimensionally smaller representation of the original data.

(v) *Fully connected layer (FC)* serves as the final layer and could be considered as a special sort of "softmax" function that computes the scores for the different classes.

The above mentioned different kinds of layers in a convolutional neural network are described in detail as follows.

2.3.2 Convolutional Layer (CONV)

The CONV layer is the vital building block in a convolutional neural network. In the CONV layer, the usage of learnable *kernels* or *filters* is of utmost im-

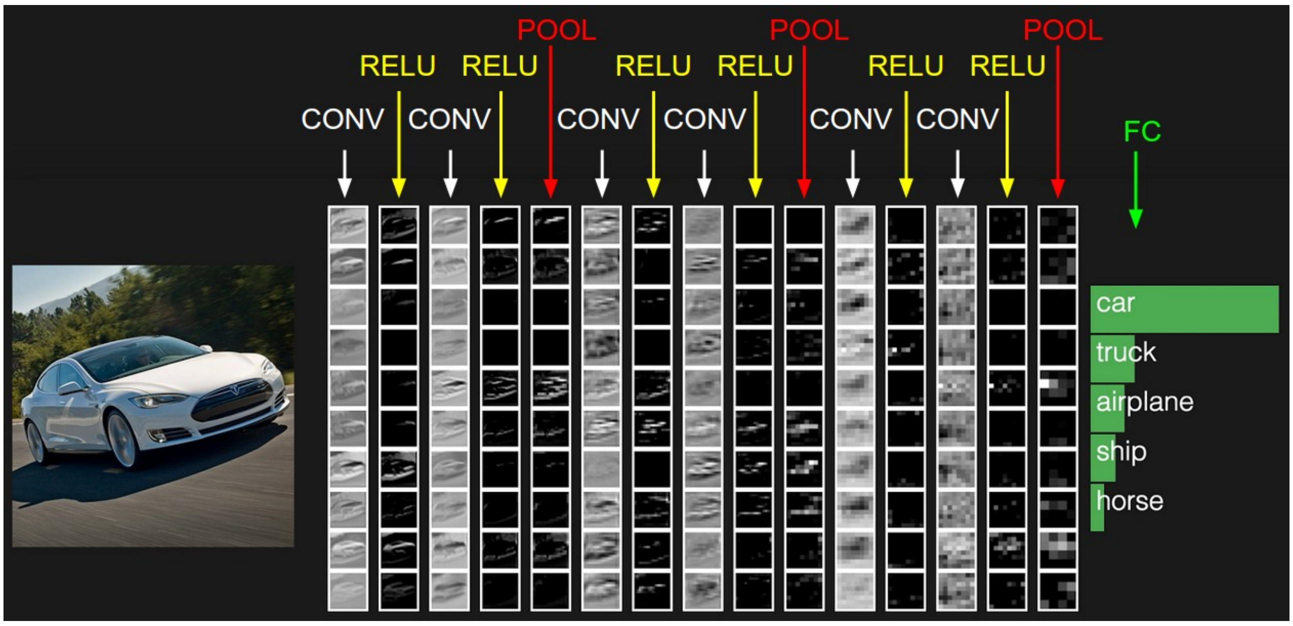

Figure 4: Activations of an example ConvNet architecture [CS231n Stanford]

portance. The kernels, although spatially small compared to the input image, extend through the full depth of the input volume. When we pass input data onto a convolutional layer, a convolution operation is conducted between the filter and the input volume, resulting in a 2-dimensional activation map. The activation maps can be visualized and used to interpret the performance of the network and thus to improve the results. A detailed discussion regarding that can be found in [ZF13]. Some relevant denotations and the formal way to compute the output of a CONV layer are as follows:

A convolutional layer l takes $n_1^{(l-1)}$ feature maps from layer $(l-1)$ as input, each of which has the size of $n_2^{(l-1)} \times n_3^{(l-1)}$. In the special case when $l = 1$, the raw data (images, videos, audios etc.) are accepted as input. In the end, the layer l gives out $n_1^{(l)}$ feature maps of size $n_2^{(l)} \times n_3^{(l)}$ as output. Each output feature map, in form of a two-dimensional array, consists of $n_2^{(l)} \cdot n_3^{(l)}$ entries. The entry at position (r, s) in the i^{th} feature map $Y_i^{(l)}$, is computed as

$$\begin{aligned}\left(Y_i^{(l)}\right)_{r,s} &= \left(B_i^{(l)}\right)_{r,s} + \sum_{j=1}^{n_1^{(l-1)}} \left(K_{i,j}^{(l)} * Y_j^{(l-1)}\right)_{r,s} \\ &= \left(B_i^{(l)}\right)_{r,s} + \sum_{j=1}^{n_1^{(l-1)}} \sum_{u=-h_1^{(l)}}^{h_1^{(l)}} \sum_{v=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{i,j}^{(l)}\right)_{u,v} \left(Y_j^{(l-1)}\right)_{r+u,s+v},\end{aligned} \tag{2.10}$$

where $B_i^{(l)}$ represents the bias matrix and $K_{i,j}^{(l)}$ the filter between the j^{th} feature map in layer $(l-1)$ and the i^{th} feature map in layer l [LKF10].

In essence, the CONV layer tries to learn specific features at spatially local positions of the input. The kernels will "fire up" or *activate* when those specific features are observed. After stacking all the activation maps corresponding to all the kernels used in the convolutional layer, the full output volume of the CONV layer will be calculated. One element in the output volume can be regarded as the activation output of a visual neuron that has a limited **receptive field size**. Or in other words, this particular neuron will only be responsible for visualizing a small region in the original image. This process, imitating the character of *local connectivity* in biological visual systems, is the essence of how convolutional neural network manages to share weights and to avoid inefficient training of traditional fully connected artificial neural networks.

It can be concluded that a fully connected neural network is simply impractical. This can be demonstrated with the following example. If we take a RGB-colored image of size $32 \times 32 \times 3$ as the input for the network, a single neuron in a traditional fully neural network will have $3,072$ weights. To avoid this problem, neurons in a convolutional layer are only connected to a small local region of the input volume. This local region is often called the **receptive field**[Ben09]. Moreover, it should be noted that, even though the connections in space along width and height of the input image are limited into the receptive field, a full connection along the depth of the input is always expected. So in the previous example, a single neuron in a CONV layer has only $6 \times 6 \times 3 = 108$ weights, provided that the receptive field has a size of 6×6.

In addition to the local connectivity, three other hyperparameters can also be manipulated to further reduce the complexity of the neural network. They are listed and discussed as follows[ON15]:

(i) **Depth:** In standard artificial neural networks, all of the neurons in the hidden layers are connected to all of the neurons in the previous layer. In convolutional layers however, we add a new hyperparameter called the **depth**, which allows us to decide the number of neurons that are connected to the same region of the input volume. So essentially, depth is a way to reduce the number of connections or weights in the network model. When a convolutional neural network is trained, all the different neurons will learn to "fire up" for different features from the input layer. A very intuitive example can be seen in the activation maps of a ConvNet for the MNIST dataset[LeC+98]. We can see that different neurons in the convolutional layer will activate when facing up different sorts of shapes. Also, as common practice, a group of neurons that are all connected to the same region in the input layer can be referred to as a *depth column*. Importantly, we should be aware that due care should be taken when selecting the proper depth for a CONV layer, as reducing the *depth* of a layer can lead to a significant reduction of the total number of neurons of the model on one hand, and can cause a worsening learning capabilities for the classifier on the other hand.

(ii) **Stride:** To control the allocation mechanism of the depth columns around the spatial dimensionality of the input, the hyperparameter of **stride** is introduced. For instance, if the stride is set to 1, then the receptive field will be heavily overlapped and very large activations will be produced. Alternatively, a higher stride will lead to less overlapping receptive fields and in turn an output volume of lower spatial dimensions.

(iii) **Zero-padding:** Just like its name suggests, **zero-padding** is the simple process of padding the input with zeros on the border of the input volume. By controlling the size of zero-padding, we can have better control as to the spatial size of the output volumes.

If the input volume size is denoted by $V_{in} = height \times width \times depth$, the size of the receptive field by R, the size of the zero-padding set by Z and the stride by S, then the spatial dimensionality of the convolutional layer output, denoted by V_{out}, can be determined with the following formula:

$$V_{out} = \frac{(V_{in} - R) + 2Z}{S + 1} \tag{2.11}$$

If the resulting V_{out} is not integer, then it means that the stride has not been correctly set. And if we set zero-padding to be $Z = \frac{R-1}{2}$ and the stride S to be 1, then we have $V_{out} = V_{in}$, which means that the spatial sizes remain constant after CONV layer. This has couple of advantages:

(i) It is easier to manage the sizes for the later on POOL layers, which alone should be responsible for down-sampling the volumes spatially. Otherwise, if $S > 1$, then it would be very tricky to keep track of the volume sizes throughout the CNN structure, especially when very deep CNN structure is utilized.

(ii) It turns out in practice that smaller strides yield better performance, even though the previous compromise about the stride holds in general.

Even after the aforementioned hyperparameters and local connectivity are introduced into the network model, then number of weights involved in many real-world cases is more often than not unacceptably high. This is why further methods like **parameter sharing** have been developed to reduce the number of parameters into a more reasonable range.

Parameter sharing has a very simple intuition: if a regional feature is useful for computation at one spatial region, then it should also be useful in another region. If the same weights and bias are used in each individual activation map within the output volume, then the number of the parameters in the layers can be reduced drastically.

Rectified Linear Units (ReLU). A Rectified Linear Units (ReLU) layer or rectification layer l takes $n_1^{(l-1)}$ feature maps from layer $(l-1)$ as input, each of which has the size of $n_2^{(l-1)} \times n_3^{(l-1)}$. This layer applies activation function and serves the purpose of increasing the nonlinear properties. It has been shown in several experiments[Jar+09] that ReLU layer is crucial for achieving quicker training process and better performance. The results of the layer l are computed as follows:

$$Y_i^{(l)} = \left|Y_i^{(l)}\right|, \tag{2.12}$$

where the absolute value is computed elementwise and the amount and the sizes of the feature maps remain unchanged:

$$n_1^{(l)} = n_1^{(l-1)} \tag{2.13}$$

$$n_2^{(l)} \times n_3^{(l)} = n_2^{(l-1)} \times n_3^{(l-1)} \tag{2.14}$$

2.3.3 Pooling layer (POOL)

One major issue with traditional multilayer perceptron structure is its inclination towards overfitting, mainly due to its full connectivity and the huge amount of parameters that come along with it. To avoid this issue, the concept of *pooling* is introduced in convolutional neural network. Very often, pooling layers are inserted between convolutional layers periodically in a CNN framework. Essentially, a pooling layer serves as a basic form of downsampling process and results in a spatially smaller representation of the original data that is robust when facing noise and distortions. The basic idea behind the POOL layer is that the exact spatial information of a feature is less important than its rough relative location.

A Pooling layer l takes $n_1^{(l-1)}$ feature maps from layer $(l-1)$ as input. If each feature map has the size of $n_2^{(l-1)} \times n_3^{(l-1)}$, and we conduct pooling with a 2×2 filter and the stride equals 2, then the pooling layer outputs $n_1^{(l)} = n_1^{(l-1)}$ feature maps, each of which has the size of $\frac{n_2^{(l-1)}}{2} \times \frac{n_3^{(l-1)}}{2}$. In every 2×2 non-overlapping window, we keep one out of four element value as the output for the window. Several pooling and other subsampling units are stated as follows:

(i) **Max pooling:** as the most popular pooling method, max pooling outputs the maximum for each window.

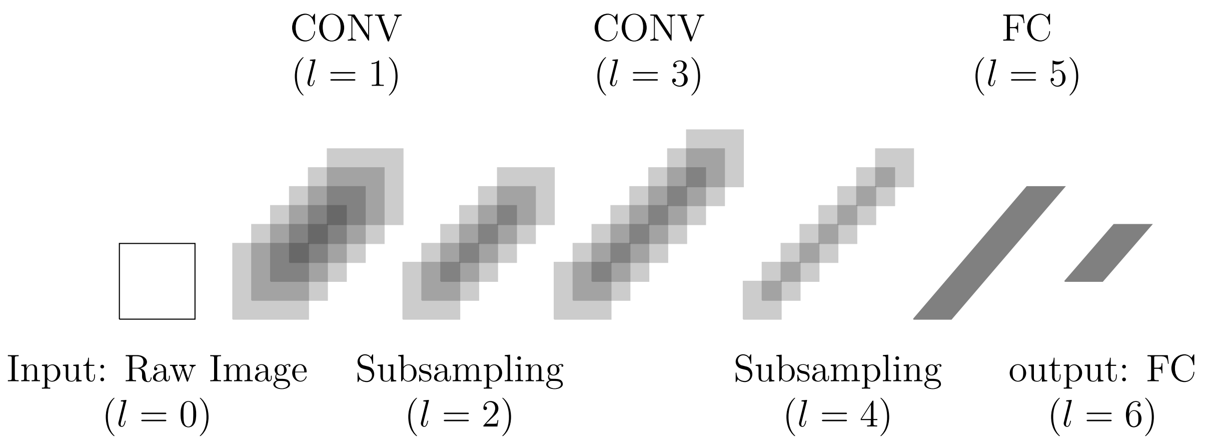

Figure 5: The architecture of a typical convolutional neural network, modified from [ON15]. The input image data goes through several pairs of convolutional layer, rectification layer and subsampling layer, just as depicted in the figure, in which CONV stands for convolutional layer including rectification, POOL for pooling layer and FC for fully connected layer. In the end, after several rounds of CONV-ReLU-POOL, the data goes through fully connected layer(s), which will work as the high level classifier and uses softmax activation functions to output the final results like class scores.

(ii) **Average pooling:** instead of using MAX operation, this pooling method uses the average of each window as the output.

(iii) **Skipping:** a even more straightforward method is to skip a constant amount of elements in horizontal as well as in vertical direction.

Fully connected layer (FC). As its name suggests, the neurons in a fully connected layer (FC) l are fully connected to all the neurons in the previous layer $(l-1)$. The output of the i^{th} neuron in this layer is computed with the following equation:

$$Y_i^{(l)} = f\left(\sum_{j=1}^{m_1^{(l-1)}} w_{i,j}^{(l)} \left(Y_j^{(l-1)}\right)\right). \tag{2.15}$$

2.3.4 Architectures

In recent years, convolutional neural networks follow certain layer patterns to allow for a better streamlined architecture designing procedure[Sze+14]. The general idea behind the most of the layer patterns is:

(i) to let the original data go through an arbitrary amount of CONV-RELU layer pairs, which can exploit the local information hidden in the original data and introduce non-linearities at the same time,

(ii) to input the data into POOL layers, or subsampling layers in a more general sense, which reduces the amount the parameters and avoids over-fitting problem,

(iii) to go through FC layers, which calculate the final outputs.

2.3.5 Applications

Interestingly, one of the earliest applications of convolutional networks was found in 1989, when Waibel et.al. developed the *time-delay neural network* (TDNN)[Wai+90] and successfully implemented it on speech recognition. In the performance evaluation of their original paper, a superior recognition rate of **98.5%** has been achieved in comparison to **93.7%** with traditional hidden markov models. In principle, time-delay neural network and convolutional neural network have a lot in common when it comes to network parameter reduction. Due to page length constraint as well as the similarity between TDNN and CNN, the details of TDNN will not be discussed here in this thesis.

Although the fundamental framework of CNNs has been introduced in the 1980s, one of more recent and the most significant success for convolutional neural networks was in 2012, when Alex Krizhevsky et.al.[KSH12] managed to train a large and deep CNN on the ImageNet dataset[Rus+15] and obtained **16.4%** of error rate, whereas the next best non-CNN model achieved only **26.2%**. The reason for the late blooming has less to do with the model itself than with the advancements in computation speed with help of GPU computing and the fact that substantially more data is available nowadays than just twenty years ago.

Since these early breakthroughs, convolutional networks have drawn a lot of attention in the machine learning community and works as the foundation of many sophisticated and very successful systems. Again take ImageNet challenge as example, in 2014, CNNs have achieved human-level performance[Rus+14].

