UNIVERSITY OF WEST ATTICA

DEPARTMENT OF INFORMATION COMPUTER ENGINEERS

THESIS

Machine Learning Model in Time Series

Panagiotis Tamvakidis
Klajdi Isufaj

Supervisor: Nikolaos Zacharis

Contents

Abstract 7

Περίληψη 8

Chapter 0: Introduction 9

0.1: Project aims and contributions 9

0.2: Thesis outline 9

Chapter 1: Background 11

1.1: Machine Learning 11

1.1.1: History 11

1.1.2: Methods of Machine Learning: 12

1.1.3: Who’s using Machine Learning? 13

1.1.3.1: Services using Machine Learning: 13

1.2: Deep Learning 15

1.2.1: History 16

1.2.2: Neural Network 17

1.3: Time Series 17

1.3.1: What makes time series dataset deferent from other datasets? 18

1.3.1.1: Time series dataset 18

1.3.1.2: Time series forecasting 18

1.4: Regression Model 19

1.4.1: Linear Regression 19

1.4.2: Lasso Regression 20

1.4.3: Ridge Regression 20

1.4.4: Logistic Regression 21

1.5: Python 21

1.5.1: Python Use Cases 22
1.5.1.1: Script Writing and Automation 22
1.5.1.2: Back-End Development 23
1.5.1.3: Data Science and Machine Learning 23
1.5.1.4: Internet of Things (IoT) 23
1.5.2: Interpreted languages 23
1.6: Pandas Software 24
1.6.1: Pandas features 24
1.6.2: Example with pandas 25
1.7: PyCharm 25
1.8: Jupyter 26
1.8.1: The notebook web application 26
1.8.2: Kernels 27
1.8.3: Notebook documents 27
1.9: NumPy 28
1.9.1: History 29
1.10: Particulate Matter (PM) 30
1.10.1: PM Effects 31
1.11: Keras 32
1.11.1: Keras Functional model 33
1.11.2: Keras Sequential model 33
1.11.3: LSTM Network 33
1.12: TensorFlow 36
1.13: Matplotlib 37
1.14: Scikit-learn 38
1.15: Anaconda 38

Chapter 2: Method 39
2.1: Python files 39
2.2: Import Libraries-Modules 39
2.3: Data Set.. 39
2.4: Input Data 42
2.5: Data Normalization 43
2.6: Splitting Data 43
2.7: Building the model.. 44
2.8: R squared (R^2) coefficient of determination regression score function 49
Chapter 3: Experimental results.. 51
3.1: CPU Mode 51
3.2: GPU Mode.. 52
3.3 CPU & GPU Comparison 53
3.4: R^2 (coefficient of determination) regression score function Results 54
Chapter 4: Source Code 56
4.1:ThLSTM.py File.. 56
4.2: HelpingFunctions File.. 58
Chapter 5: Conclusion.. 61
Chapter 6: Sources.. 63

Abstract

Machine learning is a significant part of human technology as it offers a wide range of applications daily, from internet search to talking robots. Time series forecasting is one of many applications that are provided and it has important role in many industries. Machine learning time series forecasting is the main topic of thesis and it will be analyzed a specific approach. Currently, weather prediction as long as air pollution predictions are used extensively with great success. The big advantage of time sequence predictions is that many issues in the near future can be prevented by giving a short-term period for preparation before the issue occurs. Based on that argument can one say that Particular Mater air pollution prediction is a very important achievement. The question we answer is how a machine learning model can respond to that needs and how accurate can it be. To achieve this statement, we worked with time series analysis practices as long as with machine learning libraries which are specialized for this type of predictions. The methodology is based on best practices according to documentation and the results of the model are analyzed in detail.

Περίληψη

Η μηχανική μάθηση αποτελεί σημαντικό μέρος της ανθρώπινης τεχνολογίας, καθώς προσφέρει ένα ευρύ φάσμα εφαρμογών καθημερινά, από αναζήτηση στο διαδίκτυο έως και ρομπότ που μιλάνε. Η πρόβλεψη χρονοσειρών είναι μία από τις πολλές εφαρμογές που παρέχονται και έχει σημαντικό ρόλο σε πολλές βιομηχανίες. Η πρόβλεψη χρονοσειρών μηχανικής μάθησης είναι το κύριο θέμα της διατριβής και θα αναλυθεί με συγκεκριμένη προσέγγιση. Επί του παρόντος, η πρόβλεψη για τον καιρό όσο οι προβλέψεις για την ατμοσφαιρική ρύπανση χρησιμοποιούνται εκτεταμένα με μεγάλη επιτυχία. Το μεγάλο πλεονέκτημα των προβλέψεων χρονικής αλληλουχίας είναι ότι πολλά ζητήματα στο εγγύς μέλλον μπορεί να αποφευχθούν δίνοντας ένα χρονικό διάστημα για την προετοιμασία πριν παρουσιαστεί το πρόβλημα. Βάσει αυτού του επιχειρήματος μπορούμε να πούμε ότι η πρόβλεψη για την ατμοσφαιρική ρύπανση των μικροσωματιδίων είναι ένα πολύ σημαντικό επίτευγμα. Το ερώτημα που απαντάμε είναι πώς ένα μοντέλο μηχανικής μάθησης μπορεί να ανταποκριθεί σε αυτές τις ανάγκες και πόσο ακριβής μπορεί να είναι. Για να επιτευχθεί, εργαστήκαμε με πρακτικές ανάλυσης χρονοσειρών, όσο με τις βιβλιοθήκες μηχανικής μάθησης που είναι εξειδικευμένες για αυτό το είδος προβλέψεων. Η μεθοδολογία βασίζεται στις βέλτιστες πρακτικές σύμφωνα με την βιβλιογραφία και αναλύονται λεπτομερώς τα αποτελέσματα του μοντέλου.

Chapter 0: Introduction

0.1: Project aims and contributions

This thesis contains three main aims. The first is to give an overview of machine learning and regression models predictions for time series forecast. Secondly, provides a practical implementation techniques of machine learning tools in order to analyze, prepare, visualize and predict the particular matter (PM2.5) average data of Shenyang city in China. In conclusion, the results of the predictive model as far as the accuracy and the loss are studied in detail according the true data.

0.2: Thesis outline

The latter part of this thesis has the following structure:

- Chapter 1: Background chapter contains a very brief general introduction to the basic concepts of machine learning and regression models as well as time series and python characteristics.
- Chapter 2: Method chapter contains the programming part of the model. It makes an analysis for all sections of the code and explains in detail the methodology as well as the purpose of the imported modules and libraries that been used.
- Chapter 3: Experimental results chapter provides a clear view about the results of thesis machine learning model. In addition, it mentions what we can conclude from these results and is achieved from them.
- Chapter 4: Conclusion chapter summarizes the thesis and suggests future research directions.

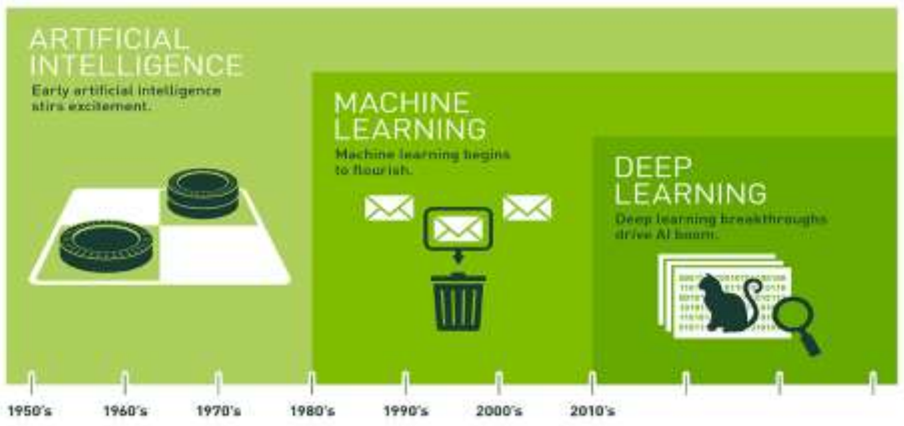
Chapter 1: Background

1.1: Machine Learning

One field of computer science is Machine Learning. Actually, Machine Learning is a sub-set of Artificial Intelligence where computer algorithms are used to autonomously learn from data and information. Machine Learning gives computers the ability to get programmed without the help of humans. Is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that machines should be able to learn and adapt through experience. Today, machine learning algorithms enable computers to communicate with humans, autonomously drive cars, write and publish sport match reports and find terrorist suspects.

1.1.1: History

The term “Machine Learning” coined in 1959 from Arthur Samuel while he was at IBM. Samuel was an American pioneer in the field of computer gaming and artificial intelligence. This specific term was born from pattern recognition and the theory that computers can learn without being programmed to perform specific tasks. Machine learning researchers were interested in artificial intelligence in order to understand if computers could learn from data – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions through building a model from sample inputs. Machine learning today is not like machine learning of the past, because of new computing technologies.


ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's

Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

1.1.2: Methods of Machine Learning:

The requirements of a machine leaning system in order to be efficient are multiple and the result depends on:

- Data: For output prediction input data are mandatory.
- Algorithms: Statistical algorithms are responsible to determine data types of machine learning.
- Automation: It is the ability to make systems automatically.
- Iteration: The complete process is iterative i.e. repetition of process.
- Scalability: The capacity of the machine can be increased or decreased in size and scale.
- Modeling: The models are created according to the demand by the process of modeling.

There are two categories that machine learning categories are teamed up:

- Supervised Learning: The majority of machine learning uses supervised learning. It is defined with pairs of input variables (X) and output variables (Y). A supervised learning algorithm analyzes the training data (input) and produces an inferred function which can be used for new mapping examples (output):

$$\mathbf{Y=f(X)}$$

- Unsupervised Learning: This type of machine learning is based on data that has not been labeled, categorized or classified. Unsupervised learning recognizes similarities among data and reacts based on the presence or absence of such commonalities on data pieces. This procedure variously called clustering analysis.

1.1.3: Who's using Machine Learning?

Machine Learning nowadays is widespread, and it is used in industries which have to work with great amount of data. Their main goal is to gain advantage over their competitors and to become more efficient in market through real time data analysis.

1.1.3.1: Services using Machine Learning:

Government: Government challenge is to provide citizens more efficient, effective, and transparent services with strict and often decreasing budgets. Analyzing sensor data, for example, identifies ways to increase efficiency and save money. Also Machine learning can help to identify anomalies or signatures to address proliferation, terrorism, money laundering, counterfeit devices, threats, fraud and other criminal activity.

Business and Marketing: According to a MIT survey of 168 large companies found that 76% of them are using machine learning technologies to assist their sales growth strategies. For example one common use of machine learning is in the websites. By entering a website the browser recommend you items which might based on previous purchases. These recommendation engines are algorithms which can accurately predict what you might buy. These engines are seen on Amazon, the New York Times and Netflix. Also Google use AI and Machine Learning for her search engine in order to return more relevant results. Also big companies use ML in their products to win the competition. To be more specific, a brief example is that Apple in the new iPhone's use Siri which is an intelligent personal assistant. This assistant uses voice queries and a natural language user interface to answer questions, make recommendations, and perform actions by delegating requests to a set of Internet services. In addition, now in the new iPhone both cameras use ML to help the user to take better pictures.

Financial Services: All financial industries and banks are using ML for two main reasons. To identify investment opportunities and prevent the fraud. Many banks are using complex algorithms to assess loan risk, and approve or deny based on their conclusion alone. The traditional loan officer is no longer needed, other than to pass along the decision to the client. One new way to manage the assets of a company or civilian are the Robo-Advisors and it is estimated that in the next five years they will manage the 17% of the total assets.

Health Care: Machine Learning now is a very trustworthy method to diagnose the symptoms and genetic details because it has the ability to compare these symptoms against millions of possible diagnoses. It determines the probability of each and most likely cause in a fraction of the time it takes a human doctor. In addition, ML helps the fast-growing health-care industry, all the wearable devices that are used in order to check patient's health are collecting data from the human body in real time. Then these data are analyzed with the help of machine learning and help the doctor to give the right diagnose and treatment. Last but not least, robot surgeons (robo-doctors) can perform operations with precision and very low possibility of error. Consequently, that means faster healing and less time in hospital for the patients.

Transportation: Data analysis and machine learning have brought us to self-driving cars. By analyzing data, transportation companies can identify trends and patterns. These two can help to make routes more efficient and predict problems. The cars with the latest technology can maintain speed, staying in the lane and slow down if it is necessary. Furthermore, cars with autopilot can change lanes, move from one highway to another, exit freeway and they have the ability to self-park. Autopilot feature uses a system of cameras that provide a 360-degree view around the vehicle to a long range. For example, autopilot can detect environmental conditions like dust, rain and react accordingly.

