

REAL -TIME PREDICTION OF TAXI DEMAND USING NEURAL NETWORK

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Science degree in Computer Science

By

Mayuresh (Reg.No : 38290045)

Akash (Reg.No : 38290004)



**DEPARTMET OF COMPUTER SCIENCE
SCHOOL OF COMPUTING**

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI-600119

APRIL - 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC
JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI
www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of A.BHUVANASREE (37290006), Y.N DEEPIKA (37290008), who carried out the project entitled "REAL- TIME PREDICTION OF TAXI DEMAND USING RECURRENT NEURAL NETWORK" under our supervision from November 2019 to April 2020.

Internal Guide

Mrs .J.REFONAA, M.E.,(Ph.D)

Head of the Department

(Dr. L. LAKSHMANAN, M.E., Ph.D.,)

Submitted for Viva voce Examination held on

Internal Examiner

External Examiner

DECLARATION

I, Mayuresh (Reg.No : 38290045) and Akash (Reg.No : 38290004) hereby declare that the Project Report entitled “REAL TIME PREDICTION OF TAXI DEMAND USING RECURRENT NEURAL NETWORK” done by me under the guidance of Mrs.J.Refonaa M.E.,(Ph.D.), is submitted in partial fulfilment of the requirements for the award of Bachelor of Science degree in Computer Science.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. SASIKALA, M.E., PH.D., Dean**, School of Computing **Dr.S. Vigneshwari M.E., Ph.D** and **Dr. L. Lakshmanan M.E., Ph.D., Heads of the Department**, for providing me necessary Support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Mrs.J.Refonnaa M.E.,(Ph.D)**, for her valuable guidance, suggestions and constant encouragement Paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science** who were helpful in many ways for the completion of the project.

ABSTRACT

Predicting taxi demand throughout a city can help to organize the taxi fleet and minimize the wait-time for passengers and drivers. In this paper, we propose a sequence learning model that can predict future taxi requests in each area of a city based on the recent demand and other relevant information. Remembering information from the past is critical here, since taxi requests in the future are correlated with information about actions that happened in the past. For example, someone who requests a taxi to a shopping center, may also request a taxi to return home after few hours. We use one of the best sequence learning methods, long short term memory that has a gating mechanism to store the relevant information for future use. We evaluate our method on a data set of taxi requests in New York City by dividing the city into small areas and predicting the demand in each area. We show that this approach outperforms other prediction methods, such as feed-forward neural networks. In addition, we show how adding other relevant information, such as weather, time, and drop-offs affects the results.

TABLE OF CONTENTS

*

CHAPTER No	TITLE	PAGE NO.
	ABSTRACT	I
	LIST OF FIGURES	II
	LIST OF ABBREVIATIONS	III
1	INTRODUCTION	1
	1.1 Aim	1
	1.2 Synopsis	1
	1.3 Hardware Requirement	1
	1.4 Software Requirement	1
2	LITERATURE SURVEY	2
3	METHODOLOGY	9
	3.1 Existing system	9
	3.2 Proposed System	9
	3.3 Problem Statement	9
	3.4 Requirement Specification	

Hardware & Software Specification	10
Introduction of Python	11
Machine Learning	12
Design & Implementation constraint	16
Other non-function requirement	30
4 SYSTEM DESIGN	31
Architecture Diagram	33
Sequence Diagram	33
Use case Diagram	34
Activity Diagram	35
Collaboration Diagram	35
Data flow Diagram	37
5 MODULES & FUTURE WORK	38
Modules	39
Modules Explanation	39
Coding & Testing	40
Naming Convention	41
Test procedure system testing	42
Test Data & Output	43
Testing Techniques	45

SOURCE CODE	50
SCREENSHOT	70
CONCLUSION	73
REFERENCE	74

LIST OF FIGURES:

FIGURE NO	FIGURE NAME
4.1	Architecture Diagram
4.2	Sequence Diagram
4.3	Use case Diagram
4.4	Activity Diagram
4.5	Colloboration Diagram
4.6	Data flow Diagram

LIST OF ABBREVIATIONS:

RNN – Recurrent Neural Network

GPS - Global Positioning System

LSTM – Long Short Term Memory

MDN - Mixture Density Networks (MDN)

CHAPTER 1

INTRODUCTION

Aim:

To predict the high demand need of pickup location for taxi services based on their previous history.

Synopsis:

The service industry is booming for the last couple of years and it is expected to grow in the near future. One of the important natures of the business is the serve the customer. To effectively utilize the resource at hand is the key factor. Businesses are using advanced technology to achieve this.

Software Requirement:

- Operating System : Windows 7 and above (64-bit).
- Python : 3.6

1.4 Hardware Requirement:

- Hard disk : 500 GB and above.
- Processor : i3 and above.
- Ram : 4GB and above

CHAPTER 2

LITERATURE SURVEY:

Project Title: A Collaborative Multi agent Taxi-Dispatch System

Author Name: Kiam Tian Seow, Nam Hai Dang and Der-Horng Lee

Year of Publishing: 2014

Abstract:

This paper presents a novel multi agent approach to automating taxi dispatch that services current bookings in a distributed fashion. The existing system in use by a taxi operator in Singapore and elsewhere attempts to increase customer satisfaction locally, by sequentially dispatching nearby taxis to service customers. The proposed dispatch system attempts to increase customer satisfaction more globally, by concurrently dispatching multiple taxis to the same number of customers in the same geographical region, and vis-a-vis human driver satisfaction. To realize the system, multi agent architecture is proposed, populated with software collaborative agents that can actively negotiate on behalf of taxi drivers in groups of size N for available customer bookings. Theoretically, an analysis of the boundary and optimal multi agent taxi-dispatch situations is presented along with a discussion of their implications. Experimentally, the operational efficiency of the existing and proposed dispatch systems was evaluated through computer simulations. The empirical results, obtained for a 1000-strong taxi fleet over a discrete range of N , show that the proposed system can dispatch taxis with reduction in customer waiting and empty taxi cruising times of up to 33.1% and 26.3%, respectively; and up to 41.8% and 41.2% reduction when a simple negotiation speedup heuristic was applied.