Even in the field of time series analysis, convolutional networks have also been applied and have given the researchers a refreshingly new way to conduct studies. In 2014, Zhang et.al.[Zhe+14] proposed a novel deep learning framework and utilized the framework in time series classification problem and have shown that CNNs has the advantage of efficiency in comparison to other state-of-the-art approaches and demonstrates competitive accuracy performance as well. The main reason for their revelations is that a time series, in essence, can be regarded as a one dimensional image. As it generally holds that the historical information as well as the future is relevant for time series analysis, we can think of the this "informational continuity" as informational locality, which can be exploited with help of convolutional neural networks.

It is for this very reason that we design a deep convolutional network framework for the time series classification problem presented in this thesis. The experiments in Chapter 5 show that deep CNN frameworks do manage to achieve superior performance in comparison to other models.

2.4 Recurrent Neural Networks

Similar to convolutional neural networks, recurrent neural networks (RNN) are based on the simple observation that human beings excel at processing sequential data because of feedback connections between the neurons in the brain. Principally different from other traditional machine learning algorithms, recurrent neural networks are more suited for problems such as natural language processing, word prediction, speech processing, time series prediction etc, where the objective is to find the mapping between arbitrary input sequences and output sequences[Lip15]. Even though RNN is computationally more challenging and represents a more difficult task than its feedforward counterparts, considerable improvements have been accomplished during the last two decades along with new network architectures and advanced computation techniques. The training of RNNs is becoming more and more efficient and its performance has also improved greatly[Lip15]. Compared to other related models, the advantages of RNNs can be highlighted as follows:

(i) Hidden Markov Models (HMM):

   a) RNNs possess greater representational power,

   b) RNNs do not rely on the Markov assumption and can deal with long-term, dependencies

   c) HMMs have no continuous internal states.

(ii) Feedforward Neural Networks (FFNNs): FFNNs have no internal states, no feedback connections.

(iii) Support Vector Machines (SVMs): Same as FFNNs.

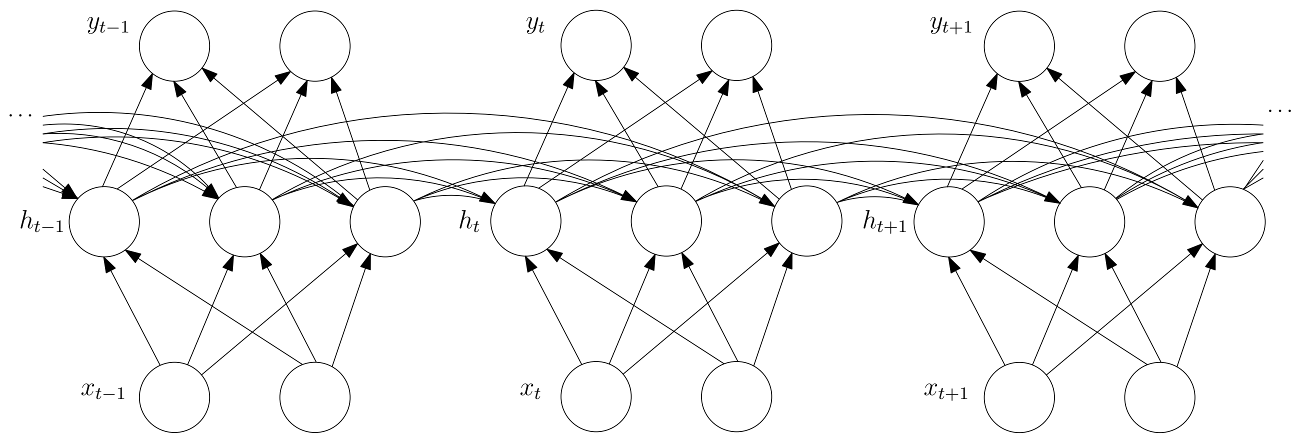

Figure 6: Architecture of a simple recurrent neural network

As stated above, a recurrent neural network aims to find the mapping between arbitrary input sequences and output sequences. Given an input sequence

$(x_1, \ldots, x_T)$, the weight matrices between input and hidden layers, hidden and hidden layers, hidden and output layers, respectively denoted by W_{xh}, W_{hh}, W_{hy}, the bias vectors for input, hidden and output layers, respectively denoted by b_y, b_h, h_0, the recurrent network calculates the sequences of hidden states and outputs with the following steps:

Algorithm 1: RNN Output Computation

```
input : input sequence $(x_1, \ldots, x_T)$
output: output sequence $(y_1, \ldots, y_T)$

1 for $t \leftarrow 1$ to $T$ do
2     $h_t \leftarrow f(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$
3     $o_t \leftarrow W_{hy}h_t + b_y$
4     $y_t \leftarrow g(o_t)$
```

where $f(\cdot)$ and $g(\cdot)$ stand for the activation functions for the hidden layers and output layer. The loss of the network is calculated with:

$$L(y; target) = \sum_{t=1}^{T} L(y_t; target_t) \tag{2.16}$$

With the backpropagation through time algorithm[Wer90], the parameters of a RNN can be computed with the following steps:

Algorithm 2: RNN Parameter Update

```
1 for $t \leftarrow T$ to $1$ do
2     $do_t \leftarrow g'(o_t) \cdot dL(y_t; target_t)/dy_t$
3     $db_y \leftarrow db_y + do_t$
4     $dW_{hy} \leftarrow dW_{hy} + do_t h_t^\top$
5     $dh_t \leftarrow dh_t + W_{hy}^\top do_t$
6     $dy_t \leftarrow f'(y_t) \cdot dh_t$
7     $dW_{xh} \leftarrow dW_{xh} + dy_t x_t^\top$
8     $db_h \leftarrow db_h + dy_t$
9     $dW_{hh} \leftarrow dW_{hh} + dy_t h_{t-1}^\top$
10    $dh_{t-1} \leftarrow W_{hh}^\top dy_t$
```

In 1989, the *universal approximation theorem* was proven by Hornik et.al., which states that standard feedforward networks with only a single hidden layer are universal approximators[HSW89]. Similarly, for recurrent neural networks, it holds that “a recurrent neural network can be trained to approximate any non-linear dynamical system with any accuracy, given that the network has an appropriate initial condition and enough hidden units”[FN93]. With the publication of these two theoretical findings, the computational power and its expressiveness can also be proven. A even more impressive characteristic of recurrent networks was discovered by Siegelmann et.al. in 1991[SS91]:

Theorem 2.2 (Turing Completeness of RNNs (informal))**.** *There exists a finite neural network built of neurons with sigmoidal activation functions that can simulate any turing machine.*

However, with great computational power comes also a critical drawback. It has been known that the training process of recurrent networks is impractically slow due to the *vanishing* and the *exploding* gradient problems, which has been discussed in great detail by Bengio et.al. in 1994[BSF] and Pascanu[PMB12].

The vanishing and exploding gradient problems. The two above mentioned issues come up as the standard recurrent neural networks have difficulties when dealing with long-term dependencies. Formally speaking, the successful prediction of an output at time point t is dependent on the input at an much earlier time point $\tau \ll t$[BSF].

When long-term dependencies are present and we try to conduct a backpropagation all the way through so many time steps with help of the chain rules, the gradients can be calculated with the following formulas. After rewriting and reviewing the formulas, we can better understand why long-term dependencies are the reason for the two fundamental problems of RNNs:

$$\frac{\partial L}{\partial \theta} = \sum_{1 \leqslant t \leqslant T} \frac{\partial L_t}{\partial \theta}, \tag{2.17}$$

$$\frac{\partial L_t}{\partial \theta} = \sum_{1 \leqslant i \leqslant t} \left(\frac{\partial L_t}{\partial h_t} \frac{\partial h_t}{\partial h_i} \frac{\partial h_i}{\partial \theta} \right), \tag{2.18}$$

$$\frac{\partial h_t}{\partial h_i} = \prod_{t \geqslant j > i} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{t \geqslant j > i} W^\top diag(f'(h_{i-1})), \tag{2.19}$$

where θ stands for the parameter set in the network which consists of the weight matrices and the bias vectors. As Figure 7 depicts, the derivatives at the two ends of the hyperbolic tangent function are close to zero. Also, the gradients will shrink at an exponential rate after multiple rounds of matrix multiplication, when the elements in the Jacobian matrix W in Equation 2.19 are small. On the other hand, when the elements in W are big, the *exploding gradient problem* will come up.

Several possible solutions have been published to address the exploding and vanishing gradient problem and are summarized as follows:

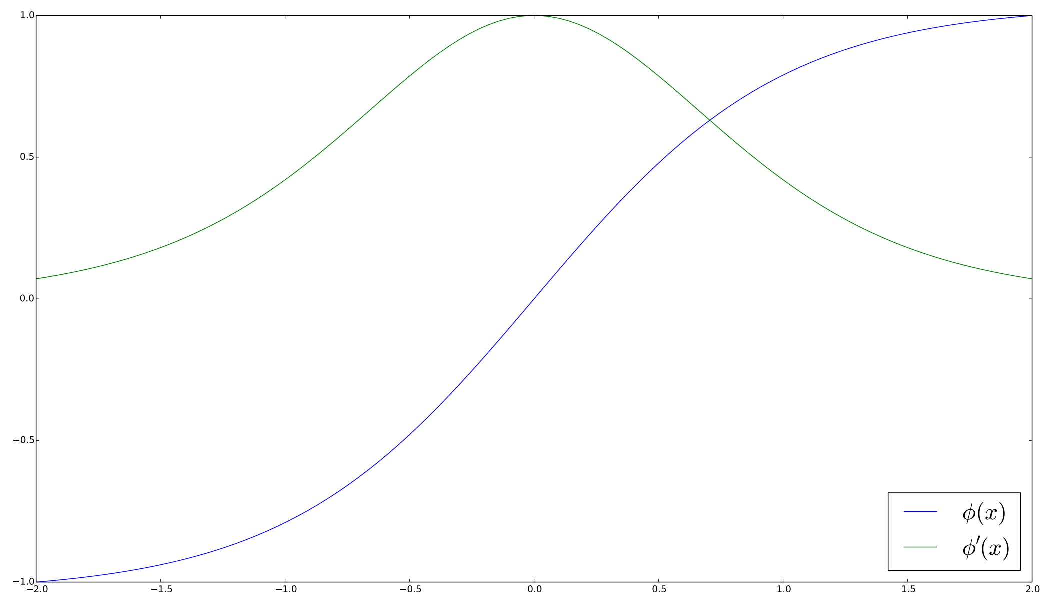

Figure 7: Tanh and its first order derivative

(i) A most early approach for the exploding gradient problem is the *truncated backpropagation through time (TBPTT)* algorithm[WZ89]. It stops the gradient propagation when a threshold for maximum number of time step is reached. This is a compromise between capability of learning long term memory and scale management of the gradients.

(ii) Usage of $L1$ or $L2$ penalty on the weights can alleviate the exploding gradient problem, but costs the network to lose the ability to deal with long term memory.

(iii) Jaeger and Haas introduced Echo State Networks in 2004[JH04], which need only the weights of the output layer as its parameters in the model.

(iv) Teacher forcing[Jae02] requires a target for neurons in the hidden layers at each time step. This method can be used to address the exploding gradient problem and also speeds up convergence.

(v) Another very simplistic method to tackle the exploding gradient problem is to rescale the weights if they increase up to a predetermined threshold[PMB12].

(vi) Pascanu et.al. proposed a gradient regularization scheme to address the vanishing gradient problem[PMB12].

(vii) Long short-term memory[HS97] recurrent neural network was proposed by Hochreiter and Schmidhuber in 1997. This architecture can handle long term dependencies much more efficiently and have been utilized in numerous applications. The following subsection will discuss long short-term memory in more detail, the theoretical foundation for this thesis.

2.5 Long Short-term Memory

As stated in the previous subsection, the training algorithm for traditional recurrent neural networks is impractically inefficient due to the well known *vanishing and exploding gradient* problem. Many studies have been conducted to address this problem and some of them have been very successful. Amongst the newer recurrent network architectures, *long short-term memory* [HS97] and *bidirectional recurrent neural networks*[SP97] are considered two works of groundbreaking significance. In the next two subsections, these two frameworks will be discussed, as they are the direct theoretical foundations of this thesis.

The long short-term memory (LSTM) model was introduced in 1997 by Hochreiter and Schmidhuber for the purpose of tackling the vanishing gradient problem. One major difference between LSTM model and traditional recurrent neural network is the concept of *memory cell*, which is depicted in Figure 8. In standard RNN model, there are two types of memory passing through the network. First type is the *long-term memory* which is stored in the weight matrices and is updated constantly. The second type, the *short-term memory* refers to the activations from neurons to neurons in the following layer. By introducing the notion of memory cell, LSTM manages to store a third type of memory, the *long short-term memory*. As a matter of fact, this is also the origin of the term LSTM. The following equations describe how to calculate a hidden state $h^{(t)}$ when the current input data and previous hidden state are provided[Lip15]:

Input gate:

$$i^{(t)} = \sigma(W^{ix}x^{(t)} + W^{ih}h^{(t-1)} + b_i) \tag{2.20}$$

Input node:

$$g^{(t)} = \phi(W^{gx}x^{(t)} + W^{gh}h^{(t-1)} + b_g) \tag{2.21}$$

Forget gate:

$$f^{(t)} = \sigma(W^{fx}x^{(t)} + W^{fh}h^{(t-1)} + b_f) \tag{2.22}$$

Output gate:

$$o^{(t)} = \sigma(W^{ox}x^{(t)} + W^{oh}h^{(t-1)} + b_o) \tag{2.23}$$

Internal cell state:

$$c^{(t)} = g^{(t)} \odot i^{(t)} + c^{(t-1)} \odot f^{(t)} \tag{2.24}$$

Hidden layer state:

$$h^{(t)} = \phi(c^{(t)}) \odot o^{(t)} \tag{2.25}$$

Several remarks on the equations to help understand how long short-term memory model works are presented in the following:

(i) $\odot$ in Equations 2.24 and 2.25 stands for pointwise multiplication.

(ii) The input, forget and output gates, respectively denoted by $i^{(t)}$, $f^{(t)}$, $o^{(t)}$, are novel concepts that were not present in other traditional neural network architectures. We can better understand these gates by seeing them as the control panel for a indoor heating system. The room temperature can be controlled by how wide open the gates are.

(iii) The input node $g^{(t)}$ is the equivalent of the hidden state in standard recurrent neural networks. In some literature, the input node is denoted by $\widetilde{c^{(t)}}$. This is because the input node is also regarded as the candidate for the internal cell state $c^{(t)}$. In LSTM's case, the hidden state is dependent on the combination of gated input node and gated previous cell state as well as the output gate. Therefore, the input gate decides how much role the input node $g^{(t)}$ plays in the forward passing procedure.

(iv) The internal cell state $c^{(t)}$ is the quintessential part in the memory cell. This is where the long short-term memory is stored. Observing the equation 2.24, we can appreciate that current internal cell state depends on both the previous internal cell state $c^{(t-1)}$ and the current input $g^{(t)}$. After input gate and forget gate are introduced into the memory cell, we can better adjust which of the two should be more relevant for specific sequential data.

(v) The forget gate $f^{(t)}$ was proposed by Gers et.al. in 1999[GSC99]. The purpose of introducing forget gates into LSTM model is to address the issue that standard long short-term memory algorithm performs very poorly when dealing with continual input streams. Even though the forget gate was only presented two years after the publication of the original LSTM[HS97], it has been proven to be a very effective component and is included in most of the modern LSTM architectures.

Another very important concept in modern LSTMs is the *peephole* connections, introduced also by Gers et.al. in 2000[GS00]. Peephole connections can pass information from the internal state $c^{(t)}$ to the input gate $i^{(t)}$ and the output gate $o^{(t)}$ directly. To their surprise, the variation has shown great learning performance and is said to be suited for challenging tasks where interval measurement is required.

It has been shown by many studies that long short-term memory recurrent networks is much superior to standard recurrent neural networks when facing sequence learning that has long term dependencies. This is why this model has been a research focus for the last decade and also why so many variations have been published. A thorough review and comparison for the several important LSTM variations can be found in LSTM: A Search Space Odyssey[Gre+15] by Greff et.al.