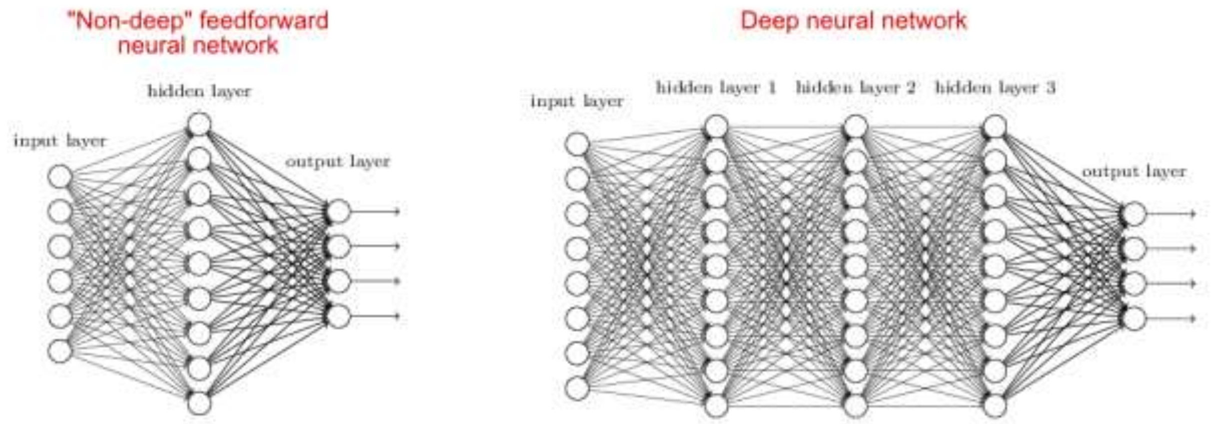
Gas & Oil: The main goal of machine learning in gas and oil industries is to find new energy sources. Also, the data analysis of the gas and oil can help the companies to refine their strategies. To be more specific, after the data analysis they can do drilling optimization, design and execute high-quality wells and reduce the nonproductive time with advanced drilling analytics. Last but not least, they forecast their production with a simple way they incorporate well surveillance and predictive models into forecasts in order to improve short and long term estimates of ultimate recovery.

1.2: Deep Learning

Deep Learning is a subfield of Machine Learning family based on base data representations. Unlike standard machine learning algorithms that break problems down into parts and solves them individually, deep learning solves the problem from end to end. Better yet, the more data and time you feed a deep learning algorithm, the better it gets at solving a task. Deep Learning architecture has been applied to field including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, drug design, bioinformatics and machine translation, where they produce results comparable to and in some cases superior to human experts.

Deep Learning:

- Learn in supervised and unsupervised patterns.
- Use some form of gradient descent for training via backpropagation.
- Use a sequence of multiple layers for feature extraction and transformation. Every successive layer uses the output from the previous layer as input.
- Learn multiple levels of representations that correspond to different levels of abstraction.



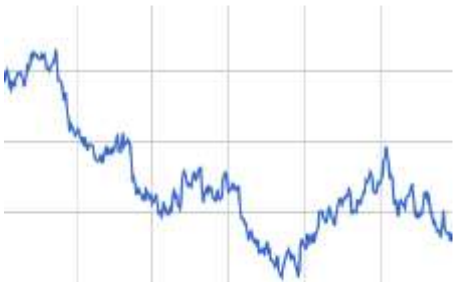
1.2.1: History

Deep Learning term was first introduced in machine learning community by Rina Dechter in 1986. In 2000 Igor Aizenberg and his colleagues has also introduced Deep Learning to Artificial Neural Networks community, in the context of Boolean threshold neurons through neural networks for reinforcement learning. Also, in 2006 Geoff Hinton, Osindero and the showed in their publication how a many-layered feedforward neural network could be effectively pre-trained one layer at a time, treating each layer in turn as an unsupervised restricted Boltzamn machine, then by using supervised backpropagation they fine-tuning it again.

In addition, another significant moment in Deep Learning history was a publication by Alexey Ivanenko and Lapa. He published the first general working learning algorithm for supervised, deep, feedforward, multilayer perceptions in 1965.

1.2.2: Neural Network

Neural Network (NN)s originate from an attempt to imitate information processing principles adopted by the brain. From a practical perspective biological realism would be difficult to emulate. The focus has instead been shifted towards creating models that mimic how biological systems process information but which also have been proven to have a great amount of practical value. A downside of NNs from the resource- constrained perspective is that they require a significant amount of resources if the network architecture is too large. This offers an interesting subject for evaluation as the size of the network will deeply affect the prediction accuracy, latency and memory footprint. These following sections will give an overview of the equations and mathematical theorems that is used when implementing the NN.



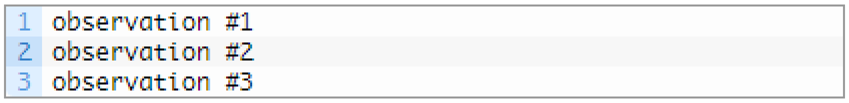
1.3: Time Series

Time Series is a series of data points indexed, listed or graphed in the time order with a natural temporal ordering. Most commonly, a time series is a sequence taken at successive equally spaced points in time. Time series are frequently plotted via line charts and can be applied to real-valued, continuous data, discrete numeric data, or discrete symbolic data. Time series analysis comprises methods for analyzing time series data in order to extract meaningful statistics and other characteristics of data. Time series analysis can be utilized in any field where understanding trends is important to decision making and reacting to changes in behavioral patterns. The most frequent examples of using time series are in any domain in applied sciences. Some examples are:

Statistics	Signal Processing	Pattern Recognition
Econometrics	Mathematical Finance	Weather Forecasting
Earthquake Prediction	Electroencephalography	Control Engineering
Astronomy	Communications Engineering	

1.3.1: What makes time series dataset deferent from other datasets?

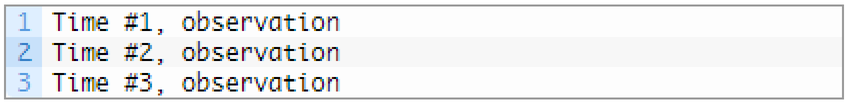
A typical machine learning dataset is a collection of observations. Time does play a role in normal machine learning datasets. Predictions are made for new data when the actual outcome may not be known until some future date. The future is being predicted, but all prior observations are almost always treated equally. Perhaps with some very minor temporal dynamics to overcome the idea of "concept drift" such as only using the last year of observations rather than all data available.



```
observation #1
observation #2
observation #3
```

1.3.1.1: Time series dataset

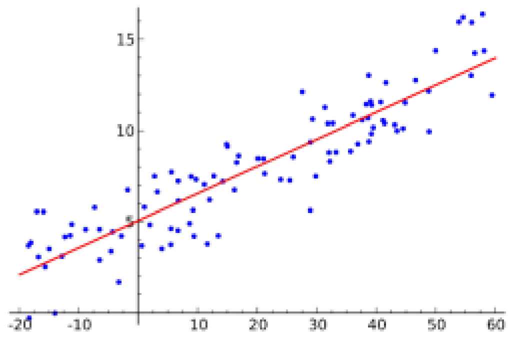
Time series adds an explicit order dependence between observations and this is a time dimension. This additional dimension is both a constraint and a structure that provides a source of additional information.



```
Time #1, observation
Time #2, observation
Time #3, observation
```

1.3.1.2: Time series forecasting

Time series forecasting is what we are going to use in our thesis. This type of data analysis uses a model to predict future values based on previous values. While Regression Analysis is specialized to test theories that the current values of one or more independent time of time series affect the current value of another time series.



1.4: Regression Model

In statistical modeling, regression analysis is a set of statistical processes for estimating the relationships among variables. It includes many techniques among variables. Regression can analyze several variables, when the focus is on the relationship between a dependent variable and one or more independent variables.

1.4.1: Linear Regression

Linear regression is a linear approach to modelling the relationship between scalar response and one or more explanatory variables. Simple linear regression is the case of one explanatory variable. For more than one explanatory variable, the process is called multiple linear regression. Linear regression uses linear predictor functions whose unknown model functions are estimated from the data. Linear regression focuses on the values of the predictors, rather than on the joint probability distribution of all of these variables, which is the domain of the multivariate analysis.

Linear regression used extensively in practical applications. The reason why is because models which depend linearly on their unknown parameters are easier to fit than models which are non-linearly related to their parameters and the statistical properties of the resulting estimator are easier to determine. Linear regression models are often fitted using the least squares approach, but they may also be fitted in other ways, such as by minimizing the "lack of fit" in some other norm (as with least absolute deviations regression), or by minimizing a penalized version of the least squares cost function as in ridge regression (L2-norm penalty) and lasso. The least squares approach can be used to fit models that are not linear models. Although the terms least squares and linear model are linked closely.

1.4.2: Lasso Regression

Least absolute shrinkable and selection operator (Lasso) is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces. The lasso does this by imposing a constraint on the model parameters that causes regression coefficients for some variables to shrink toward zero. Variables with a regression coefficients equal to zero after the shrinkage process are excluded from the model. Variables with non-zero regression coefficients variables are most strongly associated with the response variable.

Lasso was originally formulated for least squares models. This reveals a substantial amount about the behavior of the estimator, including its relationship to ridge regression and the best subset selection. Through originally defined least squares, lasso regularization is easily extended to a wide variety of statistical models including generalized linear models, generalized estimating equations, proportional hazards models, and M-estimators. Lasso was originally introduced in literature in 1986. Later, independently rediscovered and popularized in 1996 by Robert Tibshirani.

1.4.3: Ridge Regression

Is a technique for analyzing multiple regression data suffer from multicollinearity. Multicollinearity is the existence of near-linear relationships among the independent variables. When it occurs, least squares are unbiased, but their variances are large so they may be far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors. In ridge regression, one have to first standardize the variables dependent and independent by subtracting their means and dividing by their standard deviations.

1.4.4: Logistic Regression

Logistic model is a widely used statistical model that in its basic form, uses a logistic function to model a binary dependent variable. Logistic regression is estimating the parameters of a logistic model. It is a form of binomial regression. Binary logistic models are depending on two possible values "0" and "1". These values represent other possible values such as pass/fail, win/lose, healthy/sick. In the logistic model the logarithm of the odds for value labeled "1" is a linear combination of one or more independent variables. Independent variables can each be a binary variable or a continuous variable. The corresponding probability of the value labeled "1" can vary between 0 and 1. The function that converts the logarithm of the odds scale is called logistic function.

Logistic regression was developed by statistician David Cox in 1958. The binary logistic regression model has extensions to more than two levels of the dependent variable:

- Multinomial logistic regression, models categorical outputs with more than two values.
- Ordinal logistic regression, if the multiple categories are ordered.

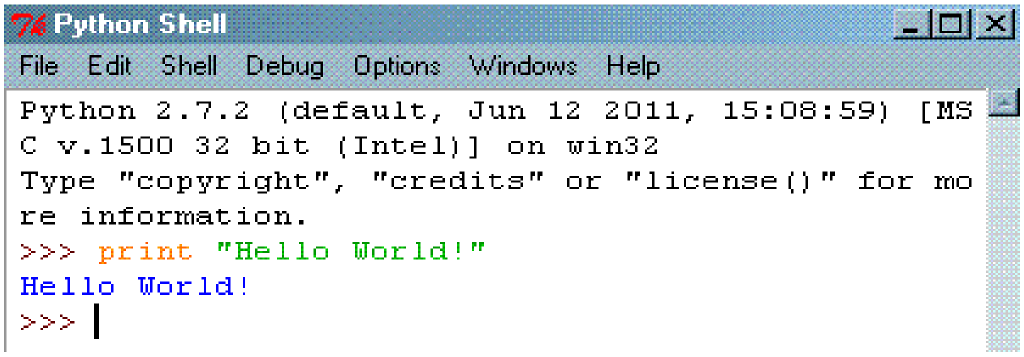
The model itself simply models probability of output in terms of input, and does not perform statistical classification (it is not a classifier), though it can be used to make a classifier.

1.5: Python

Python realized in 1991. The historical development of Python starts before the official release in 1989 as a hobby project. Python is a famous high-level programming language for general purpose programming. The creator of Python is a Dutch programmer Guido Van Rossum who is famous at the Python community as Benevolent Dictator for Life, which means that he continues to oversee the Python development process. Python is an interpreted language and has design philosophy that emphasizes code readability. Her syntax allows programmers to express concepts in fewer line code than other languages, like Java or C and as a result the cost of program maintenance is getting the lesser and easier. Python's structure gives the opportunity to the programmers to programming in large or small scales.