Project Title: A Multi-Level Clustering Approach for Forecasting Taxi Travel Demand

Author Name: Neema Davis, Gaurav Raina and Krishna Jagannathan

Year of Publishing: 2016

Abstract:

In this paper, we use time-series modeling to forecast taxi travel demand, in the context of a mobile application based taxi hailing service. In particular, we model the passenger demand density at various locations in the city of Bengaluru, India. Using the data, we first shortlist time-series models that suit our application. We then analyze the performance of these models by using Mean Absolute Percentage Error (MAPE) as the performance metric. In order to improve the model performance, we employ a multi-level clustering technique where we aggregate demand over neighboring cells/geohashes. We observe that the improved model based on clustering leads to a forecast accuracy of 80% per km². In addition, our technique obtains an accuracy of 89% per km² for the most frequently occurring use case.

Project Title: Predicting Taxi–Passenger Demand Using Streaming Data

Author Name: Luis Moreira-Matias, Gama, Michel Ferreira, João Mendes-Moreira, Luis Damas

Year of Publishing: 2014

Abstract:

Informed driving is increasingly becoming a key feature for increasing the sustainability of taxi companies. The sensors that are installed in each vehicle are providing new opportunities for automatically discovering knowledge, which, in return, delivers information for real-time decision making. Intelligent transportation systems for taxi dispatching and for finding time-saving routes

are already exploring these sensing data. This paper introduces a novel methodology for predicting the spatial distribution of taxi-passengers for a short-term time horizon using streaming data. First, the information was aggregated into a histogram time series. Then, three time-series forecasting techniques were combined to originate a prediction. Experimental tests were conducted using the online data that are transmitted by 441 vehicles of a fleet running in the city of Porto, Portugal. The results demonstrated that the proposed framework can provide effective insight into the spatiotemporal distribution of taxi-passenger demand for a 30-min horizon.

Project Title: Quantifying the benefits of vehicle pooling with shareability networks

Author Name: Paolo Santi, Giovanni Resta, Michael Szell, Stanislav Sobolevsky, Steven Strogatz, Carlo Ratti

Year of Publishing: 2014

Abstract:

Taxi services are a vital part of urban transportation, and a considerable contributor to traffic congestion and air pollution causing substantial adverse effects on human health. Sharing taxi trips is a possible way of reducing the negative impact of taxi services on cities, but this comes at the expense of passenger discomfort quantifiable in terms of a longer travel time. Due to computational challenges, taxi sharing has traditionally been approached on small scales, such as within airport perimeters, or with dynamical ad-hoc heuristics. However, a mathematical framework for the systematic understanding of the tradeoff between collective benefits of sharing and individual passenger discomfort is lacking. Here we introduce the notion of share ability network which allows us to model the collective benefits of sharing as a function of passenger inconvenience, and to efficiently compute optimal sharing strategies on massive datasets. We apply this framework to a dataset of millions of taxi trips taken in New York City, showing that with increasing but still relatively low passenger discomfort; cumulative trip length can be cut by 40% or more. This benefit comes with reductions in service cost, emissions, and with split fares,

hinting towards a wide passenger acceptance of such a shared service. Simulation of a realistic online system demonstrates the feasibility of a shareable taxi service in New York City. Shareability as a function of trip density saturates fast, suggesting effectiveness of the taxi sharing system also in cities with much sparser taxi fleets or when willingness to share is low.

Project Title: Sequence to Sequence Learning with Neural Networks

Author Name: Ilya Sutskever, Quoc V. Le

Abstract:

Deep Neural Networks (DNNs) are powerful models that have achieved excellent performance on difficult learning tasks. Although DNNs work well whenever large labeled training sets are available, they cannot be used to map sequences to sequences. In this paper, we present a general end-to-end approach to sequence learning that makes minimal assumptions on the sequence structure. Our method uses a multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. Our main result is that on an English to french translation task from the WMT-14 dataset, the translations produced by the LSTM achieve a BLEU score of 34.8 on the entire test set, where the LSTM's BLEU score was penalized on out-of-vocabulary words. Additionally, the LSTM did not have difficulty on long sentences. For comparison, a phrase-based SMT system achieves a BLEU score of 33.3 on the same dataset. When we used the LSTM to rerank the 1000 hypotheses produced by the aforementioned SMT system, its BLEU score increases to 36.5, which is close to the previous state of the art. The LSTM also learned sensible phrase and sentence representations that are sensitive to word order and are relatively invariant to the active and the passive voice. Finally, we found that reversing the order of the words in all source sentences (but not target sentences) improved the LSTM's performance markedly, because doing so introduced many short term dependencies between the source and the target sentence which made the optimization problem easier.