Amongst the numerous variations of LSTMs, the *gated recurrent unit* (GRU) [Chu+14] is one of the simplest and has drawn much attention. Even though it possesses a relatively simpler architecture than standard LSTMs, experiments[Chu+14] have suggested that GRU displays competitive performance in comparison to LSTMs while being more computationally efficient. A hidden state $h^{(t)}$ is calculated with the following equations[Chu+14].

Update gate:

$$z^{(t)} = \sigma(W^{zx}x^{(t)} + W^{zh}h^{(t-1)} + b_z) \tag{2.26}$$

Reset gate:

$$r^{(t)} = \sigma(W^{rx}x^{(t)} + W^{rh}h^{(t-1)} + b_r) \tag{2.27}$$

Input node:

$$g^{(t)} = \phi(W^{gx}x^{(t)} + W^{gh}(r^{(t)} \odot h^{(t-1)}) + b_g) \tag{2.28}$$

Hidden layer state:

$$h^{(t)} = (1 - z^{(t)})h^{(t-1)} + z^{(t)}g^{(t)} \tag{2.29}$$

As seen in the equations, the GRU model is quite similar to LSTM model. They both rely on the concept of gates to control the flow of information. Whenever we want to calculate a hidden state with a GRU or LSTM unit, the value of the cell unit is always the combination of previous information and new input data, shown through the similarity of Equations 2.24 and 2.29. Differently from LSTMs however, GRU merges the forget gate $f^{(t)}$ and input gate $i^{(t)}$. Also, the output of a GRU unit is not controlled with a output gate $o^{(t)}$ as in LSTM's case. All in all, a GRU model has a simpler structure and is for this reason more efficient.

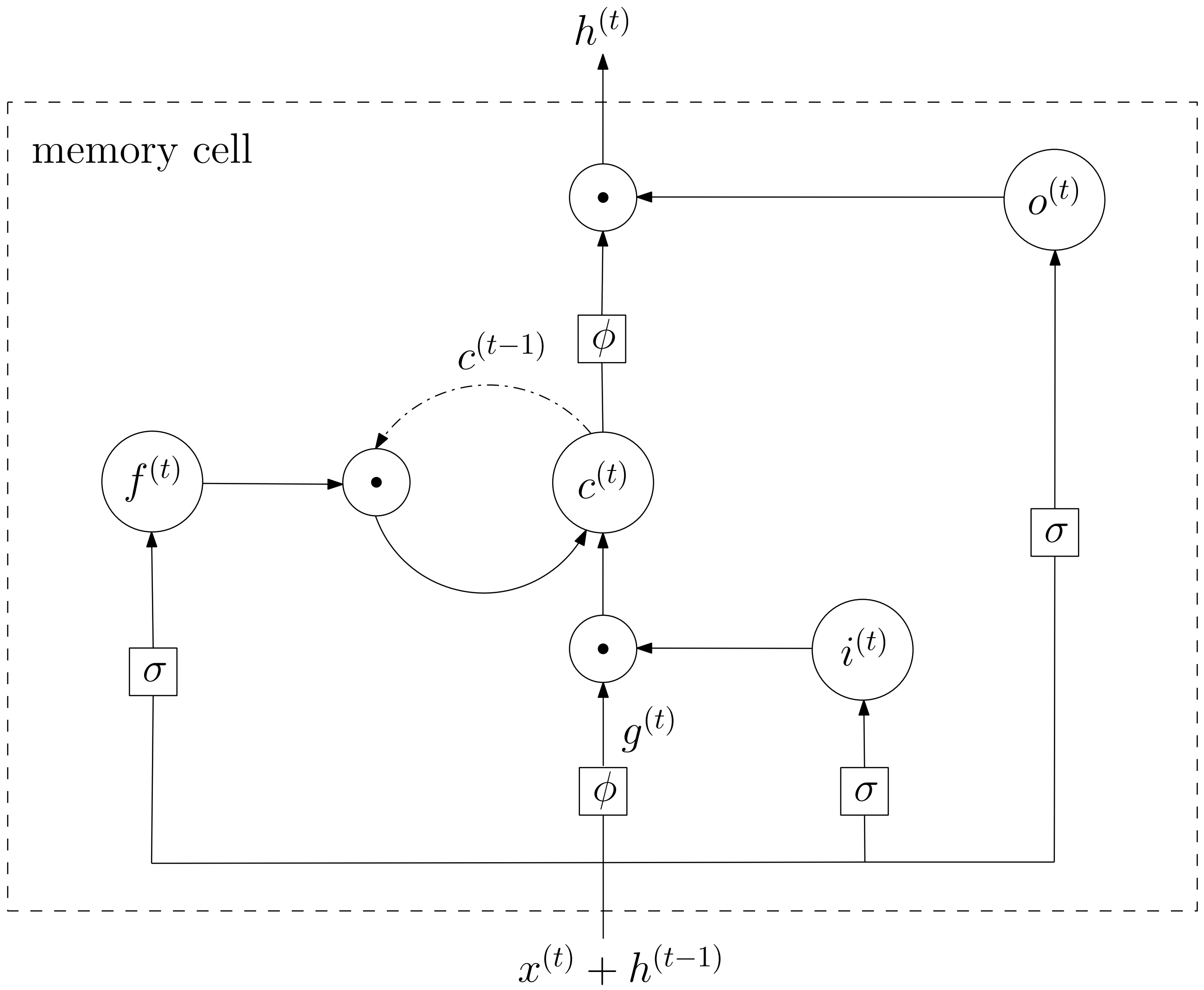

Figure 8: LSTM memory cell

2.6 Bi-directional Long Short-term Memory

Bi-directional recurrent neural network (BRNN)[SP97] is another important architecture. It was first proposed by Schuster et.al. in 1997. The intuition behind this model is that not only the previous information but the future should be taken into consideration for sequence learning. With a standard one-directional RNN, sequential information passes only in one direction, from timepoint $t = 1$ to the end $t = T$. To allow for backwards information flow, Schuster et.al. proposed that two types of hidden layers should be added in the recurrent network. A simple bi-directional RNN structure is depicted in Figure 9.

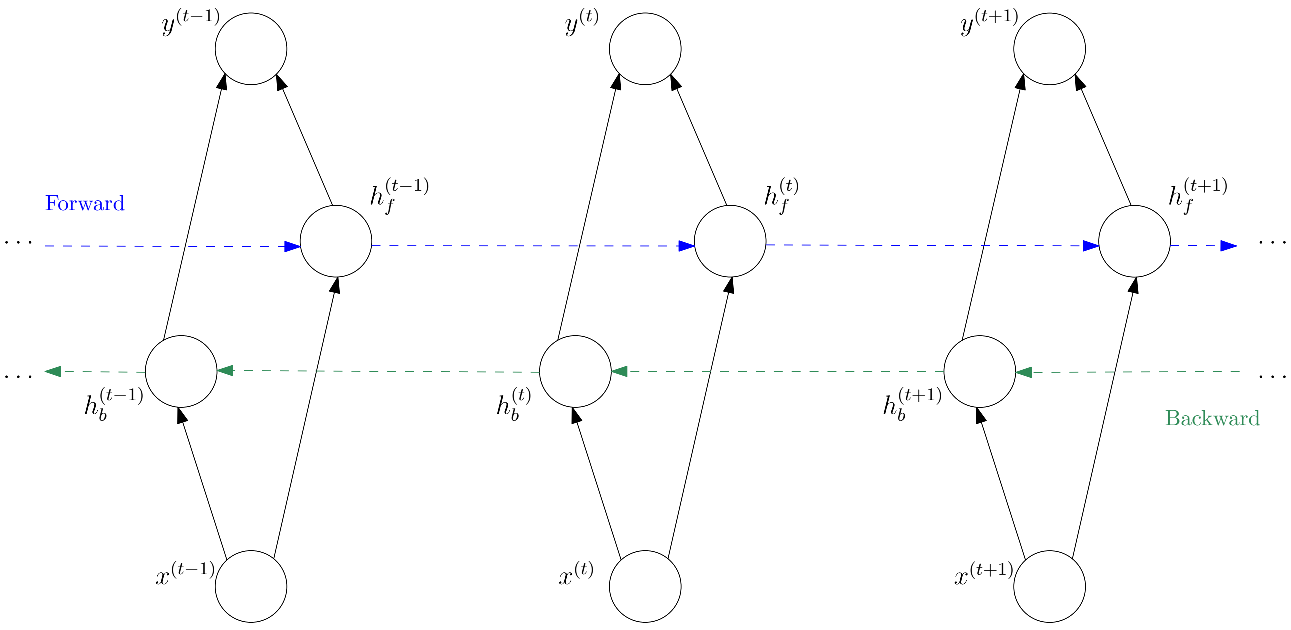

Figure 9: Architecture of a simple bi-directional recurrent neural network

As we can see from the figure, a simple three layer RNN with one hidden layer becomes a four layer bi-directional RNN with two hidden layers, in which the first hidden layer h_f is responsible for the forward direction of the sequence, and the second hidden layer h_b for the backward direction. Both h_f and h_b are directly connected to the input layer and the output layer. Although seemingly complicated, a bi-directional RNN can still be trained with standard backpropagation after it is unfolded across time. The hidden states in a bi-directional RNN can be calculated with the following equations:

$$h_f^{(t)} = \sigma(W^{h_f x} x^{(t)} + W^{h_f h_f} h_f^{(t-1)} + b_{h_f}) \tag{2.30}$$

$$h_b^{(t)} = \sigma(W^{h_b x} x^{(t)} + W^{h_b h_b} h_b^{(t+1)} + b_{h_b}) \tag{2.31}$$

After the calculation of the hidden states is finished, the output state can be calculated as well:

$$y^{(t)} = f(W^{h_f y} h_f^{(t)} + W^{h_b y} h_b^{(t)} + b_y) \tag{2.32}$$

where the function f stands for the final activation function. If the goal is classification, then we can use Softmax function in its place.

It is important to note that bi-directional RNN has its own disadvantages too in spite of its potential. The purpose of BRNN is to exploit both historical and future information. But for online problems, where the future sequences are not yet available, it is simply not possible to train the BRNN model.

After bi-directional recurrent network was proposed by Schuster et.al. in 1997, it was viewed as inefficient just like one-directional recurrent neural networks. Since LSTMs became widely utilized, BRNN also began to draw more and more attention. In 2005, Graves et.al. came up with the *bi-directional long short-term memory*[GS05]. In essence, the only difference between a BRNN and a bi-directional LSTM architecture is that, in BLSTM, LSTM units are used instead of simple hidden layer neurons that are made from connections with the previous layer and activation functions as in BRNN's case. Ever since its publication, state-of-the-art results have been reported with bi-directional LSTMs on handwriting recognition[GS05] etc.

3 Time Series Forecasting

In this chapter, we will discuss some of the previous approaches on time series forecasting, including linear models like autoregressive integrated moving average[Bar71], and more complicated hybrid models[Zha03].

Time series is conventionally defined as a sequential data vector or scalar in a certain time period. Formally, a time series $\mathbf{x}$ which depends on time t is denoted by $\mathbf{x} = \{x_0, x_1, \dots, x_t, \dots\}$. As for *time series forecasting*, it is the research field that aims to find a suitable model f to predict the data values at some future time point(s):

$$\tilde{x}_{t+s} = f(x_{t-1}, \dots) \tag{3.1}$$

where s stands for the *horizon* of the forecasting model. As the only information available in a forecasting problem setting is the historical data $\{x_{t-1}, \dots\}$, we see that the model f is actually a function mapping historical data to future data. A lot of research studies have been dedicated to time series forecasting and considerable improvements have been made. However, as it is always challenging to predict the future with limited data and computational resources, this field remains one of the hottest research topics that interests researchers in various completely different fields using different approaches.

A very straightforward idea that is the fundamental intuition of many forecasting modeling methods, which are categorically called *linear models*, is that the future could and should be a linear combination of the historical information. For example, one of the most simplistic model is the so-called *random walk* model:

$$\tilde{x}_t = x_{t-1} + \epsilon_t \tag{3.2}$$

where ϵ_t denotes the i.i.d. error term. In random walk model, one believes that the best guess for the data value at the next time point is the current data, which is surprisingly banal yet has outperformed some other more complicated models in some very volatile situations like in exchange rate estimation in the last few decades[KT01].

Another important linear model that has been very popular and dominant in time series forecasting is the autoregressive integrated moving average (ARIMA) model. This model follows the famous Box-Jenkins methodology[BJ90] and even overlaps with exponential smoothing models[Mck84], which is another class of simple and widely used models. ARIMA model is a generalization of ARMA model, which stands for autoregressive moving average model. Its major advantages are comprehensive incorporation of many types of models and capability of capturing noise, trend and season component in a time series etc. For example, autoregressive model, moving average model and ARMA model are all subsets of ARIMA model. However, as ARIMA is essentially a linear model which believes in the linear correlation between previous observations and future values, it lacks in the much needed expressiveness when it comes to more complicated and nonlinear modeling. This is the main reason why many researchers turn their attention to developing nonlinear models that can fit with real-world problems where many complex factors are at play and the correlation between future and past can not be captured otherwise with a linear model.

Throughout the history of time series forecasting research, various types of approaches have been proposed. The traditional linear models such as random walk, exponential smoothing and the autoregressive integrated moving average model have been applied for decades and have had some limited success when the time series are simple enough to be described with a linear model. Even though they are very easy to comprehend and to implement, people in forecasting community are focusing more and more on nonlinear models to overcome the insufficient expressiveness of linear models. Some important nonlinear models are listed as follows:

(i) Bilinear model[GA78] was introduced by Granger et.al. in 1978 and is regarded as one of the most natural generalization of linear time series models. This can be shown with the definition of a (p, q, P, Q) order bilinear model:

$$\tilde{x}_t = \sum_{i=1}^{p} b_i x_{t-i} + \epsilon_t + \sum_{j=1}^{q} a_j \epsilon_{t-j} + \sum_{i=1}^{P} \sum_{j=1}^{Q} c_{ij} x_{t-i} \epsilon_{t-j}, \tag{3.3}$$

where one believes in the assumption that the best guess for the next observation is the weighted combination of previous observations $x_t - i$, error terms ϵ_{t-j} and the product of historical observations and error terms $x_{t-i}\epsilon_{t-j}$.

(ii) Threshold autoregression (TAR) model[Ton83] can be viewed as "multi-regime generalization" of autoregressive (AR) models, which consists of k AR parts or regimes and can be defined as follows:

$$\tilde{x}_t = \sum_{i=1}^{k} \{b_{i0} + b_{i1} x_{t-1} + \cdots + b_{i,p_i} x_{t-p_i} + \sigma_i \epsilon_t\} I(x_{t-d} \in A_i), \tag{3.4}$$

where $k > 1$, d denotes the delay, and $I(A)$ stands for an indicator function such that $I(A) = 1$ if event A occurs and $I(A) = 0$ if otherwise.

(iii) Autoregressive conditionally heteroscedastic (ARCH) model[Eng82] was introduced by Engle et.al. in 1982. It was proven to be very successful for daily finance data but has not yet found wide applications in other fields. The definition of a $p \geqslant 1$ order ARCH model is as follows:

$$\tilde{x}_t = \sigma_t \epsilon_t \tag{3.5}$$

$$\sigma_t^2 = c_0 + b_1 x_{t-1}^2 + \cdots + b_p x_{t-p}^2, \tag{3.6}$$

where $c_0 > 0$ and $b_i \geqslant 0$.

(iv) Neural Networks: as an algorithm designed originally for machine learning problems, neural networks have been applied in time series forecasting[ZPH98] since 1990s and has shown its superiority to traditional statistical forecasting models thanks to its data-driven characteristics, generalizing ability, nonlinearity and its expressiveness[HSW89].

The basics for ARIMA models and neural network based forecasting models are presented in the following subsections.