Python is an object-oriented, imperative, functional and procedural language with dynamic semantics. Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach. To elaborate further, with the help of third-party modules Python can provide to programmers functionalities like:

- Web and Internet Development
- Desktop Graphic User Interfaces
- Network Programming
- Software & Game Development
- Scientific & Numeric
- Database Access
- Education



```
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MS
C v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for mo
re information.
>>> print "Hello World!"
Hello World!
>>> 
```

1.5.1: Python Use Cases

1.5.1.1: Script Writing and Automation

Python is a scripting language which allows writing scripts for parsing a text file, generating sample inputs to test an application, content scrapping from blogs and sites. Also, Bash scripts are easily and painlessly replaced with Python scripts. In addition, Python provides many tools such as Fabric, Salt or Ansible to automate repetitive and basic processes like mass mail-send outs and other deployments.

1.5.1.2: Back-End Development

Writing APIs and interacting with the database can easily achieved with Python. Many famous sites such as Instagram, Quora and DropBox use Python for back-end integrations and development. The most famous back-end development tools are Django and Flask.

1.5.1.3: Data Science and Machine Learning

As Python is open source and flexible language there are many libraries for data analysis, manipulation and visualization. Python libraries such as pandas, NumPy, scikit-learn and others bring features from R and MATLAB to Python development. These features can be data frames, modeling and matrix operations. Last but not least, Python features a plethora of libraries to build and implement machine learning algorithms. Some libraries are PyBrain, OpenCV, SimpleCV, Pylearn2.

1.5.1.4: Internet of Things (IoT)

With the help of Python, it is easy to build IoT and connected solutions and products like robots, home automation, smart agriculture and more. This phenomenon happens because Python is a go-to language for Rasberry PI and other microprocessors.

1.5.2: Interpreted languages

Interpreted programming languages do not need to be compiled before the program executes. The program executed directly by the interpreter and each statement is translated into a sequence of one or more subroutines and then into another language. An interpreter is a computer program that directly executes instructions written in a programming language. Interpreter in general uses the following steps for program execution:

- parse the source code and perform its behavior directly
- translate source code into some efficient intermediate representation and immediately execute it

- explicitly execute stored precompiled code made by a compiler which is part of the interpreter system

Practically interpreted languages can be contrasted with machine languages because execution and interpretation mean the same thing (fetch the next instructions from the program and execute them).

1.6: Pandas Software

In Computer programming pandas is a software library written for the Python programming language for data manipulation and analysis. Pandas, offer data structures and operations for manipulating numerical tables and time series. The creator of pandas is Wes McKinney who is an American data scientist. He started working on pandas in 2008 while at AQR Capital Management out of need for financial data. Before leaving AQR he was able to convince management to allow him to open source the library. Now it is a free software library and the name is derived from the term “panel data”, an econometrics term for data sets that include doth time series and cross sectional data.

1.6.1: Pandas features

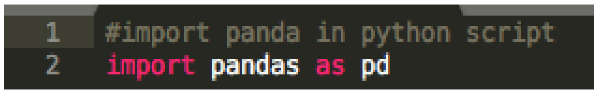
Pandas have multiple functionalities and features, some of them are:

- Time series functionality: Date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging.
- Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
- Data alignment and integrated handling of missing data.
- Group by engine allowing split-apply-combine operations on data sets.
- DataFrame object for data manipulation with integrated indexing.

- Tools for reading and writing data between in-memory data structures and different file formats.
- Reshaping and pivoting of data sets.
- Label-based slicing, fancy indexing, and subsetting of large data sets.

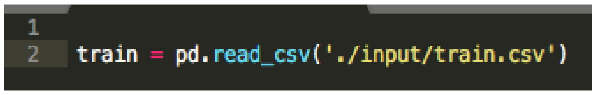
1.6.2: Example with pandas

- To begin with the most important is to import pandas in python code, we can see the way below:



```python
#import panda in python script
import pandas as pd
```

- One common function is .read_csv which allows to load files of comma separated values. For example:



```python

train = pd.read_csv('./input/train.csv')
```

1.7: PyCharm

PyCharm is a cross integrated development environment (IDE) platform. An IDE is the primary tool for any professional programmer. Normally, when we think of an IDE, we think of an enhanced text editor with a built-in compiler like Visual Studio, Eclipse, and IntelliJ. In addition, the usual syntax highlighting and programming indentation happened during the compilation which points out mistakes as well as code formatting hints and refactoring. While you don't compile Python code the same way you do Java or C#, PyCharm acts like a true IDE for Python. It contains work flows for creating and maintaining virtual environments, a visual counterpart for the PEP library manager, linters that keep your code compliant with PEP 8 standards and a host of other handy general development tools. Linters is a tool that analyzes source code to flag programming errors, bugs, stylistic errors while PEP 8 is a style guide for Python code

Typical IDEs also have a nice file and project management system in them and PyCharm is no exception. You have the ability to work with databases and leverage code templates for built in support for a variety of popular Python project types. If you're doing scientific, quantitative, or big data work, PyCharm has support for working with Jupiter notebooks. It is agnostic as to which Python environment you work with so you can work with Vanilla Python or Anaconda. PyCharm supports many of the major version control system like Git. PyCharm has tools for deploying your code to servers via SSH and it even has support for working with vagrant or virtualization tools like Docker that allow you to test your code in production-like environments. Speaking of testing, PyCharm also supports all major testing platforms for Python with the ability to run and view your tests with the familiar green bar display common to most testing tools.

1.8: Jupyter

The Jupiter Notebook is an interactive web-based computing environment that enables users to author notebook documents that include: Live Code, Interactive widgets, Plots, Narrative text, Equations, Images, Video. Jupiter Notebook was a very helpful tool for the development of this thesis because it helped in order to run all the code parts independently.

The Jupiter Notebook combines three components:

1.8.1: The notebook web application

Is an interactive web application for writing and running code interactively and authoring notebook documents which enables users to:

- Edit code in browser, with automatic syntax highlighting, indentation and tab completion/introspection.
- Run code in browser, with the results of computation attached to the code which generate them.

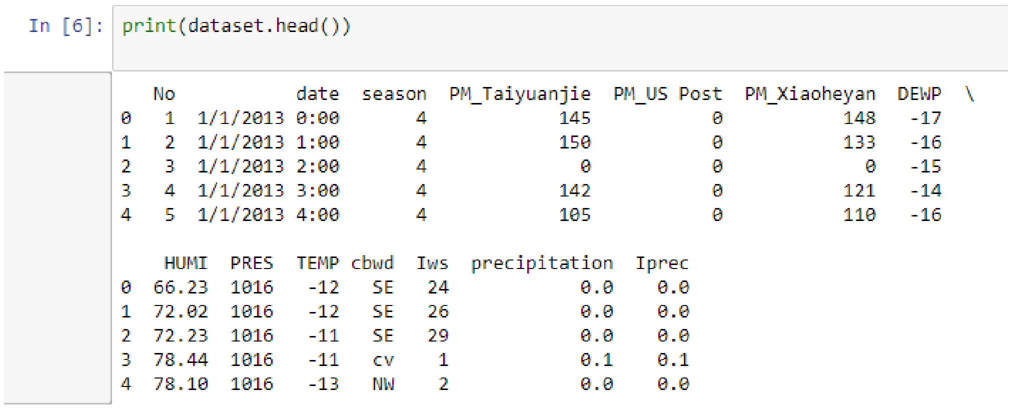
- See the results of computations with rich media representations, such as HTML, PNG,PDF.
- Author narrative text using the Markdown markup language.
- Include mathematical equations using LaTeX syntax in Markdown, which are rendered in-browser by MathJax.

1.8.2: Kernels

Kernels separate processes started by the notebook web application that runs users' code in a given language and returns the output back to the notebook web application. The kernel also handles things like computations for interactive widgets, tab completion and introspection. Through Jupyter's kernel and messaging architecture, the Notebook allows code to be run in a range of different programming languages. For each notebook document that a user opens, the web application starts a kernel that run the code for that notebook. Also kernels are available in the following languages:

Scala	**Python**	**R**	**Ruby**
Julia	**Node.js**	**Go**	**Haskell**

Below are printed the source thesis data in different segment than the other parts of code.



```
In [6]: print(dataset.head())
```

```
   No           date  season  PM_Taiyuanjie  PM_US Post  PM_Xiaoheyan  DEWP  \
0   1  1/1/2013 0:00       4            145           0           148   -17
1   2  1/1/2013 1:00       4            150           0           133   -16
2   3  1/1/2013 2:00       4              0           0             0   -15
3   4  1/1/2013 3:00       4            142           0           121   -14
4   5  1/1/2013 4:00       4            105           0           110   -16

    HUMI  PRES  TEMP cbwd  Iws  precipitation  Iprec
0  66.23  1016   -12   SE   24            0.0    0.0
1  72.02  1016   -12   SE   26            0.0    0.0
2  72.23  1016   -11   SE   29            0.0    0.0
3  78.44  1016   -11   cv    1            0.1    0.1
4  78.10  1016   -13   NW    2            0.0    0.0
```

1.8.3: Notebook documents

Contain the inputs and outputs of an interactive session as well as narrative text that accompanies the code but is not meant for execution. Rich output generated by running code, including HTML, images, video, and plots, is embedded in the notebook, which makes it a complete and self-contained record of a computation. By running a notebook web application on one's computer, notebook documents are just files on one's local filesystem with a .ipynb extension. This allows one to use familiar workflows for organizing your notebooks into folders and sharing them with others.

Notebooks consist of a linear sequence of cells. There are four basic cell types:

- **Code cells:** Input and output of live code that is run in the kernel.
- **Markdown cells:** Narrative text with embedded LaTeX equations.
- **Heading cells:** 6 Levels of hierarchical organization and formatting.
- **Raw cells:** Unformatted text that included, without modification, when notebooks are converted to different formats using nbconvert.

In the heart of notebook documents there are JSON data with binary values. As a result, this allows them to be read and manipulated programmatically by any programming language.

1.9: NumPy

NumPy is a library, for scientific computing in Python. This fundamental package provides array object, various derived objects, and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operation and random simulation. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In NumPy's core is the ndarray object. This object encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several significant differences between NumPy arrays and the standard Python sequences:

- NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an ndarray will create a new array and delete the original.

- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception is that one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.
- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.
- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today's scientific/mathematical Python-based software, just knowing how to use Python's built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

1.9.1: History

Python was not initially designed for numerical computing. Although, the scientific and engineering community was attracted early on and a special interest group was founded in 1995 with the scope to define an array computing package. Python's designer Guido Van Rossum was member of that development team to implement Python's syntax extension.

Jim Fulton completed a matrix package implementation of a matrix package which generalized by Jim Hugunin to become Numeric, which is variously called NumPy (Numeric Python). Hugunin, a graduate student at (MIT) Massachusetts Institute of Technology joined the Corporation for National Research Initiatives (CNRI) to work on JPython in 1997 leaving Paul Debois to take over as maintainer.

Numarray was written as a new package to replace Numeric. Numarray was fast in operating large arrays, but slower in small ones. The last version of Numeric v24.2 was released on 11 November 2005 and Numarray v1.5.2 was released on 24 August 2006. In 2005 NumPy developer

Travis Oliphant ported the Numarray's features in Numeric to unify the community by releasing NumPy 1.0 in 2006. This new project was part of SciPy. To avoid installing the large SciPy package just to get an array object, this new package was separated and called NumPy.

1.10: Particulate Matter (PM)

PM

PM (Particulate Matter) known as particle pollution or atmospheric aerosol particles is a composite mixture of microscopic solid airborne particles and liquid matter. These liquid droplets are consist of acids like nitrates and sulfates, water, ammonium, organic chemicals, black carbon, metals and soil material.

There are two groups of particle pollution:

1. **Coarse Particles PM 10-2.5 :**

   These particles are frequently found near roadways and dusty industries. Their diameter is from 2.5 to 10 micrometers. All the particles between between 2.5-10 micrometers in size are placed in the coarse particle category.
2. **Fine Particles PM 2.5 :**

   These particles have diameters less than 2.5 micrometers and can be found in smoke and haze.

The PM2.5 particles which are those we are going to analyze are grouped in two categories, the primary and the **secondary.** The PM2.5 particles which are known as primary are those which are directly emitted into as solid and liquid particles. The secondary are those which are formed by chemicals reactions of gases in the atmosphere.

Primary fine particles have as major sources cars and trucks gases, open burning, wildfires, woodstoves, cooking, dust from roads, constructions, agricultural operations and oil burning boilers. Major sources of the secondary fine particles are power plants and some industrial processes, including oil refining and pulp and paper production.