Project Title: SPEECH RECOGNITION WITH DEEP RECURRENT NEURAL NETWORKS

Author Name: Alex Graves, Abdel-rahman Mohamed and Geoffrey Hinton

Abstract:

Recurrent neural networks (RNNs) are a powerful model for sequential data. End-to-end training methods such as Connectionist Temporal Classification make it possible to train RNNs for sequence labeling problems where the input-output alignment is unknown. The combination of these methods with the Long Short-term Memory RNN architecture has proved particularly fruitful, delivering state-of-the-art results in cursive handwriting recognition. However RNN performance in speech recognition has so far been disappointing; with better results returned by deep feed forward networks. This paper investigates deep recurrent neural networks, which combine the multiple levels of representation that have proved so effective in deep networks with the flexible use of long range context that empowers RNNs. When trained end-to-end with suitable regularization, we find that deep Long Short-term Memory RNNs achieve a test set error of 17.7% on the TIMIT phoneme recognition benchmark, which to our knowledge is the best recorded score.

Project Title: Taxi Dispatch with Real-Time Sensing Data in Metropolitan Areas a Receding Horizon Control Approach

Author Name: Fei Miao, Shan Lin, Sirajum Munir, John A. Stankovic, Hua Huang, Desheng Zhang, Tian He, George J. Pappas

Abstract:

Traditional transportation systems in metropolitan areas often suffer from inefficiencies due to uncoordinated actions as system capacity and traffic demand change. With the pervasive deployment of networked sensors in modern vehicles, large amounts of information regarding traffic demand and system status can be collected in real-time. This information provides opportunities to perform various types of control and coordination for large scale intelligent

transportation systems. In this paper, we present a novel receding horizon control (RHC) framework to dispatch taxis, which combines highly spatiotemporally correlated demand/supply models and real-time GPS location and occupancy information. The objectives include reducing taxi idle driving distance and matching spatiotemporal ratio between demand and supply for service quality. Moreover, our RHC framework is compatible with different predictive models and optimization problem formulations. This compatibility property allows us to model disruptive passenger demands and traffic conditions into a robust optimization problem. Extensive trace driven analysis with a real taxi data set from San Francisco shows that our solution reduces the average total idle distance by 52%, and reduces the total supply demand ratio error across the city by up to 45%.

Project Title: Taxi-Passenger-Demand Modeling Based on Big Data from a Roving Sensor Network

Author Name: Desheng Zhang, Tian He, Shan Lin, Sirajum Munir, John A. Stankovic

Abstract:

Investigating passenger demand is essential for the taxicab business. Existing solutions are typically based on offline data collected by manual investigations, which are often dated and inaccurate for real-time analysis. To address this issue, we propose D model, employing roving taxicabs as real-time mobile sensors to (i) infer passenger arriving moments by interactions of vacant taxicabs, and then (ii) infer passenger demand by customized online training with both historical and real-time data. D model utilizes a novel parameter called pickup pattern based on an entropy of pickup events (accounts for various real-world logical information, e.g., bad weather) to reduce the size of big historical taxicab data to be processed. We evaluate D model with a real-world 450 GB dataset of 14, 000 taxicabs for a half year, and results show that compared to the ground truth, D model achieves 83% accuracy and outperforms a statistical model by 42%. We further present an application where D model is used to dispatch vacant taxicabs to achieve equilibrium between passenger demand and taxicab supply across urban regions.

Project Title: The Neural Autoregressive Distribution Estimator

Author Name: Hugo Larochelle Iain Murray

Year of Publishing: 2011

Abstract:

We describe a new approach for modeling the distribution of high-dimensional vectors of discrete variables. This model is inspired by the restricted Boltzmann machine (RBM), which has been shown to be a powerful model of such distributions. However, an RBM typically does not provide a tractable distribution estimator, since evaluating the probability it assigns to some given observation requires the computation of the so-called partition function, which itself is intractable for RBMs of even moderate size. Our model circumvents this difficulty by decomposing the joint distribution of observations into tractable conditional distributions and modeling each conditional using a non-linear function similar to a conditional of an RBM. Our model can also be interpreted as an auto encoder wired such that its output can be used to assign valid probabilities to observations. We show that this new model outperforms other multivariate binary distribution estimators on several datasets and performs similarly to a large (but intractable) RBM.

CHAPTER 3

METHODOLOGY

EXISTING SYSTEM (recurrent neural network)

TAXI drivers need to decide where to wait for passengers in order to pick up someone as soon as possible. Passengers also prefer to quickly find a taxi whenever they are ready for pickup. The control center of the taxi service decides the busy area to be concentrated. Sometimes the taxi were scattered across the larger area missing the time based busy area like Airport, Business area, school area, Train stations etc.,.

Problem Statement:

- Managing fleet of taxi to crowded area.
- sEffective utilization of resources to reduce waiting time for passengers.
- Serve more customers in short time by organizing the availability of taxi.

Proposed System:

Effective taxi dispatching can help both drivers and passengers to minimize the wait-time to find each other. Drivers do not have enough information about where passengers and other taxis are and intend to go. Therefore, a taxi center can organize the taxi fleet and efficiently distribute them according to the demand from the entire city. To build such a taxi center, an intelligent system that can predict the future demand throughout the city is required. Our system uses GPS location and other properties of the taxi like drop point, pickup point etc. to predict the future demand. An Recurrent Neural Networks (RNN) based model is trained with given history data. This model is used to predict the demand in different areas of the city.

REQUIREMENT SPECIFICATIONS :

AXI drivers need to decide where to wait for passengers in order to pick up someone as soon as possible. Passengers also prefer to quickly find a taxi whenever they are ready for pickup. Effective taxi dispatching can help both drivers and passengers to minimize the wait-time to find each other. Drivers do not have enough information about where passengers and other taxis are and intend to go. Therefore, a taxi center can organize the taxi fleet and efficiently distribute them according to the demand from the entire city [1], [2]. This taxi center is especially needed in the future where self-driving taxis need to decide where to wait and pick up passengers. To build such a taxi center, an intelligent system that can predict the future demand throughout the city is required. Predicting taxi demand is challenging because it is correlated with many pieces of underlying information. One of the most relevant sources of information is historical taxi trips.