3.1 Autoregressive Integrated Moving Average

In comparison to an ARMA model, an autoregressive integrated moving average model has another parameter d standing for *differencing*, which, if $d = 1$, is the simple concept of computing the differences between data values at two consecutive timepoints: $\nabla^{d=1} \equiv (x_t - x_{t-1})$. For $d > 1$, the difference can be denoted with help of the *backward shift operator* $\mathbf{B}$:

$$\nabla^d \equiv (1 - \mathbf{B})^d, \tag{3.7}$$

where $\mathbf{B}x_t = x_{t-1}$. By definition, a time series $\mathbf{x} = \{x_0, x_1, \ldots, x_t, \ldots\}$ follows an ARIMA(p, d, q) model if the dth differences of the time series follow an ARMA(p, q) model. The ARIMA model with order of (p, d, q) can now be defined as:

$$\theta_p(\mathbf{B})(1 - \mathbf{B})^d x_t = \phi_q(\mathbf{B})\epsilon_t, \tag{3.8}$$

where θ_p and ϕ_q respectively stand for polynomials of orders p and q. The formula above seems complicated and unintelligible, but still conveys the main idea of the ARIMA model, which says that the best guess of a variable can be obtained with a linear combination of last p previous observations and q random errors. This is exactly the same as in ARMA model's case. Also, once again, the random errors ϵ_t for all timepoints are independently and identically distributed, which have also a mean of 0 and a variance of σ^2.

From Equation 3.8, we can clearly appreciate why ARIMA is considered one of the most generalized and impactful linear models. This model combines the strengths of autoregressive models, moving-average models and the concept of differencing. When we set $p = 0$, then we see that a moving-average model with order of p is a special case of ARIMA model. The same goes for autoregressive models too. When $q = 0$, we see that an AR model with order of q is also a special case of ARIMA model.

Finally, to construct a ARIMA model for a specific time series, it is imperative to find the appropriate order (p, d, q). This can be accomplished with the Box-Jenkins methodology[BJ90], which consists of the following three steps:

(i) Identification: in this step, we try to determine the appropriate order of the ARIMA model. According to Box-Jenkins' initial proposal[BJ90], the autocorrelation function and the partial autocorrelation function can be used for this very purpose. The reason behind this is that autocorrelation as well as partial autocorrelation features can be detected if an ARIMA model holds for a time series. It should be noted that data preprocessing of some sorts is more often than not necessary for the ARIMA model to work. Common practices are to apply differencing and power transformation so that potential trends can be removed and the variance be stabilized.

(ii) Estimation: after the order of the ARIMA is identified, the next step is to use the available data and estimate the parameters by minimizing a error measurement.

(iii) Diagnostic checking: lastly, we check if the prediction of the estimated model is accurate enough.

Usually, the three steps are repeated for several times until an adequately fit model has been found. Afterwards, the final model can be used later for prediction purposes or for gaining deeper insights into the time series.

3.2 Neural Networks

Neural networks have been successfully applied in time series forecasting since early 1990s[TF93; BFC95; FDH01]. It gained its popularity largely due to its data-driven property that does not require any assumption on the desired models and learns purely by training with sample data and to its universal approximation capability[HSW89]. A general framework of network based forecasting model is depicted in Figure 10. Note that the feedforward/recurrent network architecture can be filled with any available network structure like feedforward networks, long short-term memory or even bi-directional LSTMs.

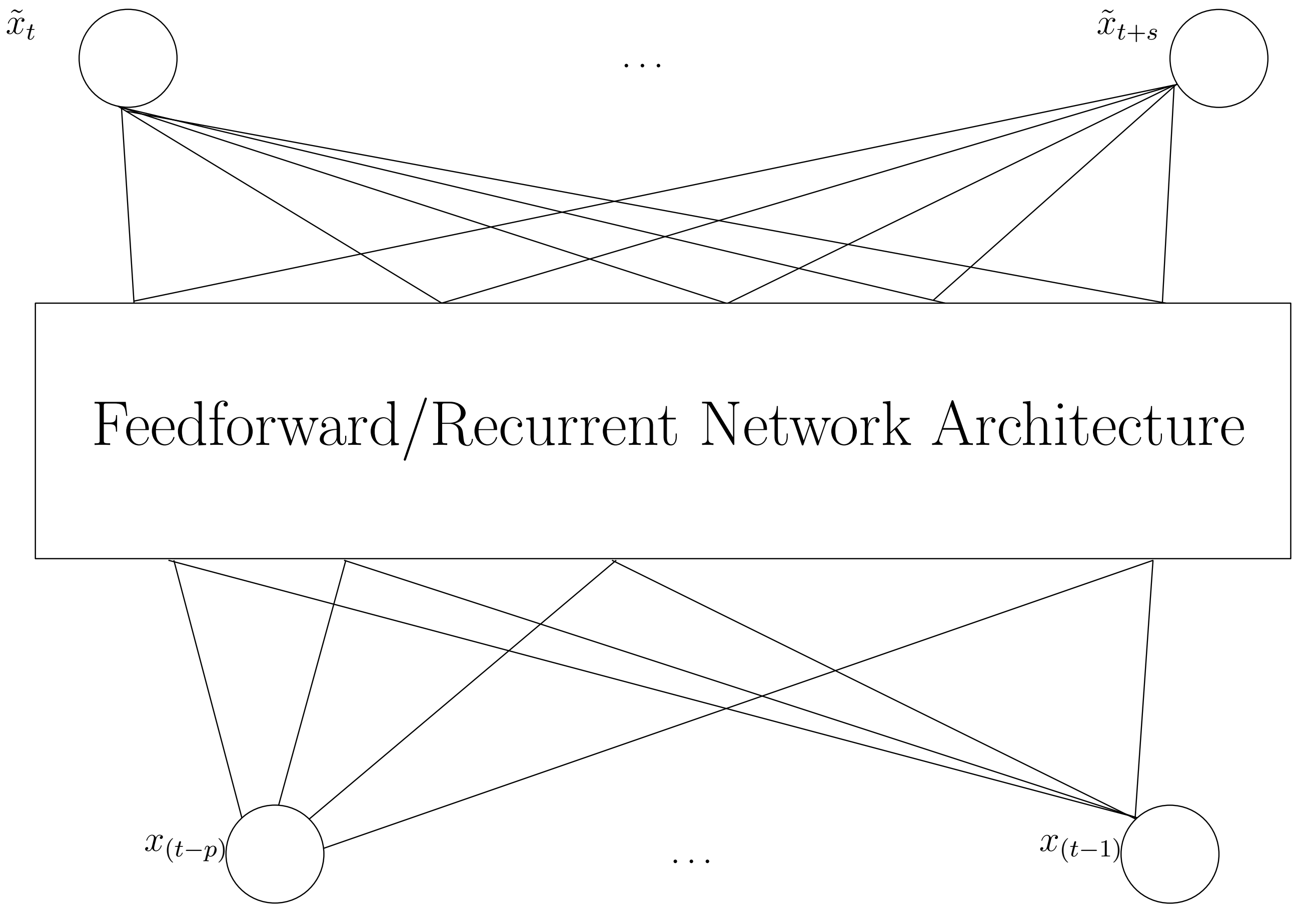

Figure 10: General framework of neural network based time series forecasting model

Just as in ARIMA's case, it is very important to determine the order of the model, which in NN means the number of input nodes and output nodes. The number of the output nodes can be easily determined by the specific requirements in the model building process like how far into future we want to predict. On the other hand, no solid theoretical findings have been published yet regarding the size of the input layer. Therefore, some trial and error experiments are often expected to find a suitable input layer size.

3.3 Hybrid Models

Similar to machine learning, a final predictive decision in forecasting problem setting is often reached by combining several similar or principally different approaches. The motivation behind forecasting method combination is to take advantage of all the single models' unique expressive features and to capture and understand various patterns in complicated real-world time series data. A couple of very early publications on this topic include: The Combination of Forecasts by Bates et.al.[BG01] and another literature under the same name by Winkler et.al. in 1983[Rob83]. Several experimental studies and thorough reviews have been conducted to showcase and discuss the strengths of hybrid models in comparison to simple linear or nonlinear models[Win89; Cle89; Wal11].

Many a hybridizing approach has been introduced since the last several decades. As early as 1963, Barnard has made an empirical argument that the simple average from two forecasting methods has smaller Mean Square Error and therefore should be considered a better prediction[Bar63]. Even until recently, the very simplistic intuition of *simple average* is still one of the most popular combining techniques[Bun85]. Of course, the simple average has trouble outputting fruitful results under certain circumstances. For example, one might argue that it is more reasonable to put different weights on the the forecasts of individual methods as they could have different precisions and have different importance levels. In 1994, Deutsch et.al. proposed a *regime-switching* scheme that allows the weights of different forecasts to change over time[DGT94]. A more interesting and refreshing combining approach was introduced by Fiordalison in 1998[Fio98]. This article is one of the earliest published nonlinear combining techniques, which ingeniously utilized *fuzzy theory* to construct a nonlinear forecast combination system.

In real-world situations, it is very often to encounter a complex time series system that exhibits linear and nonlinear features at the same time. In these cases, a cooperative modular combination will be very instrumental to grasp the full picture of the time series. Hybrid models that integrate some linear model and nonlinear model have been discussed in this effort. It has been shown that the hybridization of ARIMA model and artificial neural networks can yield desirable forecasting performance[Zha03; KBR09; KB11].

One important note regarding the hybrid model scheme planning is that hybrid techniques that are comprised of linear and nonlinear models do not guarantee better estimation results. This phenomenon has been spotted by Terui et.al.[TD02] and by Casey et.al.[TC05]. Therefore, it is imperative to understand that prudent model selection is still much needed and combining linear and nonlinear models is not always the answer despite its evident appeal.

Zhang's Hybrid Methodology. In 2003, Zhang et.al. developed the one of the earliest hybrid schemes[Zha03] that integrates the auto-regressive integrated moving average model and artificial neural network model. His original motivation is simple. ARIMAs have been the dominant approach in time series forecasting for several decades and has not yet lost its popularity. Artificial neural networks have intrigued many researchers and have aroused quite some discussions in time series analysis as well thanks to its previously mentioned advantages. However, many experimental studies have shown neither of the two approaches are suitable for every problems, which has been mentioned in Zhang's original paper too[Zha03]. This is very easy to imagine. As a linear model class that assumes linear correlation, ARIMAs is not adequate for nonlinear settings. ANNs, on the other hand, do not always provide with better results for linear problem modeling. For extremely complicated problems in which fully capturing the linearity and nonlinearity is highly unlikely, a hybrid methodology might be a better alternative.

According to Zhang's methodology, a time series $\mathbf{x}$ is comprised of a linear component L_t and a nonlinear component N_t. Also, he assumes that the data value at a certain timepoint t is equal to the linear combination of the two components. Formally, it says:

$$x_t = L_t + N_t \tag{3.9}$$

Zhang's methodology contains two steps. Firstly, we try to model the linearity of the time series. In this step, the linear component L_t should be calculated with the ARIMA model using the available data. The residuals from this step $\mathbf{e}$ will be used for the second step as it contains the nonlinear relationship.

$$L_t = \tilde{L}_t + e_t, \tag{3.10}$$

where $\tilde{L}_t$ denotes the estimated value for the linear component at timepoint t and e_t the residual from the ARIMA model at timepoint t. Traditionally, in pure ARIMA models, residuals are used in the third step of ARIMA procedure to give indication whether the linear model is sufficient or not. After several rounds of Box-Jenkins methodology, the ARIMA will give out the final configuration of the models and it will be served later for predictive purposes. But as mentioned before, in real-world problems, even after final model is presented, the ARIMA might still perform poorly as it will fail to model the nonlinear relationships hidden in the residuals $\mathbf{e}$.

Therefore, we come to the second step of the hybrid methodology, which is responsible for nonlinear modeling. In Zhang's original paper, a feedforward neural network is used, which takes the previous residuals as the input data for the network and outputs the residual at timepoint t:

$$\tilde{e}_t = f(e_{t-1}, e_{t-2}, \ldots, e_{t-n}) + \varepsilon_t, \tag{3.11}$$

where f denotes the forward passing function of the FNN and ε_t the random error. In this step, certain fine-tuning and optimization techniques need to be applied, or ε_t might not be random after all. In the end, the final estimation of Zhang's methodology is determined with the following equations:

$$\tilde{N}_t \equiv \tilde{e}_t \tag{3.12}$$

$$\tilde{x}_t = \tilde{L}_t + \tilde{N}_t. \tag{3.13}$$

3.4 Forecast Evaluation and Accuracy Metrics

When it comes to performance evaluation of a time series forecasting model, many accuracy measurements can be used. A thorough review can be found in [HK06]. The following paragraphs list some commonly used metrics.

Same as before, the data value at timepoint t in a time series $\mathbf{x}$ is denoted as x_t. The forecast of x_t is $\tilde{x}_t$. Some more relevant definitions are:

(i) Forecast error $e_t = x_t - \tilde{x}_t$,

(ii) Percentage error $p_t = \frac{100 e_t}{x_t}$,

(iii) Relative error $r_t = \frac{e_t}{e_t^*}$, where e_t^* stands for the forecast error of a benchmark forecasting method. It is common practice to use *random walk* model as the benchmark forecasting method.

(iv) Scaled error $q_t = \dfrac{e_t}{\frac{1}{n-1}\sum\limits_{i=2}^{T} |x_i - x_{i-1}|}$

Moreover, the notations of $mean(x_t), median(x_t), gmean(x_t)$ are utilized to symbolize the sample mean, median and the geometric mean values respectively. Categorically, there are five types of forecasting accuracy measurements.

Scale-dependent Metrics: the scale of these accuracy metrics is dependent on the scale of the time series data. This type is particularly useful if we are to compare the performance of different approaches on the same dataset or at least dataset of same type.

$$\mathrm{Mean\,Square\,Error\,(MSE)} = \mathrm{mean}(\mathrm{e_t^2})$$
$$\mathrm{Root\,Mean\,Square\,Error\,(RMSE)} = \sqrt{\mathrm{MSE}}$$
$$\mathrm{Mean\,Absolute\,Error\,(MAE)} = \mathrm{mean}(|\mathrm{e_t}|)$$
$$\mathrm{Median\,Absolute\,Error\,(MdAE)} = \mathrm{median}(|\mathrm{e_t}|)$$

Percentage Error based Metrics: their major advantage against scale-dependent metrics is their scale-independence, which allows them to be used for performance evaluation across different types of datasets. However, they have also an annoying weakness. If the data value x_t at any timepoint $1 \leqslant t \leqslant T$ is equal to 0, then the percentage error based metrics will be infinite.

$$\mathrm{Mean\,Absolute\,Percentage\,Error\,(MAPE)} = \mathrm{mean}(|\mathrm{p_t}|)$$
$$\mathrm{Median\,Absolute\,Percentage\,Error\,(MdAPE)} = \mathrm{median}(|\mathrm{p_t}|)$$
$$\mathrm{Root\,Mean\,Square\,Percentage\,Error\,(RMSPE)} = \sqrt{\mathrm{mean}(\mathrm{p_t^2})}$$
$$\mathrm{Root\,Median\,Square\,Percentage\,Error\,(RMdSPE)} = \sqrt{\mathrm{median}(\mathrm{p_t^2})}$$

Relative Error based Metrics: by comparing the forecasting approach of interest with a benchmark method, the aforementioned issues can be avoided. However, as the forecast error of a benchmark method varies drastically when facing with different datasets, the results for different datasets will lose their significance.

$$\begin{gathered}
\text{Mean Relative Absolute Error (MRAE)} = \mathrm{mean}(|\mathrm{r_t}|) \\
\text{Median Relative Absolute Error (MdRAE)} = \mathrm{median}(|\mathrm{r_t}|) \\
\text{Geometric Mean Relative Absolute Error (GMRAE)} = \mathrm{gmean}(|\mathrm{r_t}|)
\end{gathered}$$

Scaled Error based Metrics:

$$\begin{gathered}
\text{Mean Absolute Scaled Error (MASE)} = \mathrm{mean}(|\mathrm{q_t}|) \\
\text{Median Absolute Scaled Error (MdASE)} = \mathrm{median}(|\mathrm{q_t}|) \\
\text{Root Mean Squared Scaled Error (RMSSE)} = \sqrt{\mathrm{mean}(\mathrm{q_t^2})}
\end{gathered}$$

Relative Measures: Similar to relative error based metrics, this type of accuracy measurements also makes comparison with a benchmark method. Their difference is that relative measures use relative metrics rather than relative errors. So if we take MAE for example, then a relative MAE (RelMAE) can be defined as follows:

$$\mathrm{RelMAE} = \frac{\mathrm{MAE}}{\mathrm{MAE}^*}.$$

4 Methodology

This chapter describes the problems that are to be solved with this thesis and inspect the simulated dataset from the electromobility model in detail. Afterwards, solutions for the listed problems will be proposed.