1.10.1: PM Effects

The negative effects of the PM (Particulate Matter) are multiple in both humans and in the environment. To start with, the health impacts are unprecedented. The small particles which are less than 10 micrometers in diameter cause huge problems because they stored easily in the deepest part of the lungs, the larger ones are filtered in the nose and throat. PM2.5 are tend to penetrate into the gas exchange regions of the lung which are called alveolus; and the very small particles may affect other organs of the human body. The size of the particles is not the only characteristic that makes a particle to penetrate into the lungs, other characteristics are the shape and the chemical composition. Also the exposure on a regular basis can affect the human heart too. Some main problems that are caused because of the particle pollution exposure are: premature death in people with heart or lung disease, irregular heartbeat, nonfatal heart attacks, aggravated asthma, increased respiratory symptoms, such as irritation of the airways, coughing or difficulty breathing, decreased lung function.

According to the World Health Organization estimations in 2005 about the 3% of mortality from cardiopulmonary disease caused by fine particulate air pollution, also the 5% of mortality from cancer of trachea, lung and bronchus was because of the fine particulate air pollution too.

In addition, particles afflict the environment too. Particles with the help of wind can be carried over long distances and then settle on ground or water. Depending on their chemical composition, the effects of this settling may include: making lakes and streams acidic, changing the nutrient balance in coastal waters and large river basins, depleting the nutrients in soil, damaging sensitive forests and farm crops, affecting the diversity of ecosystems, contributing to acid rain effects.

1.10.2: PM2.5 in China

Energy consumption is the main impact of China’s industrialization. According to National Bureau of Statistics, China in thirteen years has increase the coal equivalent consumption from 1555 million tons per year to 4260 tons. This massive increase in energy consumption related with the luck of strict environmental laws have bring very harmful results to atmosphere. One main constituent of air pollution as mentioned before is particular matter.

This big air pollution of China’s atmosphere drive the U.S Embassy in Beijing to collect the hourly PM2.5 $\mu g/m^3$ hourly from 2008. Four more cities followed up and one of them is Shenyang which is the one we analyze in this thesis. Data reliability for all these cities comes from three analyses:

- Comparing the frequencies of occurrence of various PM2.5 concentration ranges and their duration.
- Checking on the air quality assessment obtained from the U.S. posts and the MEP sites.
- Investigating the effect of winter heating in Beijing and Shenyang, the two cities which have centrally controlled heating.

1.11: Keras

Keras is an open source framework for building deep neural networks with Python. In addition, Keras enables as to build deep learning systems with little complexity and with a few lines of code. This library is capable to run on top of TensorFlow, Theano, Microsoft Cognitive Toolkit. The primary author and maintainer of Keras is François Chollet. 2017 was a great year for Keras Library and the reason is that Google’s Tensorflow team decided to support Keras in TensorFlow’s core library. The big win of Keras is that makes easy to develop deep learning models regardless of the computational backed used.

Keras deep learning library provide to us:

- Easy and fast prototyping through user friendliness, modularity, and extensibility.
- Both convolutional and recurrent networks as well as the combination of the two.
- Great compatibility with strong or weak CPU & GPU.

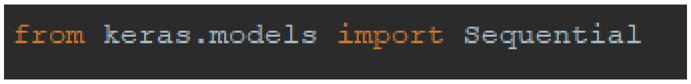
To elaborate further, Keras library use two types of models, the first is the sequential models and the second is the non-sequential (Functional) models. Last but not least, minimizes the number of user actions required for common use cases and is easy to add new classes and functions as long as other existing modules.

1.11.1: Keras Functional model

The Keras functional API provides a more flexible way for defining models. Specifically, it allows you to define multiple input or output models as well as models that share layers. Moreover, it allows you to define ad hoc acyclic network graphs. Models are defined by creating instances of layers and connecting them directly to each other in pairs, and then defining a Model that specifies the layers to act as the input and output to the model, via the parameters inputs and outputs, respectively.

1.11.2: Keras Sequential model

Sequential model is the one this thesis use for our machine learning prediction. In order to use this model in our machine learning code import it with the way below:



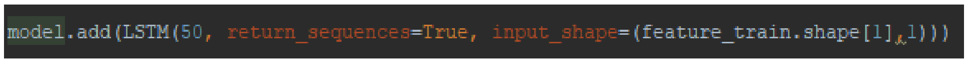
```
from keras.models import Sequential
```

This model is a simple stack of sequential layers. To build a sequential model is mandatory to follow the below steps:

- Instantiate a Sequential model.
- Add layers to it one by one using add module.
- Compile the model with a normal loss function, an optimizer and optional evaluation metrics.
- Use data to fit the model.
- Evaluate mode, persist or deploy model, start new experiment.

1.11.3: LSTM Network

Long short-term memory units are units of a recurrent neural network. An RNN composed of LSTM units often called an LSTM network. This type of network is great for classifying, processing and making predictions based in time series data and this is the main reason why this network is used in that thesis. The main idea of LSTM development is to deal with exploding and vanishing gradient problems that can be encountered when train Recurrent Neural Networks. The way to make an LSTM network with python code is shown below.

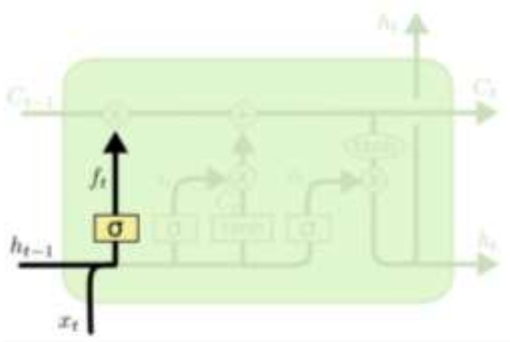


```
model.add(LSTM(50, return_sequences=True, input_shape=(feature_train.shape[1],1)))
```

The main concept of LSTM's are the cell state and its various gates. The cells state act as a transport highway that transfers relative information all the way down the sequence chain. The cell can carry relevant information throughout the processing of the sequence. So even information from the earlier time steps can make its way to later time steps, reducing the effects of short-term memory. As the cell state goes on its journey, information get's added or removed to the cell state via gates. The gates are different neural networks that decide which information is allowed on the cell state. The gates can learn what information is relevant to keep or forget during training.

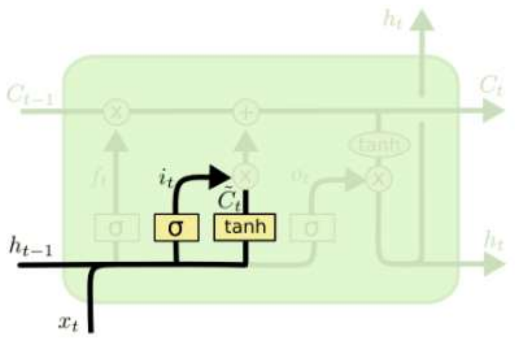
To give a more laborious view of the LSTM network the step by step process happens as follows.

1. LSTM's first step is to decide what information is going to through away. Sigmoid layer is responsible for throwing away that information. These layers are squishing values between 0 and 1. The zeros are the one that are getting "forgotten" and ones are the information that stays in the network. The sigmoid layer takes the inputs X(t) and h(t-1) and is called forgot gate. The output of this gate is the multiplication C(t-1) and f(t). C(t-1) is the cell state from the previous LSTM network.



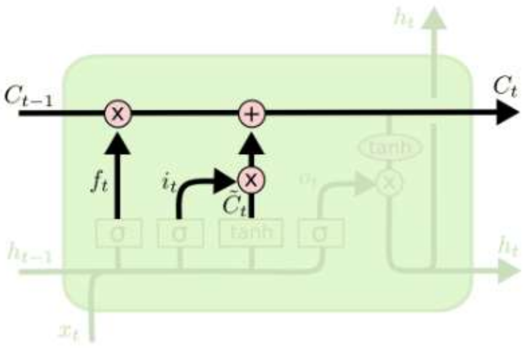
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

2. The next step is the decision of what information is going to stay in the network. That process is separated in two parts. It starts with one more sigmoid layer which is called the input gate layer and it decides which values will be updated. The second step creates a vector which contains new values with the help of tahn layer. Tanh function is a rescaled version of the sigmoid, and its output range is [− 1,1] instead of [0,1].



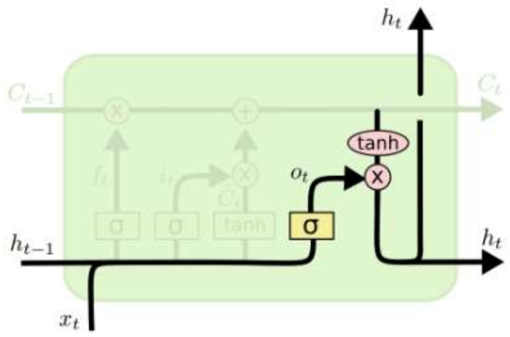
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

The combination of the two previous steps creates an update to state from C(t-1) to Ct.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

3. The last step is to define the output. It starts running a sigmoid layer which decides what information will be as output. Afterwards information is pushed in tahn layer which changes all the values to be between [-1,1] and then the cell state is multiplied by the output of the sigmoid gate.



$$o_t = \sigma\left(W_o\,[h_{t-1}, x_t] + b_o\right)$$
$$h_t = o_t * \tanh(C_t)$$

1.12: TensorFlow

As mentioned before, Keras is capable to run many backend engines and Tensorflow is the one we use for our model prediction. Tensorflow, is the default backend engine of Keras and is configured in the installation in a json file as follows:

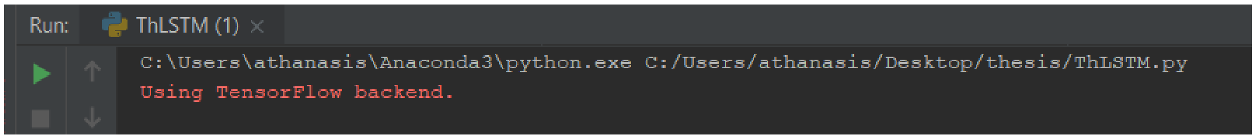
```
{
    "image_data_format": "channels_last",
    "epsilon": 1e-07,
    "floatx": "float32",
    "backend": "tensorflow"
}
```

Additionally, TensorFlow is an open source software library for high performance numerical computation. The historical context of TensorFlow starts in 2011. Google Brain team built DistBelief, a proprietary machine learning system based on deep learning neural networks. The big evolution of DistBelief codebase assigned by Google and many computer scientists; the result was a faster, more rebost application-grade library, which became TensorFlow. Both computer operation systems like Linux, macOs ,Windows and mobile computing platforms like Android and iOS are able to run TensorFlow and the reason why is that its flexible architecture allows multiple hardware to support this library.

There is a great number of TensorFlow use cases but the main can be count as follows:

- Voice and Sound Recognition
- Text Based Applications
- Image Recognition
- Time Series
- Video Detection

In terms of our thesis, Tensorflow is used as backend. There no modifications in the tensorflow functionalities from us. Keras is the one who modifies TensorFlow in order to satisfy the needs of our model. By running our model and import all libraries TensorFlow backend activated before the first lines of model start their execution.



```
C:\Users\athanasis\Anaconda3\python.exe C:/Users/athanasis/Desktop/thesis/ThLSTM.py
Using TensorFlow backend.
```

1.13: Matplotlib

Is a plotting library for Python programming language. Matplotlib provides Python 2D plotting in a variety of hardcopy formats and interactive environments across platforms. Object-oriented API of Maplotlib gives the flexibility to embed plots into applications using GUI toolkits. Users can generate plots, histograms, power spectra, bar charts, error charts, scatterplots with few lines of code. One main plotting module of Matplotlib is pyplot. This module provides a MATLAB-like interface as long as many same functionalities as MATLAB. Matplotlib helped to visualize the data providing a more vivid site about this thesis.

1.14: Scikit-learn

Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy. The one module we are using in our python model is the MinMaxScaler. It provides data normalization between the limits one set. The parameter which makes this data modification is “feature_range”. Here one can set minimum and maximum value of data transformation.

1.15: Anaconda

Anaconda is a free open source platform which provides package management and deployment for R and Python. It contains over 1,400 data science packages and it is supported for all operation systems. In addition, it is used for machine learning applications , predictive analytics and large scale data processing. The embedded Navigator of Anaconda contains applications which all are used variously for Python and R tasks. This applications are:

- JupyterLab
- Jupyter Notebook
- Spyder
- Rstudio
- Visual Studio Code

In terms of this thesis the application which was used was Jupiter Notebook. It was helpful for debugging and run each part of code separately. Also all the libraries that are used in the model development are supported from Anaconda’s library repository.