Thanks to the Global Positioning System (GPS) technology, taxi trip information can be collected from GPS enabled taxis [3], [4]. Fig. 1 shows an example of taxi request patterns in two areas. Analyzing this data shows that there are repetitive patterns in the data that can help to predict the demand in a particular area at a specific time. Several previous studies have shown that it is possible to learn from past taxi data [5]–[8]. In this paper, we propose a real-time method for predicting taxi demands in different areas of a city. We divide a big city into smaller areas and aggregate the number of taxi requests in each area during a small time period (e.g. 20 minutes). In this way, past taxi data becomes a data sequence of the number of taxi requests in each area. Then, we train a Long Short Term Memory (LSTM) [9] recurrent neural network (RNN) with this sequential data. The network input is the current taxi demand and other relevant information while the output is the demand in the next time-step. The reason we use a LSTM recurrent neural network is that it can be trained to store all the relevant information in a sequence to predict particular outcomes in the future. In addition, taxi demand prediction is a time series forecasting problem in which an intelligent sequence analysis model is required. LSTMs are the state of the art sequence learning models that are widely used in many

applications such as unsegmented handwriting generation [10] and natural language processing [11]. LSTM is capable of learning long term dependencies by utilizing some gating mechanisms to store information. Therefore, it can for instance remember how many people have requested taxis to attend a concert and after a couple of hours use this information to predict that the same number of people will request taxis from the concert location to different areas. However, predicting real-valued numbers is tricky because many times simply learning the average of the values in the dataset does not give a valid solution. It will also confuse

LSTM in the next time-step since the network has not seen the average before. Therefore, we add Mixture Density Networks (MDN) [12] on top of LSTM. In this way, instead of direct predicting a demand value, we output a mixture distribution of the demand. A sample can be drawn from this probability distribution and be treated as the predicted taxi demand. The remainder of this paper is organized as follows. Section II introduces related works on prediction applications using past taxi data and sequential learning applications of LSTMs. Section III shows how we encode the huge number of GPS records and a brief explanation of recurrent neural networks. Section IV describes the proposed sequence learning model, as well as the training and testing procedures.

In Section V, we show the performance metrics and present the experiment results.

Lastly, in Section VI we conclude the paper.

HARDWARE AND SOFTWARE SPECIFICATION

3.5.1HARDWARE REQUIREMENT

- | | |
|-------------|--------------------|
| Hard Disk | :500 GB and above. |
| • Processor | : i3 and above. |
| • Ram | : 4GB and above |

SOFTWARE REQUIREMENTS

- Operating System : Windows 7 and above (64-bit).
- Python : 3.6

TECHNOLOGIES USED:

RNN – Recurrent Neural Network

Introduction to Python

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- System scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

Good to know

- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
- Python 2.0 was released in 2000, and the 2.x versions were the prevalent releases until December 2008. At that time, the development team made the decision to release version 3.0, which contained a few relatively small but significant changes that were not backward compatible with the 2.x versions. Python 2 and 3 are very

similar, and some features of Python 3 have been backported to Python 2. But in general, they remain not quite compatible.

- Both Python 2 and 3 have continued to be maintained and developed, with periodic release updates for both. As of this writing, the most recent versions available are 2.7.15 and 3.6.5. However, an official End of Life date of January 1, 2020 has been established for Python 2, after which time it will no longer be maintained.
- Python is still maintained by a core development team at the Institute, and Guido is still in charge, having been given the title of BDFL (Benevolent Dictator for Life) by the Python community. The name Python, by the way, derives not from the snake, but from the British comedy troupe Monty Python's Flying Circus, of which Guido was, and presumably still is, a fan. It is common to find references to Monty Python sketches and movies scattered throughout the Python documentation.
- It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

Python Syntax compared to other programming languages

- Python was designed to for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Python is Interpreted

- Many languages are compiled, meaning the source code you create needs to be translated into machine code, the language of your computer's processor, before it can be run. Programs written in an interpreted language are passed straight to an interpreter that runs them directly.
- This makes for a quicker development cycle because you just type in your code and run it, without the intermediate compilation step.
- One potential downside to interpreted languages is execution speed. Programs that are compiled into the native language of the computer processor tend to run more quickly than interpreted programs. For some applications that are particularly computationally intensive, like graphics processing or intense number crunching, this can be limiting.
- In practice, however, for most programs, the difference in execution speed is measured in milliseconds, or seconds at most, and not appreciably noticeable to a human user. The expediency of coding in an interpreted language is typically worth it for most applications.
- For all its syntactical simplicity, Python supports most constructs that would be expected in a very high-level language, including complex dynamic data types, structured and functional programming, and object oriented programming.
- Additionally, a very extensive library of classes and functions is available that provides capability well beyond what is built into the language, such as database manipulation or GUI programming.
- Python accomplishes what many programming languages don't: the language itself is simply designed, but it is very versatile in terms of what you can accomplish with it.

Introduction Machine learning:

Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop a conventional algorithm for effectively performing the task.

Machine learning is closely related to computational statistics, which focuses on making predictions using computers. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a field of study within machine learning, and focuses on exploratory data analysis through learning. In its application across business problems, machine learning is also referred to as predictive analytics.

Machine learning tasks:

Machine learning tasks are classified into several broad categories. In supervised learning, the algorithm builds a mathematical model from a set of data that contains both the inputs and the desired outputs. For example, if the task were determining whether an image contained a certain object, the training data for a supervised learning algorithm would include images with and without that object (the input), and each image would have a label (the output) designating whether it contained the object. In special cases, the input may be only partially available, or restricted to special feedback. Semi algorithms develop mathematical models from incomplete training data, where a portion of the sample input doesn't have labels.