4.1 Problem definition

The main research tasks of this thesis are the classification and the forecasting of charging and energy consumption of electric vehicles on the island of la Réunion. The specific problems will be described as follows:

3.1.1 Charging and Driving Behavior Classification. Electric vehicles are, in a philosophical sense, the extension of their owners. So it is reasonable to think that the behavioral tendencies of vehicle owners will have major impact on energy consumption patterns the electric vehicles present. For example, some questions such as where a person lives at night and how far away his working place is, and what kind of life style the person represents, they will all play some role in the situation. This is the reason why we would like to analyze and classify the energy consumption patterns of different electric vehicles, try to understand them better and make targeted decisions for those belonging in one pattern and make decisional adjustments for some other. One specific example which is relevant to the second problem in this thesis would be a predictive model that takes advantage of already obtained behavior pattern information and make different forecasting decisions for different sorts of vehicles accordingly.

In this thesis, we will use the simulated electromobility dataset[Tor+15] and design neural network based schemes to classify different driving and charging patterns that are demonstrated by different vehicle groups. A thorough introduction to the original electromobility model that has been used for generating the dataset can be found in [Tor+15].

3.1.2 Energy Consumption Forecasting.

(i) Total Energy Consumption: At a regional, national or even global level, the capability of making reasonable and sufficiently accurate estimation on future energy consumption will be crucial for long-term energy planning. If applied to other fields like politics, finance etc, a reliable predictive tool can bring even greater values. Therefore, we aim to predict total energy consumption in a subgroup in the electromobility dataset using hybrid forecasting models and compare the performance of the hybrid models against the simpler linear and nonlinear counterparts.

(ii) Individual Energy Consumption: Even though the energy consumption of one vehicle will make no significant difference in the grand scheme of things, sometimes it still makes sense to estimate what kind of future behavior one vehicle will show. For example, the third problem this thesis tries to solve is one possible area where a individual energy consumption forecasting scheme can find its application.

3.1.3 Charging Prioritization. As is known to all, we are in a world with limited resources. A simple manifestation of this can be seen also in electric vehicle charging. Imagine we are at rush hour and many electric vehicles need to go to the charging station to recharge the battery for their vehicles. When several vehicles come to a station at the same time and not enough charging slots are available, a charging prioritization scheme is very necessary to alleviate the overcrowding situation as soon as possible. For this reason, we will design a charging prioritization algorithm with help of individual energy consumption forecasting.

4.2 The Dataset

The electromobility simulation model was designed by Enrique Kremers and Johannes Wirges at European Institute for Energy Research in 2014. The model simulates the collective and individual behaviors of electric vehicles on the island of *la Réunion*. In the model, the driving behaviors of vehicles vary according to different choosing of activities. In total, there are six types of behaviors that owners of the vehicles can perform. The behaviors and their corresponding abbreviations are listed below:

(i) Stay at House (H),

(ii) Full Time Job (W),

(iii) Part Time Job (HW),

(iv) Do Shopping (Sh),

(v) Social Recreation (SR),

(vi) Other activities (O).

For each day, each vehicle has a fixed starting location, which is set at home. As different people have different ways of life and choose activities differently, different driving patterns emerge. In the electromobility model, there are 16 possible patterns, based on distribution data originated from real studies. The distribution frequency of all 16 driving patterns can be found in Table 1. Additionally, there are 24 districts on la Réunion (see in Figure 11). All the activity locations for the vehicles are randomly selected following the activity and population density from real geographic data.

Pattern	Probability
H-W-H	23.3%
H-Sh-H	10.5%
H-W-H-SR-H	8.3%
H-SR-H	7.6%
H-Sh-H-SR-H	5.4%
H-W-H-Sh-H	4.8%
H-O-H-O-H	4.8%
H-O-H	4.4%
H-W-SR-H	4.1%
H-W-Sh-H	4.1%
H-O-W-H	3.9%
H-O-H-Sh-H	3.8%
H-HW-SR-HW-H	3.8%
H-W-H-O-H	3.8%
H-HW-Sh-HW-H	3.7%
H-Sh-Sh-H	3.6%

Table 1: Driving pattern frequency distribution

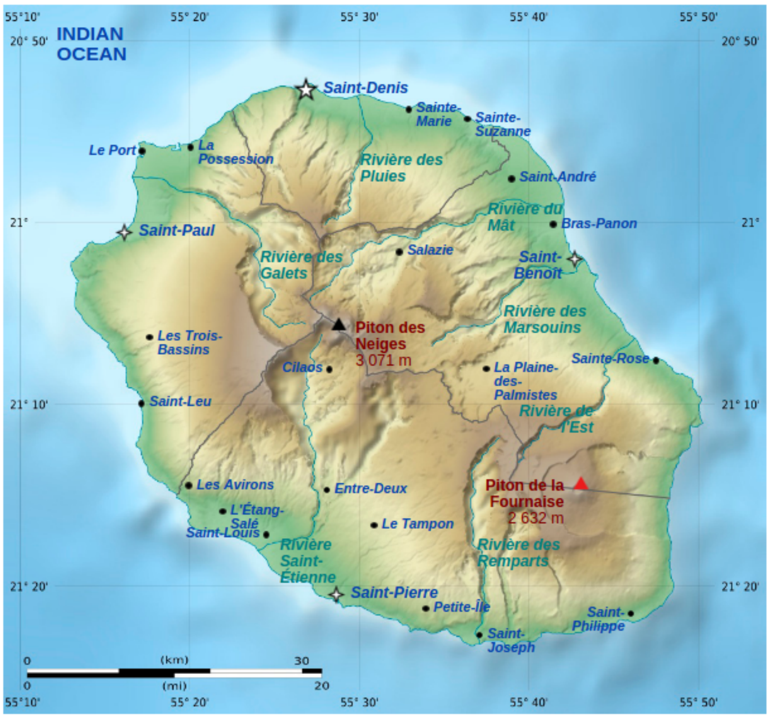

Figure 11: Districts on la Réunion [http://www.reunion.fr/]

In addition to the sixteen driving patterns, there are three possible charging behavior patterns in the dataset. They are:

(i) alwaysCharging, which means charging the vehicle whenever the vehicle has stopped and a charging station is available.

(ii) onlyAtWork, which means only charging the vehicle at workplace.

(iii) onlyAtHome, which means only charging the vehicle at home.

The number of available training (#(Train)) and testing samples (#(Test)), the dimensionalities of input (D_{in}) and output data (D_{out}) for classification and forecasting problems are listed in Table 2.

	#(Train)	#(Test)	D_{in}	D_{out}
Classification	1869903	233744	1440	16/3
Forecasting	648000	72000	TBD	TBD

Table 2: Dataset Size, Input and Output Dimensionality

Some additional remarks: as input data for the classifier, which is neural networks in this thesis, one row of input data corresponds to *one day's energy consumption value* with *one minute* as the smallest unit, resulting in $24 \times 60 = 1440$ timepoints of energy consumption value data for one vehicle in one day. Therefore $D_{in} = 1440$.

As there are 16 driving patterns, $D_{out} = 16$ for driving pattern classification. $D_{out} = 3$ for charging pattern classification.

For the forecasting problem, 500 days of energy consumption data are used. This equals $500 \times 1440 = 720000$ timepoints. The training, testing and validation data have the ratio of $8 : 1 : 1$. The reason why the dimensionality for input and output of forecasting problem is TBD (to be determined) is that they depend on how many historical observations are necessary for a optimal forecasting performance and how far into future we want to look at (one day later, one month later etc) respectively.

4.3 Charging and Driving Behavior Classification

Time series analysis can be deeply impactful in many fields, including finance when we want to make a prediction where the price of a stock will go in the near or distant future, or bioinformatics, where we want to monitor the physical activities of some patients and maybe to spot some underlying patterns and gain keen insights on the patients' current status and make adjustments on the treatment strategy accordingly. The latter example refers to the problem of time series classification. In last few years, various algorithms have been dedicated to solving this problem. Even though distance-based algorithms such as a combination of dynamic time warping (DTW) and k-nearest neighbor (kNN) have been quite popular and successful in this domain[Rak+12], it has been shown that a deep convolutional neural network architecture can accomplish better performance in some datasets[Zhe+14].

In the classification part of this thesis, we construct a deep convolutional network framework while regarding the temporal data as one dimensional images. The rationale behind this is the simple observation that, in the electromobility dataset, the energy consumption value at one timepoint is greatly related to the values at its neighboring timepoints. This is the same as in a image where strong local informational correlation manifests itself and exploitation of the correlation can result in more efficient and accurate algorithms while needing less parameters to describe the model.

The architecture of the deep CNN utilized in the electromobility classification problem is illustrated in Figure 12. As we can see from the Figure, the time series input data go through two CONV-ReLU-CONV-ReLU-POOL rounds and is fed into 2 FC layers (referred to in the figure as "keras.layers.core.Dense") afterwards. In the actual experimentation, we constructed three different CNNs, which have 1, 2 and 3 CONV-ReLU-CONV-ReLU-POOL rounds respectively. Also, a DTW-kNN classifier and a standard feedforward MLP classifier have also been trained and tested to serve as benchmarks. Meanwhile, charging and driving patterns of the vehicles in the electromobility dataset are classified separately with different label settings.

The deep convolutional neural networks are constructed under the Keras environment[Cho15]. Some basic notes on Keras are:

(i) Keras is a minimalist and modular neural network library in Python.

(ii) It can run on TensorFlow or Theano. Just as Theano, a Keras program can run both on CPU and GPU.

(iii) Algorithms in Keras are efficiently optimized and provide best possible results while offering great extensibility and flexibility.

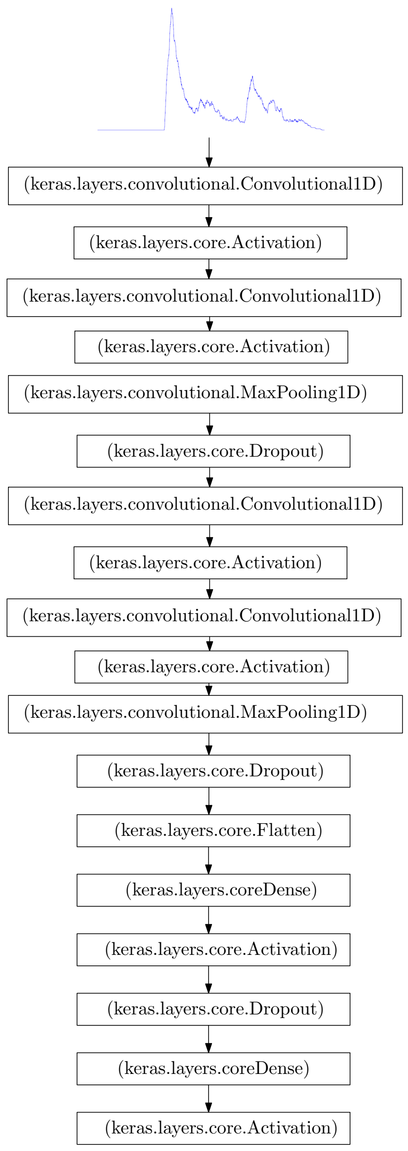
(keras.layers.convolutional.Convolutional1D)
(keras.layers.core.Activation)
(keras.layers.convolutional.Convolutional1D)
(keras.layers.core.Activation)
(keras.layers.convolutional.MaxPooling1D)
(keras.layers.core.Dropout)
(keras.layers.convolutional.Convolutional1D)
(keras.layers.core.Activation)
(keras.layers.convolutional.Convolutional1D)
(keras.layers.core.Activation)
(keras.layers.convolutional.MaxPooling1D)
(keras.layers.core.Dropout)
(keras.layers.core.Flatten)
(keras.layers.coreDense)
(keras.layers.core.Activation)
(keras.layers.core.Dropout)
(keras.layers.coreDense)
(keras.layers.core.Activation)

Figure 12: Architecture of two round convolutional neural network used in the electromobility classification problem

4.4 Energy Consumption Forecasting

Total Energy Consumption Forecasting. As mentioned in Chapter 2 Subsection 2.2 - Time Series Forecasting, when both linear and nonlinear patterns occur, it is beneficial to use a hybrid model that can capture both linearity and nonlinearity at the same time. In the first part of our energy consumption forecasting problem, we follow the following procedure[1] (depicted in Figure 13) to construct the hybrid model.

(i) Check the autocorrelation function (ACF) and the partial autocorrelation function (PCF) to analysis the order of linear model ARIMA.

(ii) The ARIMA model outputs the estimated linear component $\tilde{L}$ and the residuals e_t.

(iii) Train recurrent neural networks (LSTMs and bi-directional LSTMs) to model the nonlinear component $\tilde{N}_t \equiv e_t$.

(iv) Train a feedforward neural network to model the final hybrid result:

$$\tilde{x}_t = f(\tilde{L}_t, \tilde{N}_t). \tag{4.1}$$

Note the difference between Equation 4.1 in this subsection and Equation 3.13. Here we use a function $f()$ to symbolize the feedforward neural network at the final step of our procedure. The reason we use a FNN to model the final hybrid result instead of a linear combination as in Zhang's original paper[Zha03] is that the relationship between the linear component $\tilde{L}_t$ and the nonlinear component $\tilde{N}_t$ can never be described with a simplistic addition and this underestimation of the model complexity will result in a performance underachievement[TC05]. Therefore, it is reasonable to take advantage of FNNs' universal approximation feature in the final step of the procedure instead of making some oversimplified assumptions. Also, another significant difference between our hybrid model and previous hybrid methodologies is the usage of recurrent neural networks to model the residuals of ARIMA, as RNNs demonstrate superiority when it comes to sequential data modeling.

[1]The procedure here is very similar to the methodology originally proposed by Zhang et.al.[Zha03] and the one by Khashei et.al.[KB11].

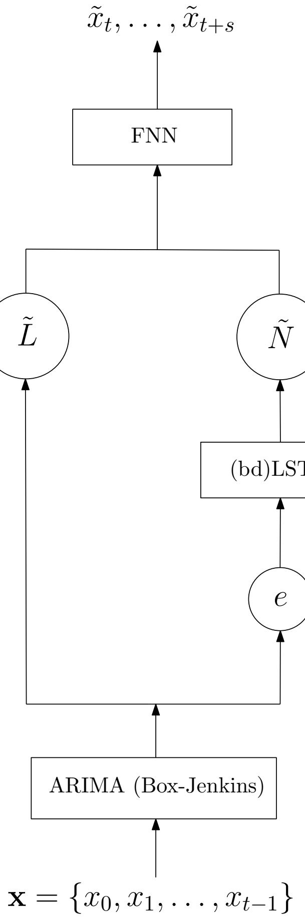
$\tilde{x}_t, \ldots, \tilde{x}_{t+s}$
FNN
$\tilde{L}$
$\tilde{N}$
(bd)LSTM
e
ARIMA (Box-Jenkins)
$\mathbf{x} = \{x_0, x_1, \ldots, x_{t-1}\}$

Figure 13: Architecture of the hybrid ARIMA+(bd)LSTM forecasting model

In addition to the hybrid model above, we also test the predictive performance of ARIMA, MLP, simple LSTM and bi-directional LSTM (bdLSTM) on the electromobility forecasting datasets and use their results as benchmark to study the relative strength of the hybrid ARIMA+(bd)LSTM model.

Individual Energy Consumption Forecasting. Originally, we had the intention of using the aforementioned hybrid ARIMA+(bd)LSTM forecasting model to predict the individual energy consumption as well. However, as it turns out in the experiment, the complex hybrid model does not yield the best results for the individual vehicle consumption problem. In fact, the methodology with the best performance in this setting is the *daily random walk* model and will be further discussed in Subsection 5.2.