Chapter 2: Method

2.1: Python files

The development of the machine learning model took place in two python files. The first is the "ThLSTM" and the second is the "HelpingFunctions".To elaborate further, "ThLSTM" is the executable file, it gets the data as import and is calling the "HelpingFunctions" to continue the data preparation and the prediction process with functions that are responsible for these actions.

2.2: Import Libraries-Modules

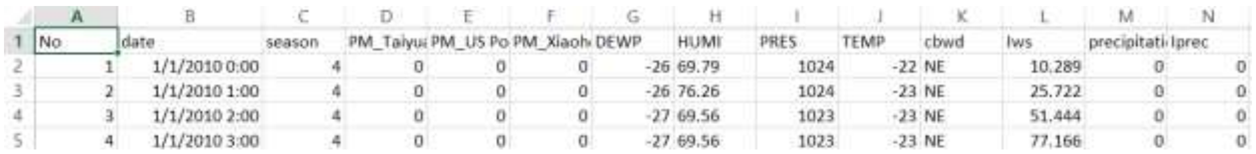
This section is mandatory, it is building the reference about the modules and libraries that are going to be used in code execution. Python programming language can achieve that by using the word "import". The libraries as shown in the source code below which have analyzed in the previous sections are necessary to achieve a good prediction with Keras API.

```
import sys
import numpy as np # linear algebra
from keras.layers import BatchNormalization
from numpy import newaxis
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM, GRU
from keras.models import Sequential
from keras import optimizers
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
from HelpingFunctions import * #we import our Helping Function file that contains all the
functions
from sklearn.metrics import r2_score
```

2.3: Dataset

The dataset can be found in Machine Learning Repository (UCI) site. This specific dataset contains Beijing, Shanghai, Guangzhou, Chengdu and Shenyang PM2.5 data. Moreover, it contains meteorological data for each of the above cities. Furthermore, this specific dataset is a multivariate time series one. Time series dataset means that all data points in the file are indexed in time order.

The original dataset of the Shenyang city has the following structure.

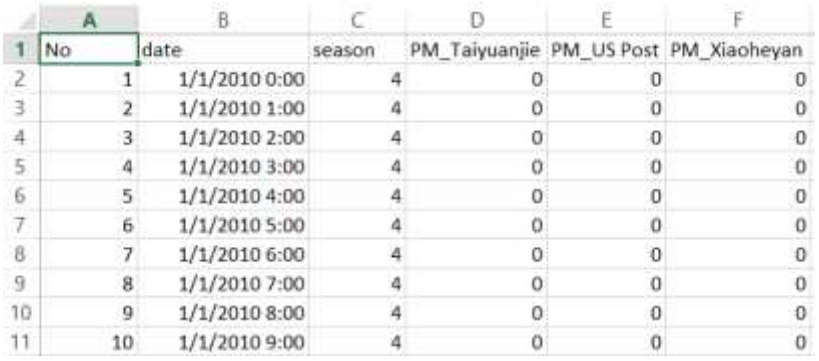


	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	No	date	season	PM_Taiyu	PM_US Po	PM_Xiaoh	DEWP	HUMI	PRES	TEMP	cbwd	Iws	precipitati	Iprec
2	1	1/1/2010 0:00	4	0	0	0	-26	69.79	1024	-22	NE	10.289	0	0
3	2	1/1/2010 1:00	4	0	0	0	-26	76.26	1024	-23	NE	25.722	0	0
4	3	1/1/2010 2:00	4	0	0	0	-27	69.56	1023	-23	NE	51.444	0	0
5	4	1/1/2010 3:00	4	0	0	0	-27	69.56	1023	-23	NE	77.166	0	0

In order to have a deeper insight of the above attribute information all data columns can be analyzed respectively as follows:

• No: row number • date: year/ month/day hour:minutes • season: season of data in this row • PM: PM2.5 concentration (ug/m^3) in PM_Taiyuanjie • PM: PM2.5 concentration (ug/m^3) in PM_US Post • PM: PM2.5 concentration (ug/m^3) in PM_Xiaoheyan	• DEWP: Dew Point (Celsius Degree) • TEMP: Temperature (Celsius Degree) • HUMI: Humidity (%) • PRES: Pressure (hPa) • cbwd: Combined wind direction • Iws: Cumulated wind speed (m/s) • precipitation: hourly precipitation (mm) • Iprec: Cumulated precipitation (mm)

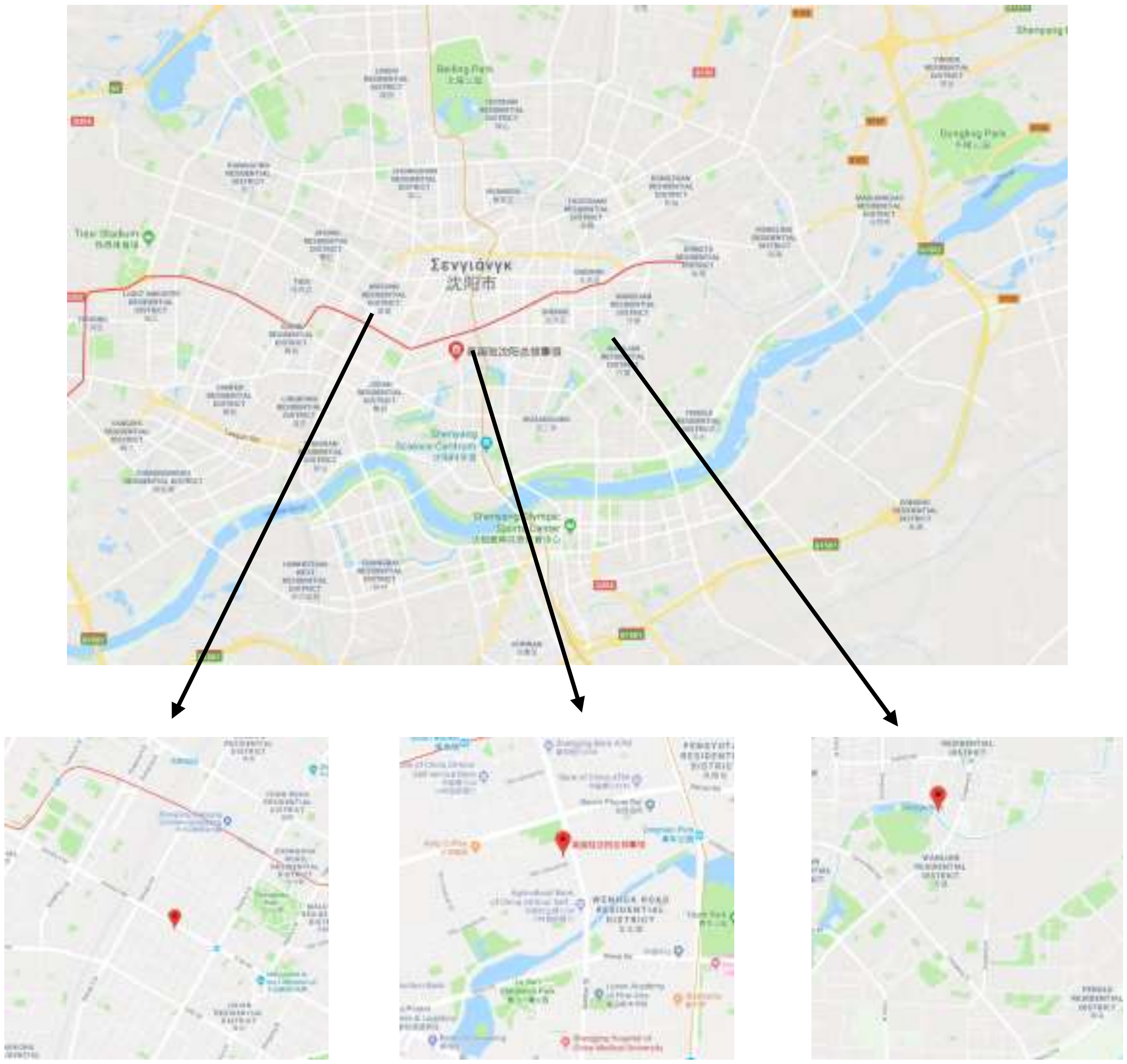
However, this thesis is focusing in Shenyang's city PM2.5 data and it's measuring stations. The reason why is to have deeper look in the main scope of thesis which is to build a model in time series and it's structure and not to focus in the data variety. The modified dataset contains historical data of Particular Mater mass in Shenyang's atmosphere and the specific position of measurements stations are Taiyuanjie, US Post and Xiaoheyan which provide the hourly data of air pollution. According to UCI Machine Learning Repository site this dataset has many missing values. It is obvious that the first part of the dataset has many missing data measurements .Consequently, there was no reason to use them because the prediction would be less accurate and it would be more time consuming to add more data in the whole process. These data values are equal to zero as shown below.



	A	B	C	D	E	F
1	No	date	season	PM_Taiyuanjie	PM_US Post	PM_Xiaoheyan
2	1	1/1/2010 0:00	4	0	0	0
3	2	1/1/2010 1:00	4	0	0	0
4	3	1/1/2010 2:00	4	0	0	0
5	4	1/1/2010 3:00	4	0	0	0
6	5	1/1/2010 4:00	4	0	0	0
7	6	1/1/2010 5:00	4	0	0	0
8	7	1/1/2010 6:00	4	0	0	0
9	8	1/1/2010 7:00	4	0	0	0
10	9	1/1/2010 8:00	4	0	0	0
11	10	1/1/2010 9:00	4	0	0	0

Dataset provides data until the start of 2016 and with the missing data our training set starts at 2013.

The three areas of Shenyang City can be found in the below map.


Σενγιάνγκ
沈阳市
Tiexi Stadium
Shenyang Science Centre
Shenyang Olympic Sports Center
WENHUA ROAD RESIDENTIAL DISTRICT

Taiyuanjie US Post Xiaoheyan

2.4: Input Data

To start editing the data we had first to input them. By using pandas read_csv function we are able to start prepare our data in python IDE by putting the directory of the file location. One mandatory part is to define the separator of the comma separated file. In our data set in the character is the semicolon “;”.

```
#Load the data
dataset = pd.read_csv('. /ShenyangECXELPM20100101_20151231---V2).csv',sep=";")
```

To predict the average PM 2.5 of Shenyang we calculate the average of the three areas per day and we create a new array to use it as the training and test set. To reach that point we created three new numpy arrays which contain the data of Taiyuanjie, US Post, Xiaoheyan respectively. After creating the numpy arrays we change them into regular lists in order to find correctly the average numbers. The average number it depends on the number of zeros on each hour. If one station has zero value then the average value is calculated from the other two stations; if all stations are not zero then the average value is calculated from all three. In other cases the average value will be calculated accordingly. The result average array of these three areas are calculated in the code section below in the new PM dataframe (newPMdf).

```
#Here we create three arrays with all PM station measurement of Shenyang and covert them to
#lists so then we can edit them one by one
one=np.array(dataset["PM_Taiyuanjie"])
one1 = one.tolist()
two= np.array(dataset["PM_US Post"])
two1 = two.tolist()
three = np.array(dataset["PM_Xiaoheyan"])
three1 = three.tolist()

#In order to calculate the average value of Shenyang we take the inputs of each station that have
non zero values
#and divide them with the sum of the station that has non zero values
PM_Shenyang = []
for i in range(len(one1)):
  #Here we calculate the average if one station has 0 measurment
  if (one1[i] == 0 and two1[i] != 0 and three1[i] != 0) or (one1[i] != 0 and two1[i] == 0 and three1[i]
      != 0) or ( one1[i] != 0 and two1[i] != 0 and three1[i] == 0):

    PM_Shenyang.append((one1[i] + two1[i] + three1[i])/2)
```

```
        #Here we calculate the average if there is no zero values in station
        elif (one1[i] != 0 and two1[i] != 0 and three1[i] != 0):
          PM_Shenyang.append((one1[i] + two1[i] + three1[i]) / 3)

        #Here we calculate the average if 2 of the station has zero values
        elif (one1[i] == 0 and two1[i] == 0 and three1[i] != 0) or (one1[i] == 0 and two1[i] != 0 and
            three1[i] == 0) or (one1[i] != 0 and two1[i] == 0 and three1[i] == 0):
              PM_Shenyang.append(one1[i] + two1[i] + three1[i])

        #Here we calulate the average if 3 of the station has zero values
        elif (one1[i] == 0 and two1[i] == 0 and three1[i] == 0) or (one1[i] == 0 and two1[i] == 0 and
            three1[i] == 0) or (one1[i] == 0 and two1[i] == 0 and three1[i] == 0):
              PM_Shenyang.append(0)

      PM_Shenyang1 = np.array(PM_Shenyang)
      #In order to overtake the error values we replace all the values over 750 with the number 400
      PM_Shenyang1[PM_Shenyang1 > 750] = 400
      print("PMnumbers ", PM_Shenyang)
```

2.5: Data Normalization

The data scale is strongly related with prediction. The big scale of a data set are able to affect negatively the prediction output. Our PM 2.5 dataset had the same issue due to the great data value deferential. As a result, to optimize the output of prediction we normalize the data with the help of scikit learn library. The module of this library is the MinMaxScaler. It provides the ability to set the range of the biggest and the smallest value of the dataset; all the values are going to be modified respectively in the range of [0-1] as the source code shows.

```
      sc = MinMaxScaler(feature range=(0, 1))
```

2.6: Splitting Data

Usually, the data are used are split into training data and test data. The training set contains a known output and the model learns on this data in order to be generalized to other data later on. Our choice was to use the 66% of the data set to train the model and the 34% for testing. In real world that means that the testing data is around 9 months. With the help of result length array we split the data into two arrays.