Classification algorithms and regression algorithms are types of supervised learning. Classification algorithms are used when the outputs are restricted to a limited set of

values. For a classification algorithm that filters emails, the input would be an incoming email, and the output would be the name of the folder in which to file the email. For an algorithm that identifies spam emails, the output would be the prediction of either "spam" or "not spam", represented by the Boolean values true and false. Regression algorithms are named for their continuous outputs, meaning they may have any value within a range. Examples of a continuous value are the temperature, length, or price of an object.

In unsupervised learning, the algorithm builds a mathematical model from a set of data that contains only inputs and no desired output labels. Unsupervised learning algorithms are used to find structure in the data, like grouping or clustering of data points. Unsupervised learning can discover patterns in the data, and can group the inputs into categories, as in feature learning. Dimensionality reduction is the process of reducing the number of "features", or inputs, in a set of data.

Active learning algorithms access the desired outputs (training labels) for a limited set of inputs based on a budget and optimize the choice of inputs for which it will acquire training labels. When used interactively, these can be presented to a human user for labeling. Reinforcement learning algorithms are given feedback in the form of positive or negative reinforcement in a dynamic environment and are used in autonomous [HYPERLINK "https://en.wikipedia.org/wiki/Autonomous_vehicle"](https://en.wikipedia.org/wiki/Autonomous_vehicle)vehicles or in learning to play a game against a human opponent. Other specialized algorithms in machine learning include topic modeling, where the computer program is given a set of natural language documents and finds other documents that cover similar topics. Machine learning algorithms can be used to find the unobservable probability density function in density estimation problems. Meta learning algorithms learn their own inductive bias based on previous experience. In developmental robotics, robot learning algorithms generate their own sequences of learning experiences, also known as a curriculum, to cumulatively acquire new skills through self-guided exploration and social interaction with humans. These robots use guidance mechanisms such as active learning, maturation, motor synergies, and imitation.

Types of learning algorithms:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

Supervised learning:

Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs. The data is known as training data, and consists of a set of training examples. Each training example has one or more inputs and the desired output, also known as a supervisory signal. In the mathematical model, each training example is represented by an array or vector, sometimes called a feature vector, and the training data is represented by a matrix. Through iterative optimization of an objective function, supervised learning algorithms learn a function that can be used to predict the output associated with new inputs. An optimal function will allow the algorithm to correctly determine the output for inputs that were not a part of the training data. An algorithm that improves the accuracy of its outputs or predictions over time is said to have learned to perform that task.

Supervised learning algorithms include classification and regression. Classification algorithms are used when the outputs are restricted to a limited set of values, and regression algorithms are used when the outputs may have any numerical value within a range. Similarity learning is an area of supervised machine learning closely related to regression and classification, but the goal is to learn from examples using a similarity function that measures how similar or related two objects are. It has applications in ranking, recommendation systems, visual identity tracking, face verification, and speaker verification.

In the case of semi-supervised learning algorithms, some of the training examples are missing training labels, but they can nevertheless be used to improve the quality of a model. In weakly supervised learning, the training labels are noisy, limited, or imprecise; however, these labels are often cheaper to obtain, resulting in larger effective training sets.

Unsupervised learning:

Unsupervised learning algorithms take a set of data that contains only inputs, and find structure in the data, like grouping or clustering of data points. The algorithms, therefore, learn from test data that has not been labeled, classified or categorized. Instead of responding to feedback, unsupervised learning algorithms identify commonalities in the data and react based on the presence or absence of such commonalities in each new piece of data. A central application of unsupervised learning is in the field of density estimation in statistics, though unsupervised learning encompasses other domains involving summarizing and explaining data features.

Cluster analysis is the assignment of a set of observations into subsets (called *clusters*) so that observations within the same cluster are similar according to one or more pre designated criteria, while observations drawn from different clusters are dissimilar. Different clustering techniques make different assumptions on the structure of the data, often defined by some *similarity metric* and evaluated, for example, by *internal compactness*, or the similarity between members of the same cluster, and *separation*, the difference between clusters. Other methods are based on *estimated density* and *graph connectivity*.

Semi-supervised learning:

Semi-supervised learning falls between unsupervised learning (without any labeled training data) and supervised learning (with completely labeled training data). Many machine-learning researchers have found that unlabeled data, when used in conjunction with a small amount of labeled data, can produce a considerable improvement in learning accuracy.

Introduction to RNN:

Traditional neural networks can't do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It's unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.

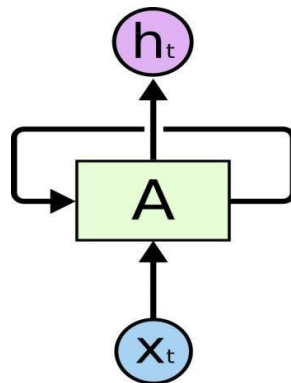


Fig: Recurrent Neural Networks have loops.

In the above diagram, a chunk of neural network, (A) , looks at some input (x_t) and outputs a value (h_t) . A loop allows information to be passed from one step of the network to the next.