The *daily random walk* model is simply a direct application of the *random walk* (Equation 3.2). In the electromobility dataset's case, we predict the individual energy consumption to be the same at the same timepoint of the previous day. So the daily random walk model for the electromobility dataset can be described with the following equation:

$$\tilde{x}_t = x_{t-1440} + \epsilon_t \tag{4.2}$$

4.5 Charging Prioritization

In comparison to the two previous problems, the charging prioritization is in fact a direct use case of the individual energy consumption forecasting model in Subsection 4.4. The detailed algorithm description of the charging prioritization scheme can be found in Algorithm 5. One assumption for the algorithm is that the concerned vehicles are waiting in line for charging all at the same timepoint t. The forecasting function for individual energy consumption is denoted by $Idv\_EC()$, which is based on the individual energy consumption forecasting model in Subsection 4.4. This function takes the vehicle id i and horizon s as input for a vehicle and outputs the estimated individual energy consumption at timepoint $t + s$. If $Idv\_EC()$ returns 1, it means the vehicle is charging. Otherwise it means the vehicle is not at a charging station.

Algorithm 3: Required Charging Time Estimation (RCTE)

```
input : vehicle id $i$ and maximum horizon $s_{max}$
output: estimated charging time that is required for the vehicle $c_i$
1 for $t \leftarrow 1$ to $s_{max}$ do
2     if $Idv\_EC(i, t) == 1$ $AND$ $Idv\_EC(i, t + 1) == 0$ then return t
3 return NULL
```

Algorithm 4: Charging Prioritization Comparator (CPC)

```
input : vehicle ids $i$ and $j$, maximum horizon $s_{max}$
output: vehicle charging priority order
1 if $RCTE(i, s_{max}) < RCTE(j, s_{max})$ then return 1
2 else return 0
```

Algorithm 5: Charging Prioritization Scheme ($n > 2$)

```
input : vehicle ids $\{1, \ldots, n\}$, maximum horizon $s_{max}$
output: vehicle charging priority order $(p_1, \ldots, p_n)$
1 return QSort($\{c_1, \ldots, c_n\}$, n, CPC)
```

The intuition behind the scheme is pretty straightforward. We compare the priority of two vehicles by estimating their respective needed charging time. If one vehicle needs less charging time, then we let it charge first to reduce the number of vehicles waiting in line as fast as possible. And we use this idea to build a priority comparator function and use it as the base comparator for QSort function which outputs the final vehicle charging priority order $(p_1, \ldots, p_n)$ for all n vehicles waiting in line. A parameter s_{max} is defined to confine the maximum horizon the forecasting model is looking into.

5 Experimental Evaluation

The experiments in the thesis have been conducted on a Linux system with Intel i7 3630QM processor, GeForce GTX 670M graphic card and 32GB DDR3 memory. The methodologies are implemented with Keras[Cho15] library in Python. The datasets for the classification as well as the time series problem are originated from the electromobility simulation model. The details on the datasets can be found in Subsection 4.2.

5.1 Charging and Driving Behavior Classification

Driving Behavior. The development of accuracy rate and categorical entropy loss for driving pattern classification is illustrated in Figure 14. Meanwhile, the confusion matrix for the final convolutional neural network model can be found in Figure 15.

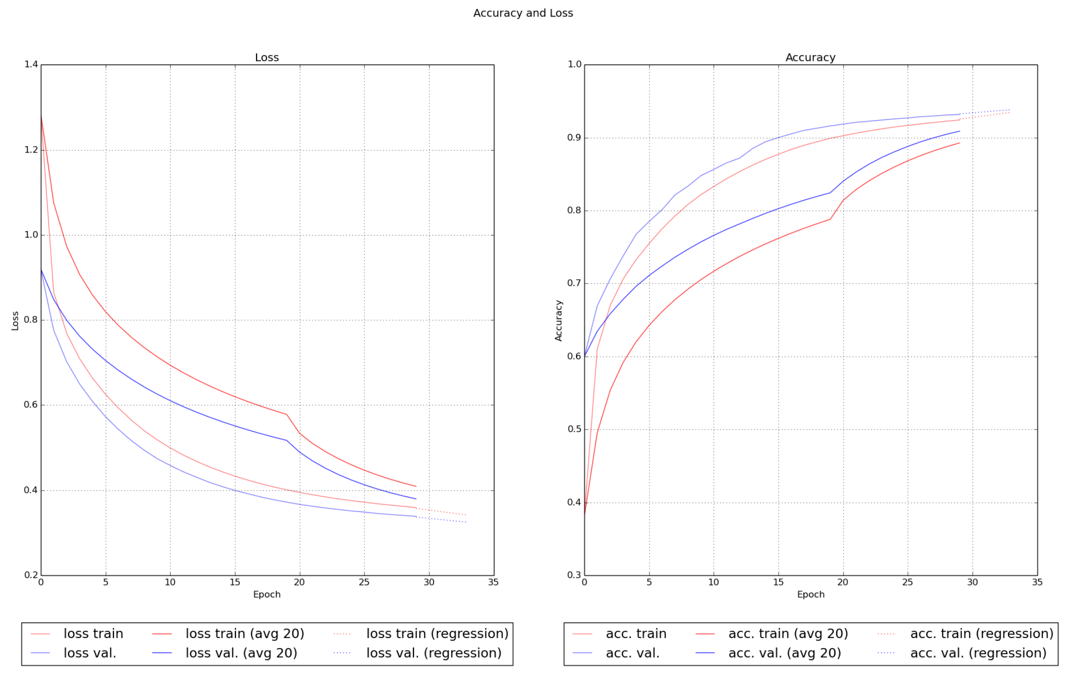

Figure 14: Accuracy and loss development over time for driving behaviors

The classifying performance (accuracy and loss) of the benchmark models: dynamic time warping - k-nearest neighbor (DTW-kNN) and standard multi-layer perceptron (MLP or FNN) as well as three convolutional neural networks (CNNs) with different numbers of CONV-ReLU-CONV-ReLU-POOL rounds (1, 2 and 3) is presented in Table 3. From the table we can see that a deep convolutional neural network with 3 rounds of CONV-ReLU-CONV-ReLU-POOL layers achieves the best classifying performance.

	MLP	DTW-kNN	CNN-1	CNN-2	CNN-3
Accuracy	84.2%	86.5%	89.7%	91.1%	**91.6%**
Loss	0.487	0.451	0.401	0.379	**0.372**

Table 3: Performance comparison between the models for driving behavior classification (best results in **bold**)

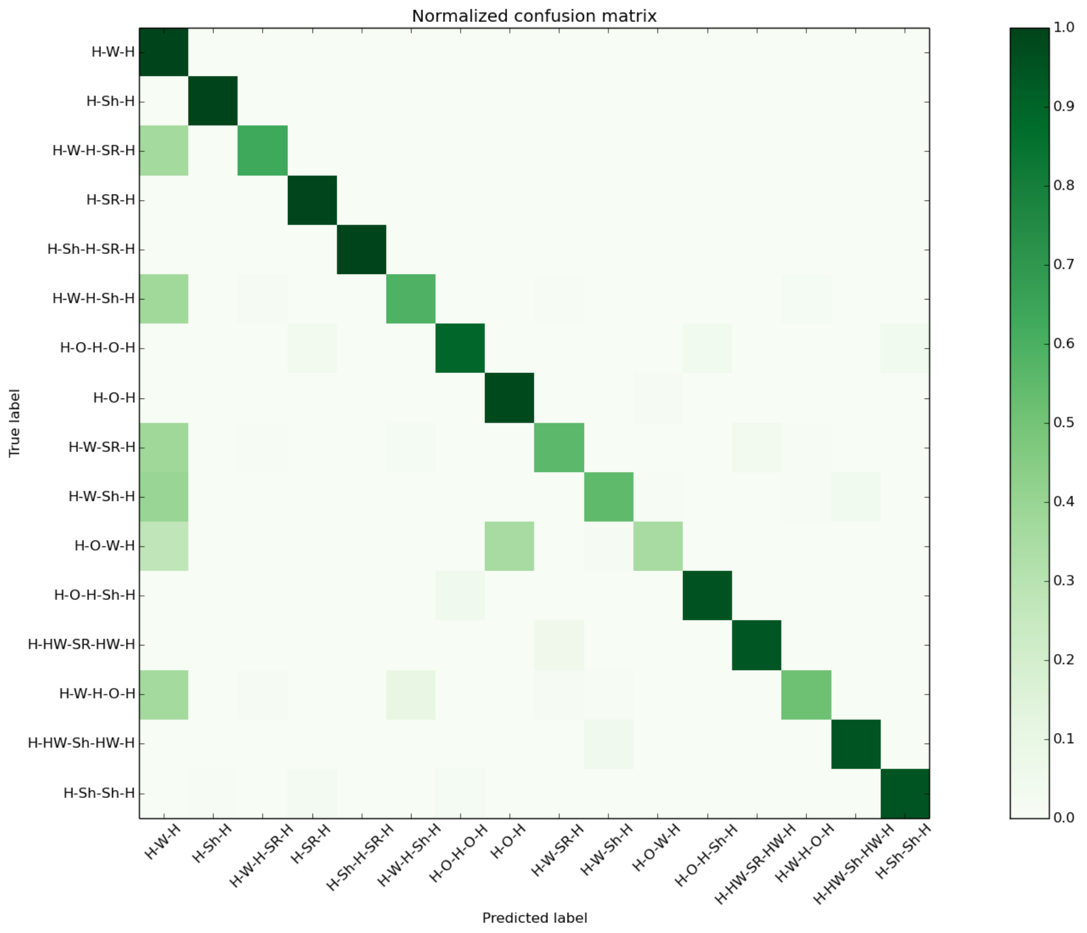

Figure 15: Normalized confusion matrix for driving behaviors

From the confusion matrix shown in Figure 15, we can gain some interesting insights into the relationship amongst different driving behaviors:

(i) Firstly, H-W-H, H-Sh-H, H-SR-H, H-Sh-H-SR-H, H-O-H, H-O-H-Sh-H, H-HW-SR-HW-H, H-HW-Sh-HW-H as well as H-Sh-Sh-H can be fairly easily separated from other behavioral patterns.

(ii) H-W-H-SR-H, H-W-H-Sh-H, H-W-SR-H, H-W-Sh-H, H-W-H-O-H are not easy to be classified and are often mistaken as H-W-H. One explanation is that vehicles in these patterns have similar driving behaviors with ones in H-W-H. This means that when a driver leaves home in the morning and decides to go to full time job, then no matter what other activities he will choose later on, the driving pattern will be very similar.

(iii) Pairs of H-O-H and H-O-W-H, H-W-H-O-H and H-W-H-Sh-H are also very difficult to discern from each other. This means that when difference between two patterns is small, the difference between the driving behaviors in the two patterns will also be negligible.

Charging Behavior. The development of accuracy rate and categorical entropy loss for charging pattern classification is illustrated in Figure 16. Interestingly, we can see from Figure 16 that even after several epochs of training, the accuracy rate still stays around **33%**. The only plausible interpretation is that the daily energy consumption for a vehicle does not change when different charging patterns are selected.

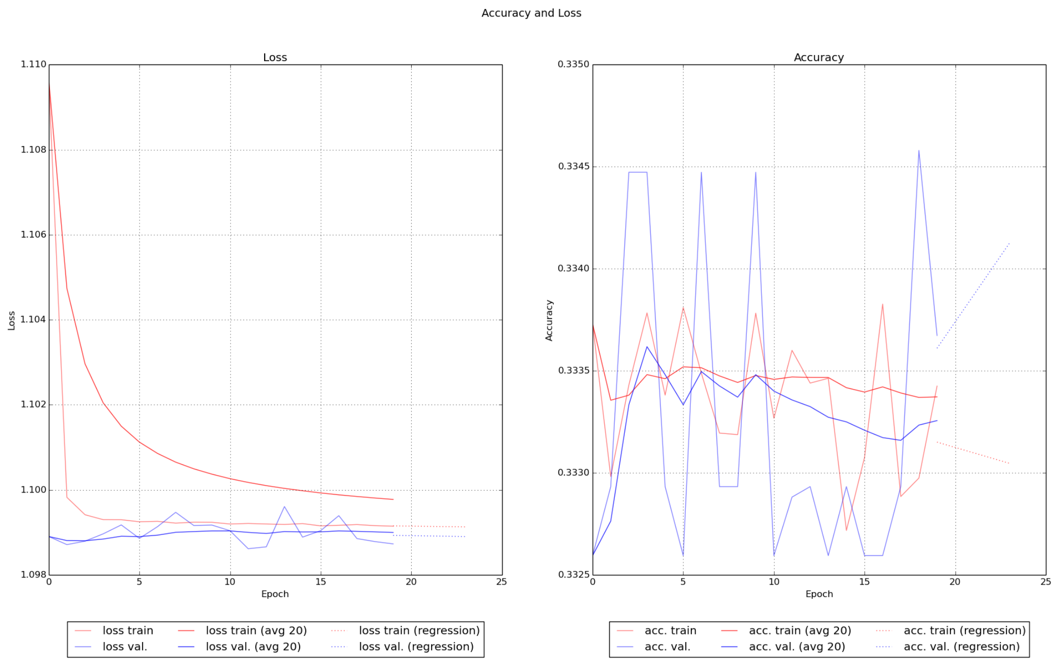

Figure 16: Accuracy and loss development over time for charging behaviors

5.2 Energy Consumption Forecasting

Total Energy Consumption. Firstly, before we conduct the hybrid forecasting procedure, let us take a look at the original time series data for alwaysCharging strategy in Figure 17.

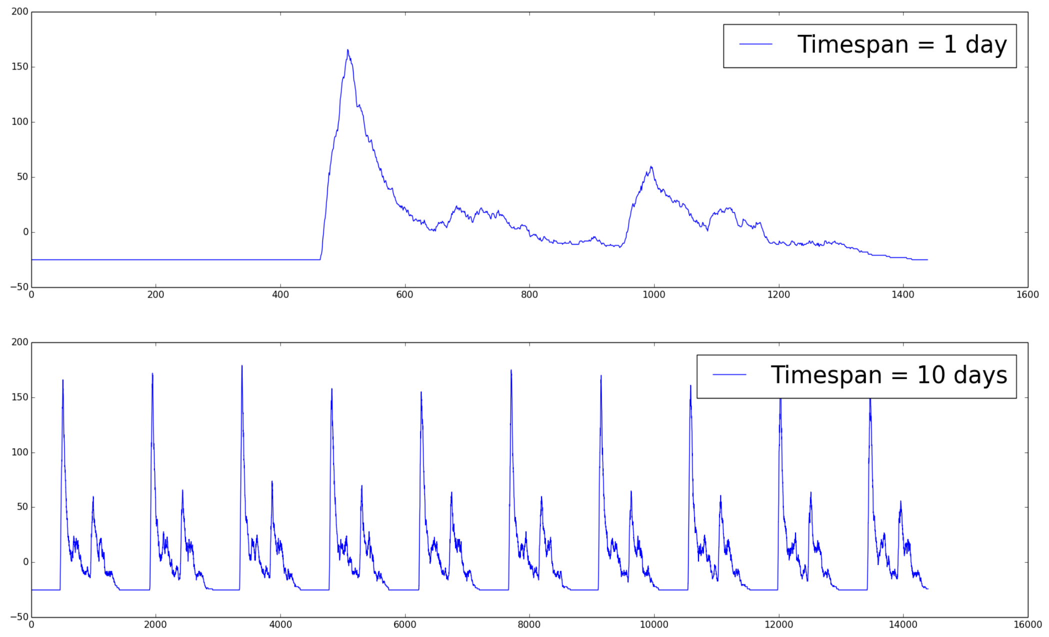

Figure 17: Original total energy consumption data - alwaysCharging

As we can clearly see from Figure 17, the temporal data shows a strong daily seasonality with minor oscillations. Also, the daily energy consumption has two peaks, one at morning rush hour when most of people begin going to work and another at evening rush hour when people leave from work to return home. After inspecting the form of original data, we check the autocorrelation function (ACF) and the partial autocorrelation function (PCF) to analysis the order of the linear model ARIMA. Figure 18 shows the ACF and PCF results for the time series data. Once again, we only present the figure for alwaysCharging strategy due to page length constraint.