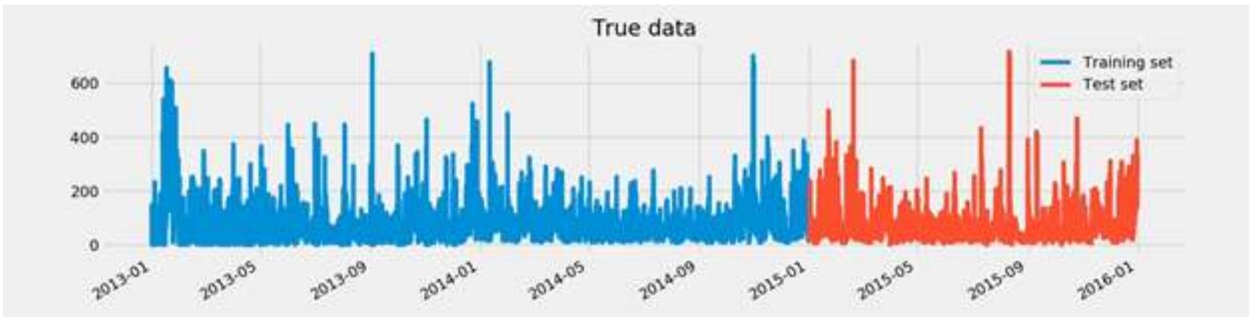
```
train_size = int(len(result) * 0.66)
train, test = result[0:train_size], result[train_size:len(result)]
```

After defining the length of the training and test set we fill four new numpy arrays as shown below.

- x_train: This numpy array contains all the input data that are responsible to train our model.
- y_train: Is the set of labels to all the data in x_train.
- x_test: This numpy array contains all the test data set.
- y_test: Is the set of labels to all the data in x_test.

```
x_train = train[:, :-1]
y_train = train[:, -1]
x_test = test[:, :-1]
y_test = test[:, -1]
```

The data visualization of train and test set showed in the following graph.



2.7: Building the model

For the ease of use, we set the Sequential() network with the word "model" and as a result we can build all the requirements and the modifications of the model on it.

```
#The LSTM model we will test, lets see if the sinus prediction results can be matched
#Here we start the model building structure
model = Sequential()
```

To specify the model requirements we just have to add them.

```
model.add(LSTM(50, kernel_initializer='normal', return_sequences=True,
input_shape=(feature_train.shape[1],1)))
model.add(Dropout(0.2))
model.add(LSTM(100, kernel_initializer='normal', return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(200, kernel_initializer='normal', return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1,kernel_initializer='normal', activation='linear'))
```

This part of code defines the Long Sort Term Memory (LSTM) network and the layers. According to keras documentation, to define a LSTM network we have to set the:

- Number of Units: The number of hidden units is a direct representation of the capacity of a neural network. By changing the number of units we can see how the training accuracy is affected. A big number can cause over-fitting.
- kernel_initializer: The neural network needs to start with some weights and then iteratively update them to better values. The term kernel_initializer is a fancy term for which statistical distribution or function to use for initializing the weights. In case of statistical distribution, the library will generate numbers from that statistical distribution and use as starting weights.
- return_sequences: By setting this parameter true the network will output the full sequence of hidden states [h1, h2, ..., hn]. If the value is false the network will only output the hidden state at the final time step.
- input_shape: Defines the shape of data that will be in the first layer-tensor of model. This layer is the only one we have to define in terms of data shape and is mandatory to have the same shape with the training data in order to have the right prediction.

Afterwards we add the layers and their dependences. As the source code shows we add two more LSTM layers in the model. The units we set in the LSTM layer are 100 and 200 respectively. Also the return sequences parameter is false in the third layer and the reason why is that we want to output the hidden state at the final time step.

The dropout parameter in keras libraries are very helpful in order to ensure that the model prediction will not overfit. The name of this parameter comes out from it’s operation to the model which is to drop out units of neural network. To be more specific, at each training stage individual nodes are either dropped out of the net with probability 1-p or kept with probability p. The p value is set in code by putting a numeric value between (0-1). This specific parameter has to set after each layer implementation. Our dropout value is 0.2 before every hidden LSTM layer.

```
        model.add(Dropout(0.2))
```

Last but not least, the output layer is defined with “dense” units equals to one. Dense layer is a linear operation in which every input is connected to every output by a weight.

```
        model.add(Dense(1,kernel_initializer='normal', activation='linear'))
```

The next step of the model development is to compile the model that is built in with the specifications that given previously.

```
model.compile(optimizer='adam',loss='mean_absolute_error',metrics=['mean_absolute_error'])
```

The dependencies that are set in the model compilation are:

- Loss: A scalar value that we attempt to minimize during our training of the model. The lower the loss, the closer our predictions are to the true labels. The 'mean_absolute_error' parameter measures the difference between two continuous variables.
- Mean Absolute Error: 'mean_absolute_error' is an average of the difference between the Original Values and the Predicted Values. It gives us the measurement of how far the predictions values were from the actual output.
- Optimizer: Is an algorithm that is set according to data, the computer power as well as the layers that are used in the model structure. Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. In addition this method is well suited for large data and it has no need of big memory requirements.
- Metrics: this parameter is related with the details that are shown during the fitting. The mean absolute error is going to be printed in each batch of epoch.

In order to train the model we use the “fit” method of model class API with specific inputs. In this thesis we use a limited number of arguments which is close related with the air particular matter data set.

```
#Train the model with Keras fit function
model.fit(feature_train, label_train, batch_size=300, epochs=5, validation_split = 0.2)
```

These arguments are counted as follows:

- X: This argument contains the training data. The “feature_train” array is the one that has the thesis’s data for training.
- y: This argument contains the label data from all the training data. The “label_train” array is the one that contains these data.
- batch_size: Is the number of samples that will train the network every time until the data set ends. This type of training requires less memory as well as the network train get faster. The batch size of the model implementation is 300.
- epochs: The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. The epoch runs in one level above the batch_size. By setting the epoch number equals to 5, the training set will run 5 times in order to teach the model.
- validation_split: This argument selects a percentage from training data and split them to new training data and validation data in order to validate them. In this thesis the percentage of the validation data is 20% of training data .

The prediction is one of the last parts of the model. It runs with the test data and imports the result to “predicted” array.

```
#Here we use the model and predict the PM2.5 of Shenyang
predicted = model.predict(feature_test)
```

The last part is to print from the ThLSTM.py file the true data and the predicted in order to compare them.

```
plot_results(predicted,label_test)
```

“plot_results” function called from the ThLSTM.py file and it executes the below code from the HelpingFunctions.py file.

```
def plot_results(predicted_data, true_data):
    fig = plt.figure(facecolor='white',figsize=(16,4))
    ax = fig.add_subplot(111)
    ax.plot(true_data, label='True Data')
    plt.plot(predicted_data,label='Prediction')
    plt.legend(['True Data', 'Prediction'])
    plt.show()
```

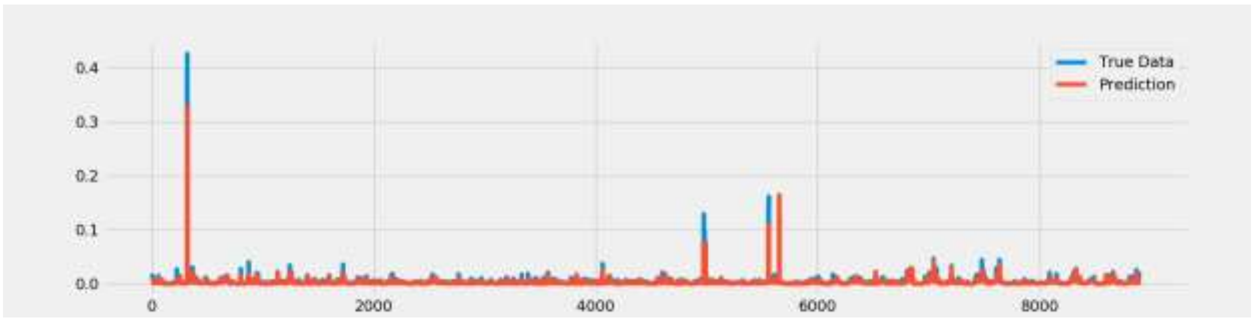
The two inputs of this function are transferred from the executable file. From this data, will be visualized the result of the model prediction. First the plot figure is set up. Is defined the background color and the size of the figure. Also, because matplotlib is related closely with the Matlab plots the plot location can be defined with the way below:

```
ax = fig.add_subplot(111)
```

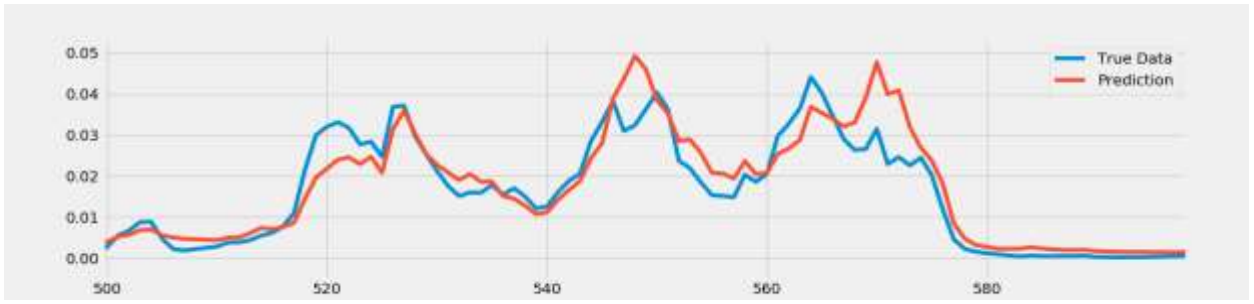
In addition, the next two lines of thesis model build the relation of data with the labels in order to make a description for them.

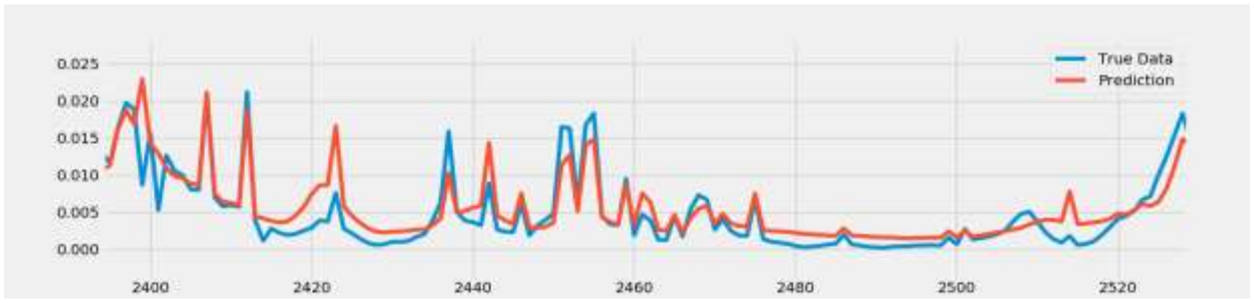
```
ax.plot(true_data, label='True Data')
plt.plot(predicted_data,label='Prediction')
```

Lastly, the data are printed in with the legend **True Data** and **Prediction** and the function to print the plot is called “show()”.



Here we visualize a closer view of the prediction and true data.





2.8: R squared (R^2) coefficient of determination regression score function

R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. The definition of R-squared is fairly straight-forward; it is the percentage of the response variable variation that is explained by a linear model. Or:

R-squared = Explained variation / Total variation

R-squared is always between 0 and 100%:

- 0% indicates that the model explains none of the variability of the response data around its mean.
- 100% indicates that the model explains all the variability of the response data around its mean.