These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they aren't all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:

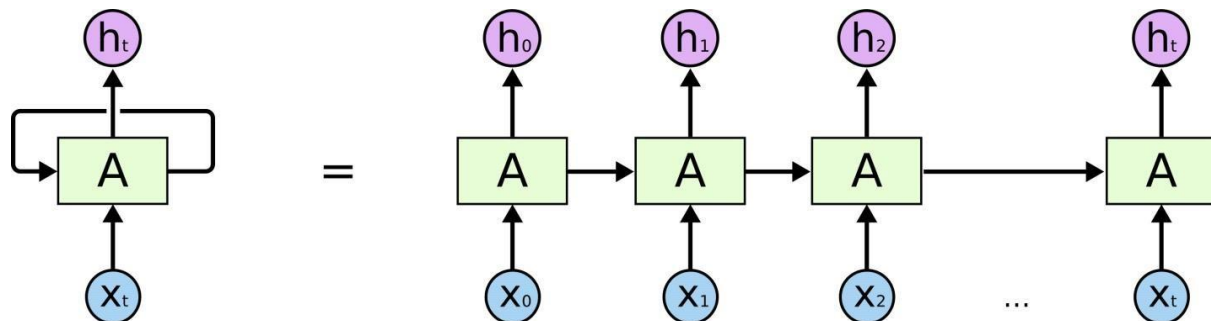


Fig: An unrolled recurrent neural network

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data.

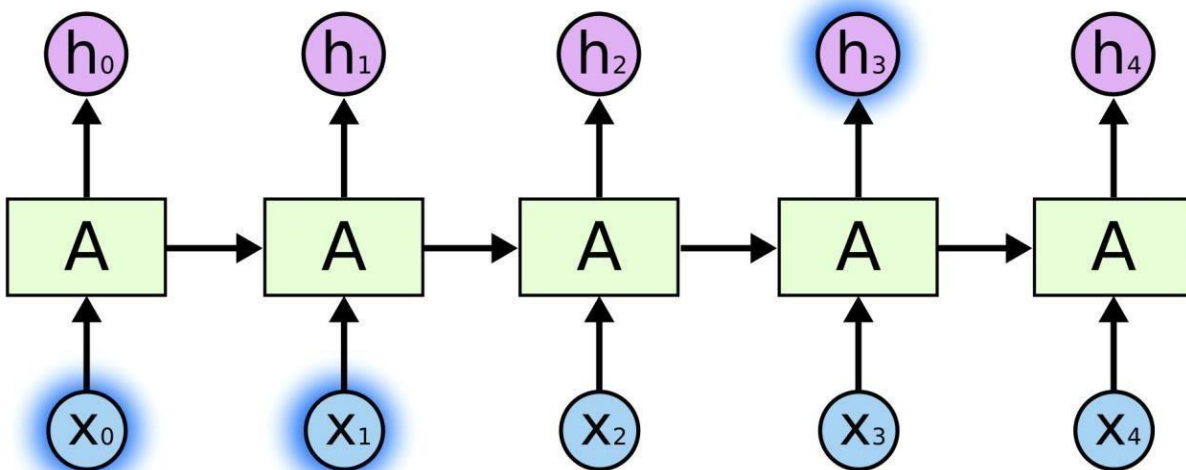
And they certainly are used! In the last few years, there has been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning... The list goes on. But they really are pretty amazing.

Essential to these successes is the use of “LSTMs,” a very special kind of recurrent neural network which works, for many tasks, much better than the standard version. Almost all exciting results based on recurrent neural networks are achieved with them. It's these LSTMs that this essay will explore.

The Problem of Long-Term Dependencies

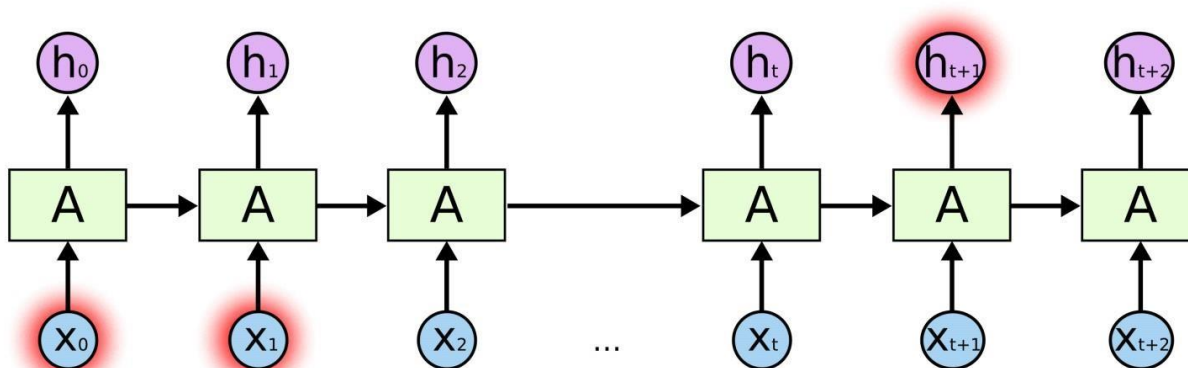
One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame. If RNNs could do this, they'd be extremely useful. But can they? It depends.

Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in “the clouds are in the sky,” we don't need any further context – it's pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.



But there are also cases where we need more contexts. Consider trying to predict the last word in the text “I grew up in France... I speak fluent *French*.” Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It’s entirely possible for the gap between the relevant information and the point where it is needed to become very large.

Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.



RNNs are absolutely capable of handling such “long-term dependencies.” A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don’t seem to be able to learn them. The problem was explored in depth by Hochreiter (1991) [German] and Bengio, et al. (1994), who found some pretty fundamental reasons why it might be difficult.

Thankfully, LSTMs don’t have this problem!