From the autocorrelation ACF of the time series data, we recognize strong seasonality that is consistent with what we first observed in Figure 17. Furthermore, strong autocorrelation between the data and the lags of itself is shown in the ACF. In PCF, a cutoff phenomena is observed indicating the order of p in the ARIMA model.

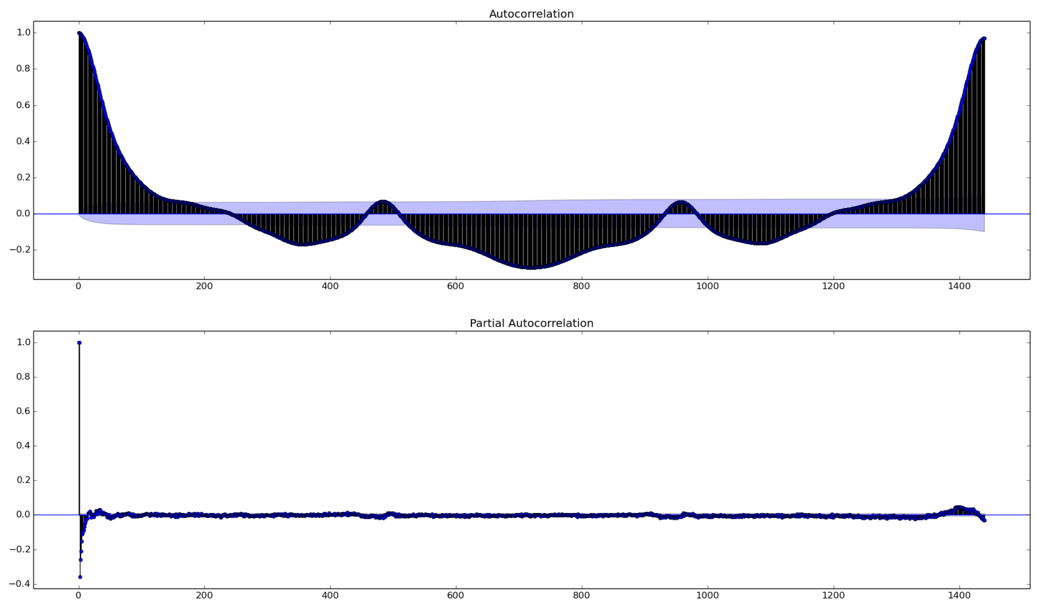

Figure 18: ACF (above) and PCF (below) of the total energy consumption data - alwaysCharging

Adhering to the hybrid forecasting procedure stated in Subsection 4.4, we conduct diagnostic analysis on the linear model ARIMA and use the best linear model to compute the linear component of the estimation. The residuals are fed into long short-term memory networks and bi-directional LSTMs which are implemented in Keras[Cho15]. In the implementation stage, we have found out that the nonlinear component can be sufficiently modeled with a LSTM network. Hybrid ARIMA+LSTM (HLSTM) model shows comparable predictive performance in comparison to ARIMA+bdLSTM (HbdLSTM) model (shown in Table 4). For evaluation, only mean square errors (MSEs) for alwaysCharging strategy are shown in Table 4 for simplicity and page length constraint reasons.

Note that the forecasting experimentation on the total energy consumption dataset concerns itself with a forecasting problem of horizon $s = 1$. By confining the horizon, the running time for the algorithm implementation can also be limited. Also, because strong seasonality is observed in the dataset, when the requirement for horizon $s > 1440$, which means we want to see at least one day ahead into future, we can simply take $x_{t+s-1440}$ as the estimation.

	ARIMA	MLP	LSTM	bdLSTM	HLSTM	HbdLSTM
MSE	50.48	37.62	22.31	21.09	18.77	**18.26**

Table 4: Performance comparison between the models for total energy consumption forecasting (best results in **bold**)

The following three figures 19, 20, 21 show the forecasting results of hybrid ARIMA+LSTM model for the three existing charging strategies.

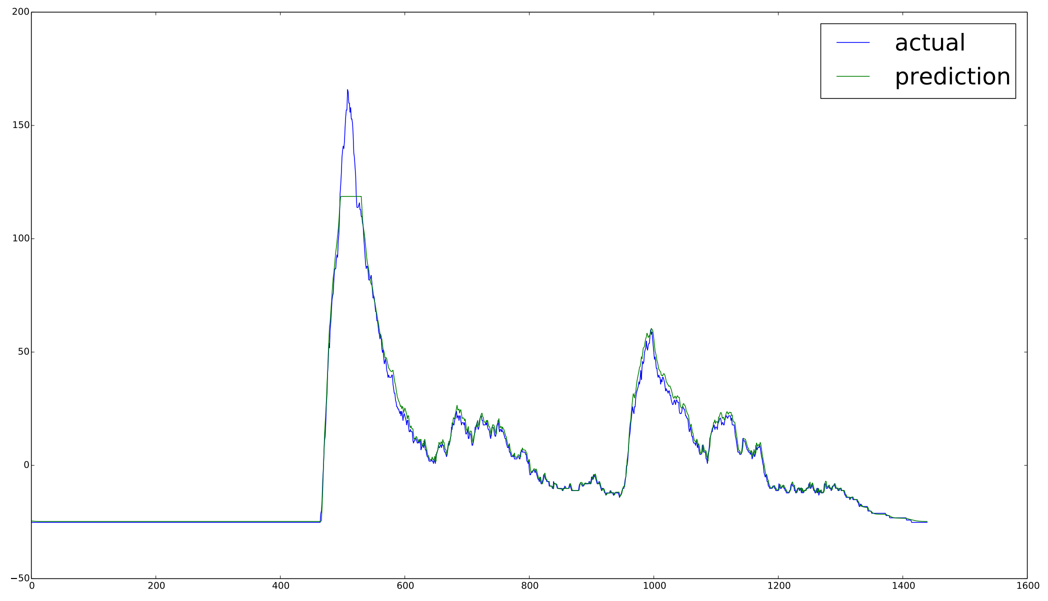

Figure 19: Forecasting result of ARIMA+LSTM for charging strategy - alwaysCharging

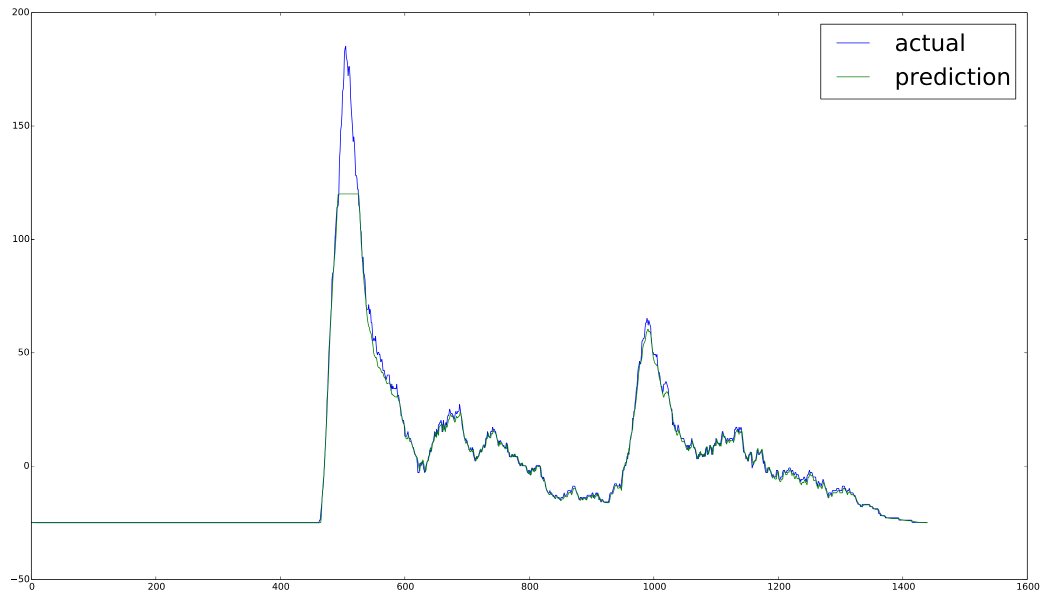

Figure 20: Forecasting result of ARIMA+LSTM for charging strategy - OnlyatWork

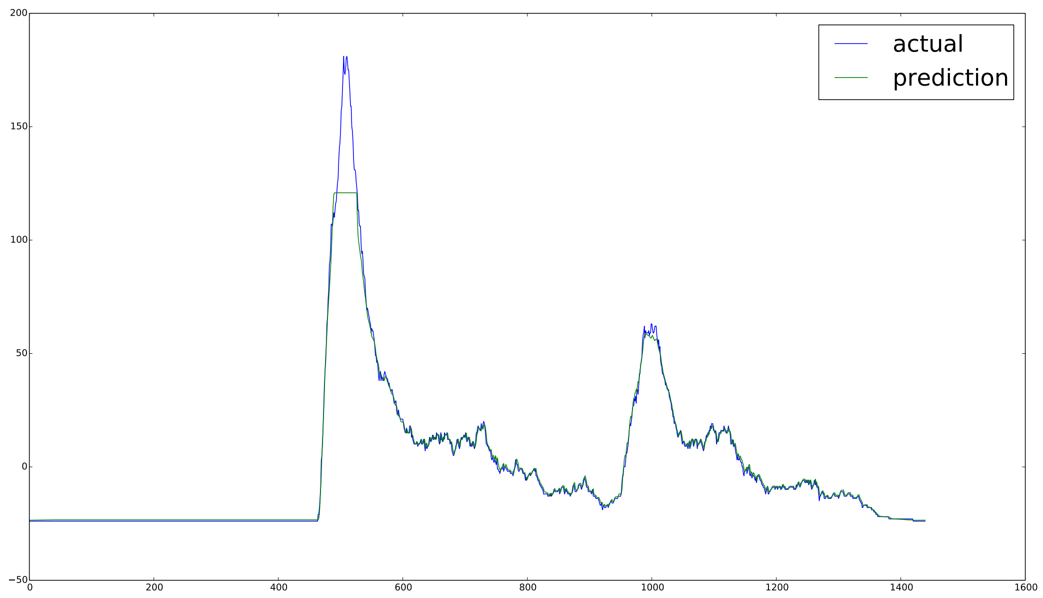

Figure 21: Forecasting result of ARIMA+LSTM for charging strategy - OnlyatHome

Individual Energy Consumption. The aforementioned hybrid model can also be applied for the individual energy consumption. But if we look at Figure 22 depicting the daily individual energy consumption of one vehicle, we will see that this would not be recommendable.

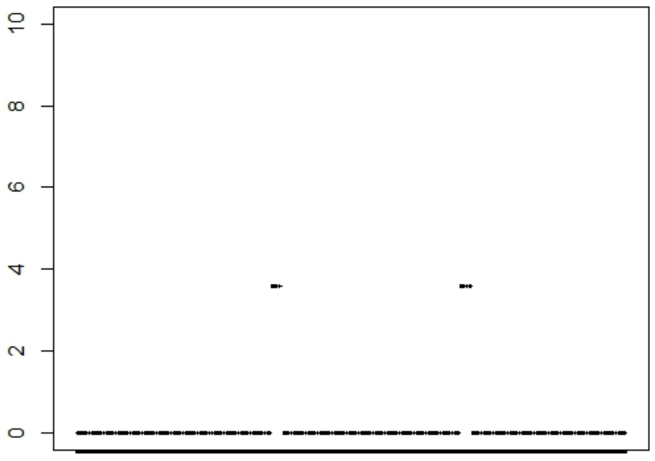

Figure 22: Daily individual energy consumption of one example vehicle

Just as shown above, a individual person normally will charge his vehicle twice (as in Figure 22) or three times a day. The form of individual energy consumption will be noncontinuous and the energy consumption values of most timepoints will be zero. We have explored the possibility of a hybrid model on individual energy consumption time series data and the resulting performance is unsurprisingly disappointing. The reason for the poor performance of the hybrid model is that almost all of predicted individual consumption values are estimated as 0, because a dominant percentage of the consumption values in the time series are 0 and the nonlinear component is poorly modeled as the LSTM/bdLSTM neurons are not “fired up”.

Therefore, we propose a more straightforward and easier solution for this problem, namely *daily random walk*, which means the best consumption estimation $\tilde{x}_t$ for timepoint t is x_{t-1440}. Or in other words, we take the value at the same timepoint of the last day as the estimate for any future observations. Interestingly, this results in a mean square error (MSE) of **0.029**. The low MSE compared to the MSE in the total consumption forecasting problem comes also from the fact that most of the data entries are zero and it is not difficult to achieve a low MSE under this kind of problem setting.

6 Conclusion

6.1 Summary

Time series classification and forecasting have always been considered challenging and extremely important tasks in research and industry communities. In this thesis, we utilize the electromobility model, which simulates the collective and individual activities of electric vehicles on the virtual island of la Réunion and design:

(i) a classification framework based on deep *convolutional neural network* (CNN) that is capable of classifying driving behaviors of various vehicles whose owners behave in different manners. After comparison of the implemented CNN model with other benchmark approaches, we confirm that deep CNN can promise the best classifying performance. Also, during the experimentation, we also have gained deeper insights into relationships between different driving patterns. However, as far as the charging pattern is concerned, we have come to the conclusion that the energy consumption patterns of vehicles show no strong correlation with their individual charging behaviors.

(ii) a hybrid forecasting model based on the integration of popular linear model *autoregressive integrated moving average* (ARIMA) and nonlinear model *long short-term memory* (LSTM). The hybrid model has demonstrated superior predictive power against other simple linear or nonlinear models for the total energy consumption forecasting problem in the electromobility dataset. On the other hand, a simple *daily random walk* model has shown to be more suitable for the individual energy consumption forecasting problem. This also proves that no model is omnipotent in field of time series forecasting and special care should be taken so that a tailored and appropriate model can be chosen for any specific problem.

(iii) a charging prioritization algorithm that allows the charging stations on the virtual island to determine the charging order when multiple vehicles come into charging station and energy resources are limited. This scheme is a special use case for the individual energy consumption forecasting approach (discussed in Subsections 4.4 and 5.2) and can be applied in real-life energy allocation and distribution systems.

Moreover, the classification, forecasting and charging prioritization models in this thesis can be directly adopted for demand side management (DSM) in a complex power grid. The results from the system can be utilized at the early stage of smart power grid management which can monitor the trends of energy consumption demand and act accordingly[Évo14]. In the long run, the models implemented in this thesis can be utilized for energy planning scenarios where massive introduction of renewable energy source is needed and a large amount of electric vehicles are present.

6.2 Future Work

Some possible suggestions for improvement and extension of the thesis project include:

(i) taking geographic information into consideration for better classification results. In our convolutional neural network approach, only the temporal data of energy consumption is exploited. But as the energy consumption of the vehicles is also directly related to the relative distances of all the visited locations in a day, it is understandable to think that some improvement can be expected if we feed all the geographic information as additional input into the classifying model.

(ii) acquiring real-life electric vehicle data and enhancing the generalizing capacities of the classification model.

(iii) constructing comprehensive forecasting architecture that can handle both linearity and nonlinearity, decide autonomously which model or hybrid model to use for specific problem settings.

(iv) coming up with more use cases for the classification and forecasting models to facilitate efficient energy allocation and precise long and short term energy planning.

References

[AB07] Nicola Armaroli and Vincenzo Balzani. “The Future of Energy Supply: Challenges and Opportunities”. In: *Angewandte Chemie International Edition* 46.1-2 (2007), pp. 52–66. URL: http://dx.doi.org/10.1002/anie.200602373.

[Bar63] G. A. Barnard. “New methods of quality control”. In: *Journal of the Royal Statistical Society* 126 (1963), pp. 255–258.

[Bar71] D. J. Bartholomew. In: *Operational Research Quarterly (1970-1977)* 22.2 (1971), pp. 199–201. URL: http://www.jstor.org/stable/3008255.

[Ben09] Yoshua Bengio. “Learning deep architectures for AI”. In: *Foundations and Trends in Machine Learning* 2.1 (2009). Also published as a book. Now Publishers, 2009., pp. 1–127.