In general, the higher the R-squared, the better the model fits your data. However, there are important conditions for this guideline that I'll talk about both in this post and my next post.

```
#Here we calculate the R^2 score of training kai testing set on our model
feature_train_pred = model.predict(feature_train)
print("The R2 score on the Train set is:\t{:0.3f}".format(r2_score(label_train, feature_train_pred)))
print("The R2 score on the Test set is:\t{:0.3f}".format(r2_score(label_test, predicted)))
```

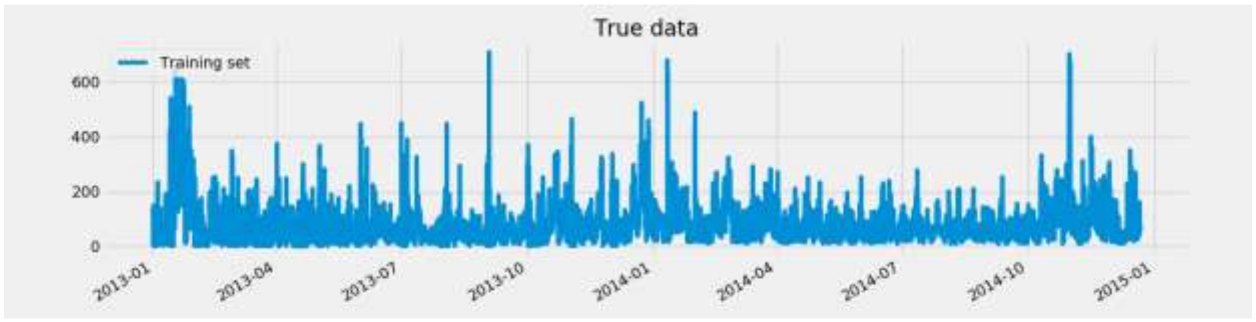
Chapter 3: Experimental results

This chapter is about to explore the results of the model and the whole process in terms of accuracy and to analyze the plots of data. Furthermore, it would relate the prediction with the city of Shenyang and her three measuring station in real life.

We run our LSTM model in two different computers:

- First system in CPU mode: Processor Intel i7-3610QM 2.3GHz, RAM 8GB
- Second system in GPU mode: Processor Intel i5-8250U 1.60GHz, RAM 8GB and GPU NVIDIA GeForce 940MX with GPU physical memory 2GB.

When we start our model, we print first the data as are in the CSV file. The data are particular matter (2.5) measurements.



And then we take those data and split them 66% training and 34% training and then we start the training and test of our model in two different computers.

3.1: CPU Mode

Time for training and testing our model in CPU mode on the second system takes **~12.34 minutes**

```
Data: 26179
Training Dataset: 17278
Testing Dataset: 8901
(17278, 100, 1)

model compiled
Train on 13822 samples, validate on 3456 samples

Epoch 1/5
2019-04-06 15:35:59.674432: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU
supports instructions that this TensorFlow binary was not compiled to use: AVX

13822/13822 [==============================] - 144s 10ms/step - loss: 0.0078 -
mean_absolute_error: 0.0078 - val_loss: 0.0026 - val_mean_absolute_error: 0.0026
```

```
Epoch 2/5
13822/13822 [==============================] - 124s 9ms/step - loss: 0.0063 -
mean_absolute_error: 0.0063 - val_loss: 0.0021 - val_mean_absolute_error: 0.0021

Epoch 3/5
13822/13822 [==============================] - 119s 9ms/step - loss: 0.0049 -
mean_absolute_error: 0.0049 - val_loss: 0.0020 - val_mean_absolute_error: 0.0020

Epoch 4/5
13822/13822 [==============================] - 126s 9ms/step - loss: 0.0050 -
mean_absolute_error: 0.0050 - val_loss: 0.0017 - val_mean_absolute_error: 0.0017

Epoch 5/5
13822/13822 [==============================] - 140s 10ms/step - loss: 0.0046 -
mean_absolute_error: 0.0046 - val_loss: 0.0016 - val_mean_absolute_error: 0.0016

8901/8901 [==============================] - 28s 3ms/step

Evaluation Test:  [0.0015303415877024901, 0.0015303415877024901]

The R2 score on the Train set is:      0.712
The R2 score on the Test set is:       0.738
```

3.2: GPU Mode

Time for training and testing our model in GPU mode on the second system takes **~8.16 minutes**

```
Data: 26179
Training Dataset: 17278
Testing Dataset: 8901
(17278, 100, 1)

model compiled
Instructions for updating:
Use tf.cast instead.
Train on 13822 samples, validate on 3456 samples

Epoch 1/5

2019-04-05 22:43:16.936091: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created
TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 1394 MB memory) -> physical
GPU (device: 0, name: GeForce 940MX, pci bus id: 0000:01:00.0, compute capability: 5.0)
```

```
2019-04-05 22:43:21.530578: I tensorflow/stream_executor/dso_loader.cc:152] successfully opened
CUDA library cublas64_100.dll locally

13822/13822 [==============================] - 83s 6ms/step - loss: 0.0077 -
mean_absolute_error: 0.0077 - val_loss: 0.0038 - val_mean_absolute_error: 0.0038

Epoch 2/5
13822/13822 [==============================] - 68s 5ms/step - loss: 0.0064 -
mean_absolute_error: 0.0064 - val_loss: 0.0036 - val_mean_absolute_error: 0.0036

Epoch 3/5
13822/13822 [==============================] - 64s 5ms/step - loss: 0.0050 -
mean_absolute_error: 0.0050 - val_loss: 0.0015 - val_mean_absolute_error: 0.0015

Epoch 4/5
13822/13822 [==============================] - 70s 5ms/step - loss: 0.0039 -
mean_absolute_error: 0.0039 - val_loss: 0.0017 - val_mean_absolute_error: 0.0017

Epoch 5/5
13822/13822 [==============================] - 68s 5ms/step - loss: 0.0038 -
mean_absolute_error: 0.0038 - val_loss: 0.0010 - val_mean_absolute_error: 0.0010

Evaluation Test
8901/8901 [==============================] - 137s 15ms/step

Evaluation Test: [0.0010050342879886407, 0.0010050342879886407]

The R2 score on the Train set is:     0.801
The R2 score on the Test set is:      0.952
```

3.3 CPU & GPU Comparison

By comparing the two implementations in both CPU and GPU systems useful results are coming up. The execution via GPU had faster and better result of the CPU.

On the first hand we have the GPU. GPU contains hundreds of simple cores as well as thousands of hardware threads. Due to multiple cores it maximizes the floating-point throughput and it has most die surface for integer and floating-point units.

On the other hand, we have the CPU. The CPU contains few very complex cores and single-thread performance optimization, transistors space dedicated to complex ILP and it has few die surfaces for integer and floating-point units.

What follows from these two statements is that the GPU provides more useful results. This comes out from building structure of GPU the model, the execution is faster and the R^2 measurement is clearly more efficient.

3.4: R^2 (coefficient of determination) regression score function Results

While the model executes the "fit" function from keras API it prints in every epoch the batch loss, mean_absolute_error, val_loss and val_mean_absolute_error. And after the model fit finish we start evaluation test and finally we calculate the R^2 (coefficient of determination) regression score function.

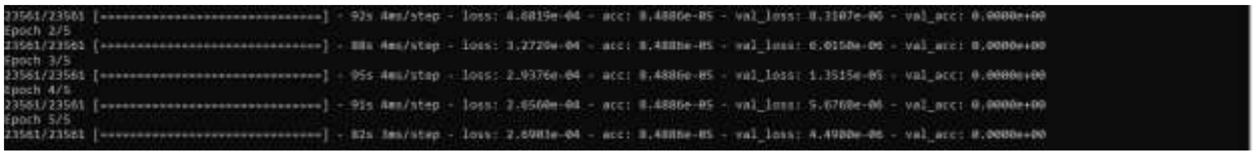
```
The R2 score on the Train set is:     0.801
The R2 score on the Test set is:      0.952
```

Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

3.5: Results overview

3.5.1: Machine Learning code overview (structure step by step. And the output)

During the machine learning model building we came across with multiple results and outputs of the model. To be begin with, the main issue we solve was overfit. In several tries the result of prediction data was a clear sign that the model overfitted. The strongest evidence that the model had this behavior was the little loss values during the validation of the model. Validation results run in every batch of the model and after running the model several times we were sure that we had to modify it. Another argument that we had overfit in the model was the value itself. To be more specific, the loss value in every batch validation had very small decimal values with the structure below:



```
23561/23561 [==============================] - 92s 4ms/step - loss: 4.8819e-04 - acc: 8.4886e-05 - val_loss: 8.3107e-06 - val_acc: 0.0000e+00
Epoch 2/5
23561/23561 [==============================] - 88s 4ms/step - loss: 3.2720e-04 - acc: 8.4886e-05 - val_loss: 6.0150e-06 - val_acc: 0.0000e+00
Epoch 3/5
23561/23561 [==============================] - 95s 4ms/step - loss: 2.9376e-04 - acc: 8.4886e-05 - val_loss: 1.3515e-05 - val_acc: 0.0000e+00
Epoch 4/5
23561/23561 [==============================] - 91s 4ms/step - loss: 2.6560e-04 - acc: 8.4886e-05 - val_loss: 5.6760e-06 - val_acc: 0.0000e+00
Epoch 5/5
23561/23561 [==============================] - 82s 3ms/step - loss: 2.6981e-04 - acc: 8.4886e-05 - val_loss: 4.4900e-06 - val_acc: 0.0000e+00
```

Afterwards, we had to concentrate in other main parts of the model structure, like plot the predicted data related with the true data as well as evaluating the model. A great way to realize if the model is accurate from a first view is to visualize and compare the true data with the predictive one. By taking a close (zoomed) view of the true-predicted data plot we distinguish that both time series are following the same norm as well as the predicted data follow the high and low picks of the true data. As a result, this means that we are have at first look that the model makes good predictions. The next step was to evaluate the model with R square value in order to be confident that the model works precisely in terms of accuracy. In general, the higher the R-squared, the better the model fits the data.

Chapter 4: Source Code

4.1:ThLSTM.py File

```
import sys
import numpy as np # linear algebra
from keras.layers import BatchNormalization
from numpy import newaxis
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM, GRU
from keras.models import Sequential
from keras import optimizers
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
from HelpingFunctions import * #we import our Helping Function file that contains all the functions
from sklearn.metrics import r2_score

#Here is the style that we use in our plots
plt.style.use('fivethirtyeight')
print ('import completed')

# Enter the steps to register on the network.
#RNN / LSTM / GRU can be taught at times times as large as the number of times you register them,
#and no longer (a fundamental limitation).
# So by design these networks are deep / long to catch repeating motifs.
Enrol_window = 100

print ('enrol window set to',Enrol_window )
print('Support functions defined')

#Load the data
dataset = pd.read_csv('./Data/ShenyangECXELPM20100101_20151231---V2).csv',sep=";")

print(dataset.head())

#Create the list which contains the date data
dat=pd.to_datetime(dataset["date"])
date =np.array(dat)

#Here we create three arrays with all PM station measurement of Shenyang and covert them to lists
#so then we can edit them one by one
one=np.array(dataset["PM_Taiyuanjie"])
one1 = one.tolist()
```

```
two= np.array(dataset["PM_US Post"])
two1 = two.tolist()
three = np.array(dataset["PM_Xiaoheyan"])
three1 = three.tolist()


#In order to calculate the average value of Shenyang we take the inputs of each station that have non
#zero values
#and divide them with the sum of the station that has non zero values
PM_Shenyang = []
for i in range(len(one1)):
   #Here we calculate the average if one station has 0 measurment
   if (one1[i] == 0 and two1[i] != 0 and three1[i] != 0) or (one1[i] != 0 and two1[i] == 0 and three1[i] !=
0) or ( one1[i] != 0 and two1[i] != 0 and three1[i] == 0):
      PM_Shenyang.append((one1[i] + two1[i] + three1[i])/2)

   #Here we calculate the average if there is no zero values in station
   elif (one1[i] != 0 and two1[i] != 0 and three1[i] != 0):
      PM_Shenyang.append((one1[i] + two1[i] + three1[i]) / 3)

   #Here we calculate the average if 2 of the station has zero values
   elif (one1[i] == 0 and two1[i] == 0 and three1[i] != 0) or (one1[i] == 0 and two1[i] != 0 and
          three1[i]== 0) or (one1[i] != 0 and two1[i] == 0 and three1[i] == 0):
       PM_Shenyang.append(one1[i] + two1[i] + three1[i])

   #Here we calulate the average if 3 of the station has zero values
   elif (one1[i] == 0 and two1[i] == 0 and three1[i] == 0) or (one1[i] == 0 and two1[i] == 0 and three1[i]
         == 0) or (one1[i] == 0 and two1[i] == 0 and three1[i] == 0):
      PM_Shenyang.append(0)

PM_Shenyang1 = np.array(PM_Shenyang)

#In order to overtake the error values we replace all the values over 750 with the number 400
PM_Shenyang1[PM_Shenyang1 > 750] = 400
print("PMnumbers ", PM_Shenyang)

#This is the finale dataframe that contains all the values for the prediction
newPMdf = pd.DataFrame(data = {'PM_Shenyang':PM_Shenyang1},index = date)
print(newPMdf)

# Prepare the dataset, note that the PM 2.5 values for PM_Shenyang data will be normalized between
0 and 1
# A feature is an input variable, in this case a PM 2.5 Value
# Selected 'PM_Shenyang' PM 2.5 Values
feature_train, label_train, feature_test, label_test, train_size, X = data_loader(newPMdf,
'PM_Shenyang', Enrol_window, True)
newPMdf["PM_Shenyang"][0:train_size].plot(figsize=(16,4),legend=True)
```

```
newPMdf["PM_Shenyang"][train_size:len(X)].plot(figsize=(16,4),legend=True) # 10% is used for
thraining data which is approx 2017 data
plt.legend(['Training dataset','Testing dataset'])
plt.title('True data')
plt.show()

#The LSTM model we will test, let’s see if the sinus prediction results can be matched
#Here we start the model building structure
model = Sequential()
model.add(LSTM(50, kernel_initializer='normal', return_sequences=True,
input_shape=(feature_train.shape[1],1)))
model.add(Dropout(0.2))
model.add(LSTM(100, kernel_initializer='normal', return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(200, kernel_initializer='normal', return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1,kernel_initializer='normal', activation='linear'))

model.compile(optimizer='adam', loss='mean_absolute_error', metrics=['mean_absolute_error'])

print('model compiled')

#Train the model with Keras fit function
model.fit(feature_train, label_train, batch_size=300, epochs=5, validation_split = 0.2)

#Here we use the model and predict the PM2.5 of Shenyang
predicted = model.predict(feature_test)
plot_results(predicted,label_test)

#Here we do the evaluation test
kappa = model.evaluate(feature_test,label_test)

print('Evaluation Test: ', kappa)

#Here we calculate the R^2 score of training kai testing set on our model
feature_train_pred = model.predict(feature_train)
print("The R2 score on the Train set is:\t{:0.3f}".format(r2_score(label_train, feature_train_pred)))
print("The R2 score on the Test set is:\t{:0.3f}".format(r2_score(label_test, predicted)))
```