LSTM Networks

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work.¹ They work tremendously well on a large variety of problems, and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

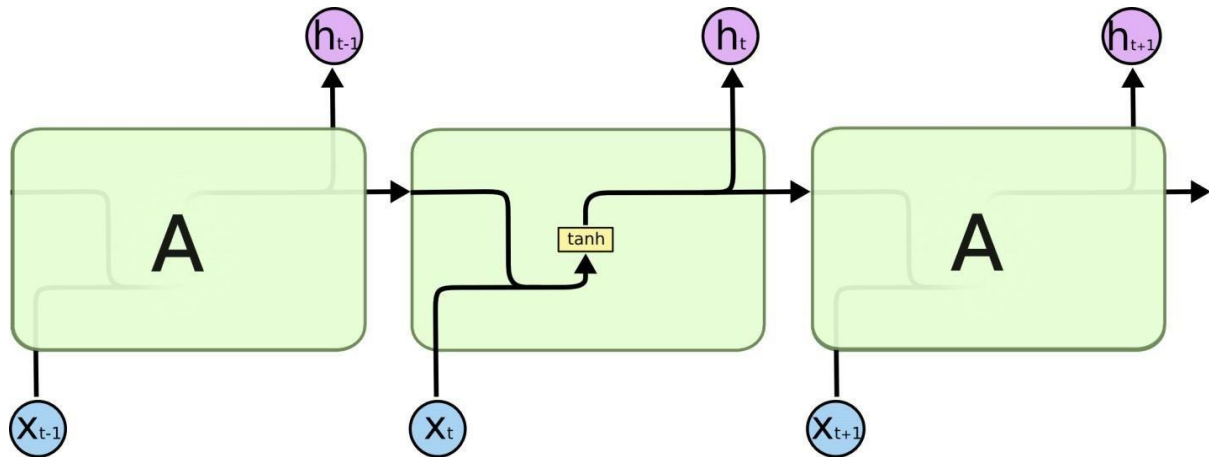


Fig: The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

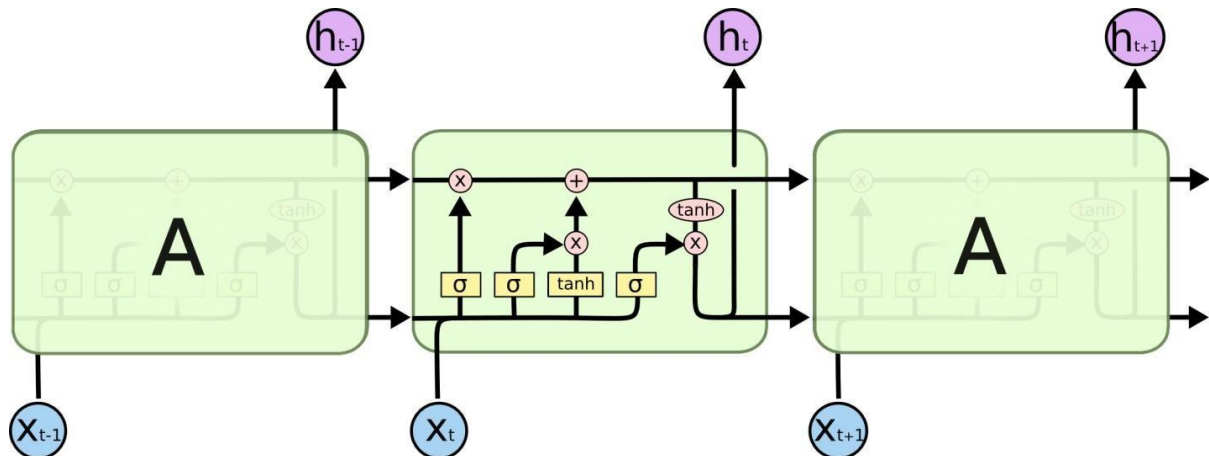
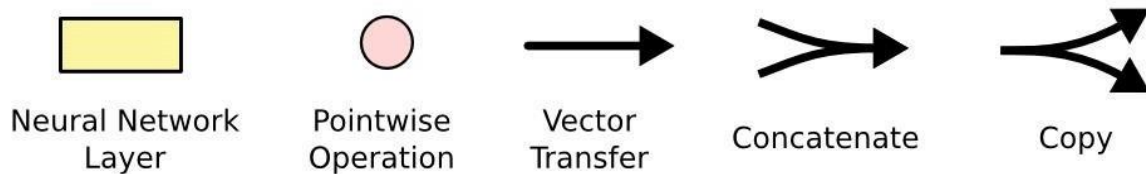


Fig: The repeating module in an LSTM contains four interacting layers.

Don't worry about the details of what's going on. We'll walk through the LSTM diagram step by step later. For now, let's just try to get comfortable with the notation we'll be using.

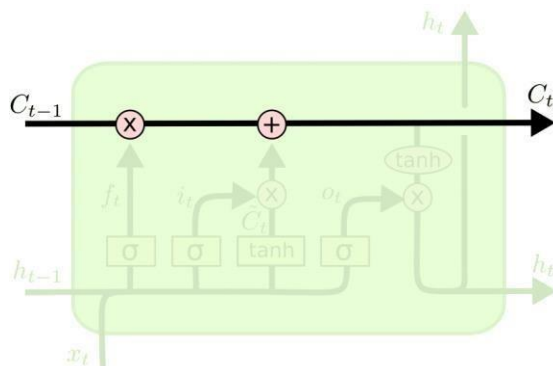


In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

The Core Idea behind LSTMs

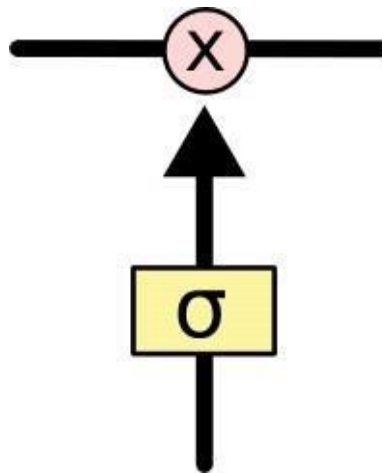
The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.