[BFC95] Samy Bengio, Francoise Fessant, and Daniel Collobert. “A Connectionist System for Medium-Term Horizon Time Series Prediction”. In: *IN PROC. INTL. WORKSHOP APPLICATION NEURAL NETWORKS TO TELECOMS*. 1995, pp. 308–315.

[BG01] J. M. Bates and C. W. J. Granger. “Essays in Econometrics”. In: ed. by Eric Ghysels, Norman R. Swanson, and Mark W. Watson. New York, NY, USA: Cambridge University Press, 2001. Chap. The Combination of Forecasts, pp. 391–410. URL: http://dl.acm.org/citation.cfm?id=766886.766907.

[BJ90] George Edward Pelham Box and Gwilym Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[Bot12] Léon Bottou. “Stochastic Gradient Descent Tricks.” In: *Neural Networks: Tricks of the Trade (2nd ed.)* Ed. by Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller. Vol. 7700. Lecture Notes in Computer Science. Springer, 2012, pp. 421–436. URL: http://dblp.uni-trier.de/db/series/lncs/lncs7700.html#Bottou12.

[BSF] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. *Learning Long-Term Dependencies with Gradient Descent is Difficult.*

[Bun85] Derek W. Bunn. “Statistical efficiency in the linear combination of forecasts”. In: *International Journal of Forecasting* 1.2 (1985), pp. 151–163. URL: http://EconPapers.repec.org/RePEc:eee:intfor:v:1:y:1985:i:2:p:151-163.

[Cho15] François Chollet. *Keras*. https://github.com/fchollet/keras. 2015.

[Chu+14] Junyoung Chung et al. “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”. In: *CoRR* abs/1412.3555 (2014). URL: http://arxiv.org/abs/1412.3555.

[Cle89] Robert T. Clemen. "Combining forecasts: A review and annotated biography (with discussion)". In: *International Journal of Forecasting* 5 (1989), pp. 559–583.

[CMS12] Dan Ciresan, Ueli Meier, and Jürgen Schmidhuber. "Multi-column deep neural networks for image classification". In: *IN PROCEEDINGS OF THE 25TH IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR 2012*. 2012, pp. 3642–3649.

[DGT94] Melinda Deutsch, Clive W.J. Granger, and Timo Teräsvirta. "The combination of forecasts using changing weights". In: *International Journal of Forecasting* 10.1 (1994), pp. 47–57. URL: http://www.sciencedirect.com/science/article/pii/0169207094900493.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *J. Mach. Learn. Res.* 12 (July 2011), pp. 2121–2159. URL: http://dl.acm.org/citation.cfm?id=1953048.2021068.

[Eng82] Robert Engle. "Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation". In: *Econometrica* 50.4 (1982), pp. 987–1007. URL: http://EconPapers.repec.org/RePEc:ecm:emetrp:v:50:y:1982:i:4:p:987-1007.

[Évo14] José Évora-Gomez. "A methodological research on software engineering applied to the design of Smart Grids using a Complex System approach". dissertation. Universidad de Las Palmas de Gran Canaria, 2014. URL: https://www.researchgate.net/publication/273008206_A_methodological_research_on_software_engineering_applied_to_the_design_of_Smart_Grids_using_a_Complex_System_approach.

[FDH01] R. J. Frank, N. Davey, and S. P. Hunt. "Time Series Prediction and Neural Networks". In: *J. Intell. Robotics Syst.* 31.1-3 (May 2001), pp. 91–103. URL: http://dx.doi.org/10.1023/A:1012074215150.

[Fio98] Antonio Fiordaliso. "A nonlinear forecasts combination method based on Takagi-Sugeno fuzzy systems". In: *International Journal of Forecasting* 14 (1998), pp. 367–379.

[FN93] Ken-Ichi Funahashi and Yuichi Nakamura. "Approximation of dynamical systems by continuous time recurrent neural networks". In: *Neural Netw.* 6.6 (1993), pp. 801–806. URL: http://portal.acm.org/citation.cfm?id=173530.

[FP02] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[GA78] C. W. J. Granger and A. P. Andersen. *An Introduction to Bilinear Time Series Models*. Vandenhoeck und Ruprecht, 1978.

[Gre+15] Klaus Greff et al. "LSTM: A Search Space Odyssey". In: *CoRR* abs/1503.04069 (2015). URL: http://arxiv.org/abs/1503.04069.

[GS00] F. A. Gers and J. Schmidhuber. "Recurrent nets that time and count". In: *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*. Vol. 3. 2000, 189–194 vol.3.

[GS05] Alex Graves and Jürgen Schmidhuber. "Framewise phoneme classification with bidirectional LSTM and other neural network architectures". In: *Neural Networks* 18.5-6 (2005), pp. 602–610. URL: http://dx.doi.org/10.1016/j.neunet.2005.06.042.

[GSC99] F. A. Gers, J. Schmidhuber, and F. Cummins. "Learning to forget: continual prediction with LSTM". In: *Artificial Neural Networks, 1999. ICANN 99. Ninth International Conference on (Conf. Publ. No. 470)*. Vol. 2. 1999, 850–855 vol.2.

[Ham98] Barbara Hammer. "On the Approximation Capability of Recurrent Neural Networks". In: *In International Symposium on Neural Computation*. 1998, pp. 12–4.

[HK06] Rob J Hyndman and Anne B Koehler. "Another look at measures of forecast accuracy". In: *International Journal of Forecasting* (2006), pp. 679–688.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. "Multilayer Feedforward Networks Are Universal Approximators". In: *Neural Netw.* 2.5 (July 1989), pp. 359–366. URL: http://dx.doi.org/10.1016/0893-6080(89)90020-8.

[HW68] David H. Hubel and Torsten N. Wiesel. "Receptive Fields and Functional Architecture of Monkey Striate Cortex". In: *Journal of Physiology (London)* 195 (1968), pp. 215–243.

[Jae02] Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach*. Tech. rep. 159. St. Augustin-Germany: Fraunhofer Institute for Autonomous Intelligent Systems (AIS), Oct. 2002.

[Jar+09] Kevin Jarrett et al. "What is the best multi-stage architecture for object recognition?" In: *ICCV*. IEEE, 2009, pp. 2146–2153. URL: http://dblp.uni-trier.de/db/conf/iccv/iccv2009.html#JarrettKRL09.

[JH04] Herbert Jaeger and Harald Haas. "Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication". In: *Science* 304.5667 (2004), pp. 78–80. eprint: http:

//science.sciencemag.org/content/304/5667/78.full.pdf. URL: http://science.sciencemag.org/content/304/5667/78.

[JJ62] Orbach JJ. "PRinciples of neurodynamics. perceptrons and the theory of brain mechanisms." In: *Archives of General Psychiatry* 7.3 (1962), pp. 218–219. eprint: /data/Journals/PSYCH/12091/archpsyc_7_3_010.pdf. URL: +%20http://dx.doi.org/10.1001/archpsyc.1962.01720030064010.

[KB11] Mehdi Khashei and Mehdi Bijari. "A Novel Hybridization of Artificial Neural Networks and ARIMA Models for Time Series Forecasting". In: *Appl. Soft Comput.* 11.2 (Mar. 2011), pp. 2664–2675. URL: http://dx.doi.org/10.1016/j.asoc.2010.10.015.

[KBR09] Mehdi Khashei, Mehdi Bijari, and Gholam Ali Raissi Ardali. "Improvement of Auto-Regressive Integrated Moving Average Models Using Fuzzy Logic and Artificial Neural Networks (ANNs)". In: *Neurocomput.* 72.4-6 (Jan. 2009), pp. 956–967. URL: http://dx.doi.org/10.1016/j.neucom.2008.04.017.

[Kri09] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[KT01] Lutz Kilian and Mark Taylor. *Why is it so difficult to beat the random walk forecast of exchange rates?* Working Paper Series 0088. European Central Bank, 2001. URL: http://EconPapers.repec.org/RePEc:ecb:ecbwps:20010088.

[LeC+98] Yann LeCun et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE*. Vol. 86. 11. 1998, pp. 2278–2324. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665.

[Lip15] Zachary Chase Lipton. "A Critical Review of Recurrent Neural Networks for Sequence Learning". In: *CoRR* abs/1506.00019 (2015). URL: http://arxiv.org/abs/1506.00019.

[LKF10] Y. LeCun, K. Kavukvuoglu, and C. Farabet. "Convolutional Networks and Applications in Vision". In: *Circuits and Systems, International Symposium on*. 2010, pp. 253–256.

[Mck84] Ed Mckenzie. "General Exponential Smoothing and the Equivalent ARMA Process". In: *Journal of Forecasting* 3 (1984), pp. 333–344.

[MP] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathemat-*

ical biophysics 5.4 (), pp. 115–133. URL: http://dx.doi.org/10.1007/BF02478259.

[MP69] Marvin Minsky and Seymour Papert. *Perceptrons : an introduction to computational geometry*. Cambridge (Mass.), London, 1969. URL: http://opac.inria.fr/record=b1080139.

[NN09] Yuichi Nakamura and Masahiro Nakagawa. “Artificial Neural Networks – ICANN 2009: 19th International Conference, Limassol, Cyprus, September 14-17, 2009, Proceedings, Part II”. In: ed. by Cesare Alippi et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. Chap. Approximation Capability of Continuous Time Recurrent Neural Networks for Non-autonomous Dynamical Systems, pp. 593–602. URL: http://dx.doi.org/10.1007/978-3-642-04277-5_60.

[ON15] Keiron O’Shea and Ryan Nash. “An Introduction to Convolutional Neural Networks”. In: *CoRR* abs/1511.08458 (2015). URL: http://arxiv.org/abs/1511.08458.

[PMB12] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “Understanding the exploding gradient problem”. In: *CoRR* abs/1211.5063 (2012). URL: http://arxiv.org/abs/1211.5063.

[Rak+12] Thanawin Rakthanmanon et al. “Searching and Mining Trillions of Time Series Subsequences Under Dynamic Time Warping”. In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’12. Beijing, China: ACM, 2012, pp. 262–270. URL: http://doi.acm.org/10.1145/2339530.2339576.

[RHW86] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–536.

[RHW88] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Neurocomputing: Foundations of Research”. In: ed. by James A. Anderson and Edward Rosenfeld. Cambridge, MA, USA: MIT Press, 1988. Chap. Learning Representations by Back-propagating Errors, pp. 696–699. URL: http://dl.acm.org/citation.cfm?id=65669.104451.

[Rob83] Spyros Makridakis Robert L. Winkler. “The Combination of Forecasts”. In: *Journal of the Royal Statistical Society. Series A (General)* 146.2 (1983), pp. 150–157. URL: http://www.jstor.org/stable/2982011.

[Roj96] Raul Rojas. *Neural Networks : A Systematic Introduction*. Springer, 1996.

[Ros57] F. Rosenblatt. *The perceptron—a perceiving and recognizing automaton*. Report 85-460-1. Cornell Aeronautical Laboratory, 1957.

[Rus+14] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *CoRR* abs/1409.0575 (2014). URL: http://arxiv.org/abs/1409.0575.

[Rus+15] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252.

[SP97] M. Schuster and K.K. Paliwal. “Bidirectional Recurrent Neural Networks”. In: *Trans. Sig. Proc.* 45.11 (Nov. 1997), pp. 2673–2681. URL: http://dx.doi.org/10.1109/78.650093.

[Sri+14] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. URL: http://dl.acm.org/citation.cfm?id=2627435.2670313.

[SS91] Hava T. Siegelmann and Eduardo D. Sontag. “Turing Computability With Neural Nets”. In: *Applied Mathematics Letters* 4 (1991), pp. 77–80.

[ST09] Shahriar Shafiee and Erkan Topal. “When will fossil fuel reserves be diminished?” In: *Energy Policy* 37.1 (2009), pp. 181–189. URL: http://www.sciencedirect.com/science/article/pii/S03014215080041

[Sze+14] Christian Szegedy et al. “Going Deeper with Convolutions”. In: *CoRR* abs/1409.4842 (2014). URL: http://arxiv.org/abs/1409.4842.

[TC05] Tugba Taskaya-Temizel and Matthew C. Casey. “A comparative study of autoregressive neural network hybrids”. In: *Neural Networks* 18.5-6 (2005), pp. 781–789. URL: http://dx.doi.org/10.1016/j.neunet.2005.06.003.

[TD02] Nobuhiko Terui and Herman K. van Dijk. “Combined forecasts from linear and nonlinear time series models”. In: *International Journal of Forecasting* 18.3 (2002), pp. 421–438. URL: http://www.sciencedirect.com/science/article/pii/S0169207001001200.

[TF93] Zaiyong Tang and Paul A. Fishwick. “Feed-forward Neural Nets as Models for Time Series Forecasting”. In: *ORSA Journal of Computing* 5 (1993), pp. 374–385.

[TH12] Tijmen Tieleman and Geoffrey Hinton. *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude.* Tech. rep. 2012.

[Ton83] Howell Tong. “Threshold Models in Non-linear Time Series Analysis”. In: New York, NY: Springer New York, 1983. Chap. Threshold Models, pp. 59–121. URL: http://dx.doi.org/10.1007/978-1-4684-7888-4_3.

[Tor+15] S. Torres et al. “Agent-based modelling of electric vehicle driving and charging behavior”. In: *23rd Mediterranean Conference on Control and Automation (MED)* (2015), pp. 459–464.

[TT13] Siang Fui Tie and Chee Wei Tan. “A review of energy sources and energy management system in electric vehicles”. In: *Renewable and Sustainable Energy Reviews* 20 (2013), pp. 82–102. URL: http://www.sciencedirect.com/science/article/pii/S1364032112006910.

[Wai+90] Alexander Waibel et al. “Readings in Speech Recognition”. In: ed. by Alex Waibel and Kai-Fu Lee. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990. Chap. Phoneme Recognition Using Time-delay Neural Networks, pp. 393–404. URL: http://dl.acm.org/citation.cfm?id=108235.108263.

[Wal11] Kenneth Frank Wallis. “Combining forecasts : forty years later”. In: *Applied Financial Economics* 21.1-2 (2011), pp. 33–41.

[Wer90] Paul J. Werbos. “Backpropagation Through Time: What It Does and How to Do It”. In: *Proceedings of the IEEE* 78.10 (1990). Reprinted in [**Werbos1994TRo**], pp. 1550–1560.

[Win89] Robert L. Winkler. “Combining forecasts: A philosophical basis and some current issues”. In: *International Journal of Forecasting* 5.4 (1989), pp. 605–609. URL: http://www.sciencedirect.com/science/article/pii/0169207089900186.

[WZ89] Ronald J. Williams and David Zipser. “A Learning Algorithm for Continually Running Fully Recurrent Neural Networks”. In: *Neural Comput.* 1.2 (June 1989), pp. 270–280. URL: http://dx.doi.org/10.1162/neco.1989.1.2.270.

[ZF13] Matthew D. Zeiler and Rob Fergus. “Visualizing and Understanding Convolutional Networks”. In: *CoRR* abs/1311.2901 (2013). URL: http://arxiv.org/abs/1311.2901.

[Zha03] Guoqiang Peter Zhang. “Time series forecasting using a hybrid ARIMA and neural network model.” In: *Neurocomputing* 50 (June 2, 2003), pp. 159–175. URL: http://dblp.uni-trier.de/db/journals/ijon/ijon50.html#Zhang03.

[Zhe+14] Yi Zheng et al. “Web-Age Information Management: 15th International Conference, WAIM 2014, Macau, China, June 16-18, 2014. Proceedings”. In: ed. by Feifei Li et al. Cham: Springer International Publishing, 2014. Chap. Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks, pp. 298–310. URL: http://dx.doi.org/10.1007/978-3-319-08010-9_33.

[ZPH98] Guoqiang Zhang, B. Eddy Patuwo, and Michael Y. Hu. “Forecasting with artificial neural networks:: The state of the art”. In: *International Journal of Forecasting* 14.1 (Mar. 1, 1998), pp. 35–62.

URL: http://www.sciencedirect.com/science/article/B6V92-3T82TC6-5/1/7016407c7843327c8d0a1746d35bee40.