4.2: HelpingFunctions File

```
import numpy as np # linear algebra
from numpy import newaxis
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM, GRU
from keras.models import Sequential
from sklearn.model_selection import train_test_split
from keras import optimizers

from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt

sc = MinMaxScaler(feature_range=(0, 1))

#it BRINGS: 1: The data, 2: Column Name to work with, 3: Just a number of Window (100), 4:Boolean
value
def data_loader(datasetname, column, seq_len, normalise_window):
    #A support function for preparing data sets for an LSTM network
    #Take the dataframe and get all the data from the column we want to work with and replace them
    # from 0 to 1 in order to make the division
    data = datasetname.loc[:, column].replace(0, 1)
    sequence_length = seq_len + 1 # make  length value greater than 1

    result = []
    for index in range(len(data) - sequence_length):
        result.append(data[index: index + sequence_length])

    if normalise_window:
        result = normalise_windows(result)
        result = sc.fit_transform(result)

    result = np.array(result)
    X = result
    train_size = int(len(X) * 0.66)
    train, test= X[0:train_size], X[train_size:len(X)]

    print('Data: %d' % (len(X)))
    print('Training Dataset: %d' % (len(train)))
    print('Testing Dataset: %d' % (len(test)))
    #print('Test Validation: %d' % (len(val)))

    x_train = train[:, :-1]
    y_train = train[:, -1]
    x_test = test[:, :-1]
    y_test = test[:, -1]

    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
    print(x_train.shape)
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
    return [x_train, y_train, x_test, y_test, train_size, X]

def normalise_windows(window_data):
    # Prepare the dataset, note that the PM  price data will be normalized between 0 and 1
    normalised_data = []
    for window in window_data:
        normalised_window = [((float(p) / float(window[0])) - 1) for p in window]
        normalised_data.append(normalised_window)
    return normalised_data

def plot_results(predicted_data, true_data):
    # Standard plot
    fig = plt.figure(facecolor='white',figsize=(16,4))
    ax = fig.add_subplot(111)
    ax.plot(true_data, label='True Data')
    plt.plot(predicted_data,label='Prediction')
    plt.legend(['True Data', 'Prediction'])
    plt.show()

print('Support functions defined')
```

Chapter 5: Conclusion

In the final analysis, machine learning practices can be worked in many cases of high importance with a great accuracy and method. Due to the need to know time related phenomenon, time series forecast which is used in this thesis, is an enormous achievement for machine learning history. The main topics that are noted down during the paper are about the history of machine learning and how a python model can be structured. All the tools are described extensively as well as the core operations of the code. Furthermore, is justified why many common practices like LSTM networks and keras API are applied for thesis. Moreover, the last part contains the results of the training and predictive model. Data visualization is a vital part of analysis as well as how accurate can the model be. Last but not least, this paper provides an overview to the best practices of time series analysis related to keras API library that can be researched further. The goal that is achieved in this thesis is to provide the core information about building and understanding the structure of a time series predictive model.

Chapter 6: Sources

«Apiumhub.» *DEEP LEARNING STARTUPS, USE CASES & BOOKS.* s.d. https://apiumhub.com/tech-blog-barcelona/deep-learning-startups/.

Bogdnov, Vik. *Python Application Development: Common Use Cases and Project Examples.* 13 4 2017. https://intersog.com/blog/python-application-development-common-use-cases-and-project-examples/.

Brandenburg, Justin. *Applying Deep Learning to Time Series Forecasting with TensorFlow.* s.d. https://mapr.com/blog/deep-learning-tensorflow/.

Brownlee, Jason. *Making Predictions with Sequences.* 4 9 2017. https://machinelearningmastery.com/sequence-prediction/.

—. *What is Deep Learning?* 16 8 2016. https://machinelearningmastery.com/what-is-deep-learning/.

—. *What Is Time Series Forecasting?* 2 12 2016. https://machinelearningmastery.com/time-series-forecasting/.

Budhiraja, Amar. *Dropout in (Deep) Machine learning.* 15 12 2016. https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5.

«Computer Hope.» *Interpreted.* 8 8 2017. https://www.computerhope.com/jargon/i/interpre.htm.

COPELAND, MICHAEL. «nvidia.» *What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?* 29 7 2016. https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/.

Diederik Kingma, Jimmy Ba. *Cornell University.* 30 1 2017. https://arxiv.org/abs/1412.6980v8.

Documentation, Jupiter Notebook. *What is the Jupyter Notebook.* 2015.

Editor, Minitab Blog. *The Minitab Blog.* 30 5 2013. https://blog.minitab.com/blog/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit.

«Exast.» *Top five use cases of Tensorflow.* 3 2 2017. https://www.exastax.com/deep-learning/top-five-use-cases-of-tensorflow/.

Fumo, David. *Medium.* 29 1 2017. https://medium.com/simple-ai/pandas-library-in-a-nutshell-intro-to-machine-learning-3-acbd39ec5c9c.

IBM, Coursera. *Applied AI with DeepLearning.* s.d. https://www.coursera.org/lecture/ai/sequential-models-in-keras-RBbLP.

Keras Backend. s.d. https://cran.rstudio.com/web/packages/keras/vignettes/backend.html.

Keras Documentation RNN. s.d. https://keras.io/layers/recurrent/.

Keras, Documentation. *Keras: The Python Deep Learning library.* s.d. https://keras.io/.

Leung, Ellery. *RNN/LSTM Example With Keras — About input shape.* 1 9 2017. https://medium.com/@ellery.leung/rnn-lstm-example-with-keras-about-input-shape-94120b0050e.

Machine Learning Repository UCI . 18 07 2017. https://archive.ics.uci.edu/ml/datasets/PM2.5+Data+of+Five+Chinese+Cities.

«NCS Statistical Software.» *Ridge Regression.* s.d. https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf.

Nguyen, Michael. *Illustrated Guide to LSTM's and GRU's: A step by step explanation.* 24 9 2018. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

Oliphant, Travis. *Wikipedia.* 17 3 2019. https://en.wikipedia.org/wiki/NumPy.

pandas, software. *Wikipedia.* 28 3 2019. https://en.wikipedia.org/wiki/Pandas_(software).

Parrish, Kevin. «Digital Trends.» *Deep learning vs machine learning.* s.d. https://www.digitaltrends.com/cool-tech/deep-learning-vs-machine-learning-explained/2/.

«Python.» *Python Executive Summary.* s.d. https://www.python.org/doc/essays/blurb/.

Read, Gary. «import.io.» *5 Industries Machine Learning is Disrupting.* 13 3 2017. https://www.import.io/post/5-industries-machine-learning-is-disrupting/.

«SAS Insights.» *Analytics Insights.* s.d. https://www.sas.com/en_us/insights/analytics/machine-learning.html.

Scipy, community. *ScyPy.org.* 10 6 2017. https://docs.scipy.org/doc/numpy-1.13.0/user/whatisnumpy.html.

stackoverflow. *Keras input explanation: input_shape, units, batch_size, dim, etc.* 2017. https://stackoverflow.com/questions/44747343/keras-input-explanation-input-shape-units-batch-size-dim-etc.

Tech Sparks. s.d. https://www.techsparks.co.in/hot-topic-for-project-and-thesis-machine-learning/.

Understanding LSTM Networks. 27 8 2015. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

«United States Environmental Protection Agency.» *Health and Environmental Effects of Particulate Matter (PM).* 20 6 2018. https://www.epa.gov/pm-pollution/health-and-environmental-effects-particulate-matter-pm.

«Unted States Environmental Protection Agency.» *Particulate Matter (PM) Pollution.* 10 04 2017. https://www3.epa.gov/region1/airquality/pm-what-is.html.

Vashishta, Vin. «Linkedin.» *Deep Learning Use Cases: What can deep learning do for businesses?* s.d. https://www.linkedin.com/pulse/deep-learning-use-cases-what-can-do-businesses-vin-vashishta/.

Wesleyan, University. *Coursera.* 2019. https://www.coursera.org/lecture/machine-learning-data-analysis/what-is-lasso-regression-0KIy7.

«Wiki, A.I.» *Deep Learning, Machine Learning & AI Use Cases.* s.d. https://skymind.ai/wiki/use-cases.

Wikipedia. *Anaconda (Python distribution).* 27 2 2019. https://en.wikipedia.org/wiki/Anaconda_(Python_distribution).

—. *Deep learning.* 31 3 2019. https://en.wikipedia.org/wiki/Deep_learning.

—. *Guido van Rossum.* 9 3 2019. https://en.wikipedia.org/wiki/Guido_van_Rossum.

—. *Interpreted language.* 9 3 2019. https://en.wikipedia.org/wiki/Interpreted_language.

—. *Interpreter (computing).* 7 3 2019. https://en.wikipedia.org/wiki/Interpreter_(computing).

—. *Keras.* 12 3 2019. https://en.wikipedia.org/wiki/Keras.

—. *Lasso (statistics).* 28 3 2019. https://en.wikipedia.org/wiki/Lasso_(statistics).

—. *Linear regression.* 26 3 2019. https://en.wikipedia.org/wiki/Linear_regression.

—. *Logistic regression.* 24 3 2019. https://en.wikipedia.org/wiki/Logistic_regression.

—. *Long short-term memory.* 30 3 2019. https://en.wikipedia.org/wiki/Long_short-term_memory.

—. *Machine learning.* 29 3 2019. https://en.wikipedia.org/wiki/Machine_learning.

—. *Matplotlib.* 26 2 2019. https://en.wikipedia.org/wiki/Matplotlib.

—. *Mean Squared Error.* 27 3 2019. https://en.wikipedia.org/wiki/Mean_squared_error.

—. *Particulates.* 22 3 2012. https://en.wikipedia.org/wiki/Particulates.

—. *Perceptron.* 19 3 2019. https://en.wikipedia.org/wiki/Perceptron.

—. *Python (programming language).* 29 3 2019. https://en.wikipedia.org/wiki/Python_(programming_language).

—. *TensorFlow.* 29 3 2019. https://en.wikipedia.org/wiki/TensorFlow.

—. *Tikhonov regularization.* 28 3 2019. https://en.wikipedia.org/wiki/Tikhonov_regularization.

—. *Time Series.* 13 3 2019. https://en.wikipedia.org/wiki/Time_series.