The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



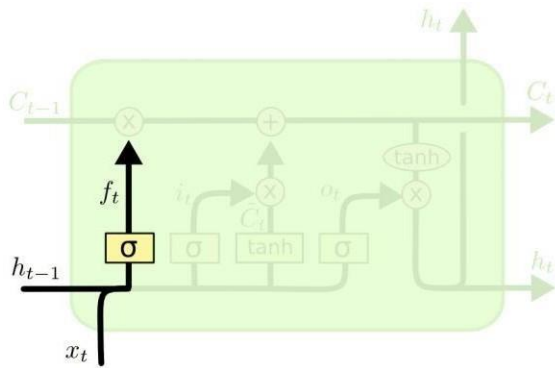
The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”

An LSTM has three of these gates, to protect and control the cell state.

Step-by-Step LSTM Walk Through

The first step in our LSTM is to decide what information we’re going to throw away from the cell state. This decision is made by a sigmoid layer called the “forget gate layer.” It looks at (h_{t-1}) and (x_t) , and outputs a number between (0) and (1) for each number in the cell state (C_{t-1}) . A (1) represents “completely keep this” while a (0) represents “completely get rid of this.”

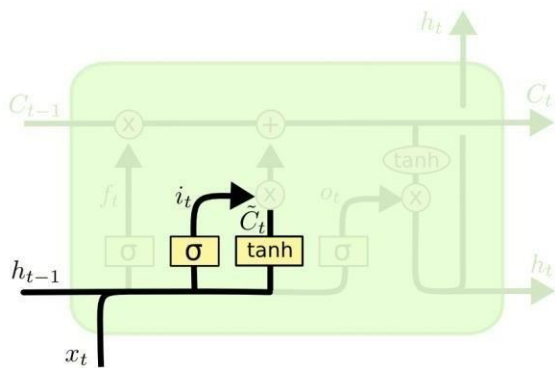
Let’s go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



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

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.

In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

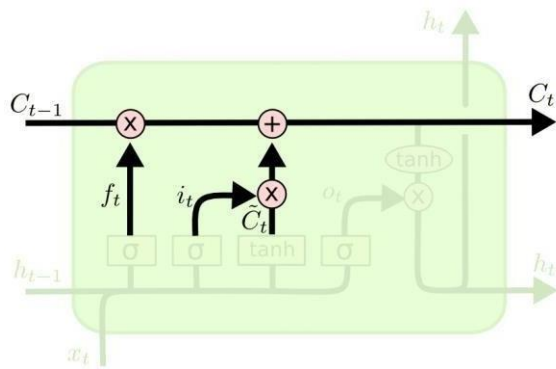
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

It's now time to update the old cell state, C_{t-1} , into the new cell state C_t . The previous steps already decided what to do; we just need to actually do it.

We multiply the old state by f_t , forgetting the things we decided to forget earlier. Then we add $i_t \cdot \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.

In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous

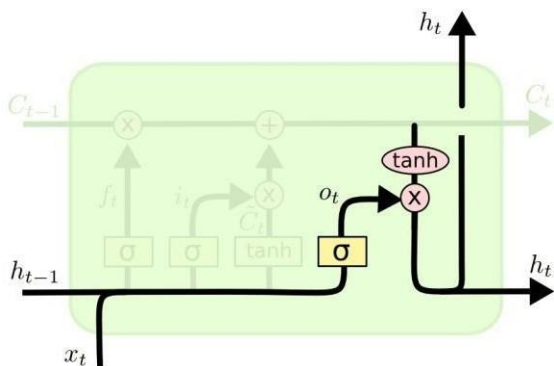
steps.



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

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through $\tanh$ (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.



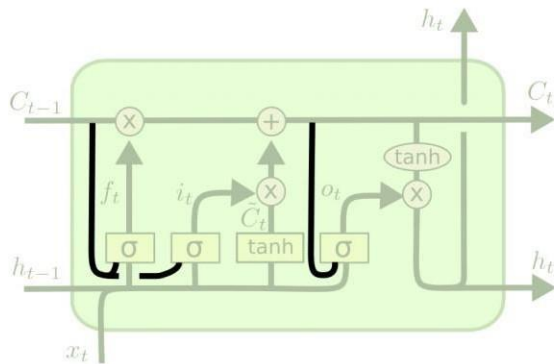
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

Variants on Long Short Term Memory

What I've described so far is a pretty normal LSTM. But not all LSTMs are the same as the above. In fact, it seems like almost every paper involving LSTMs uses a slightly different version. The differences are minor, but it's worth mentioning some of them.

One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding "peephole connections." This means that we let the gate layers look at the cell state.



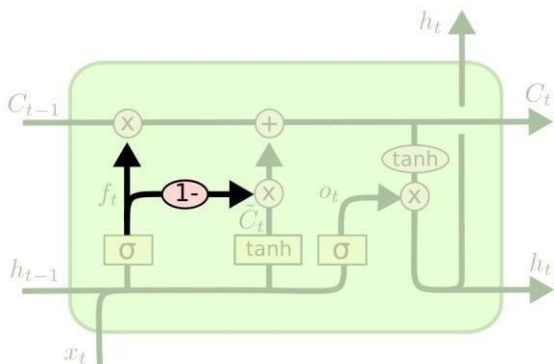
$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$

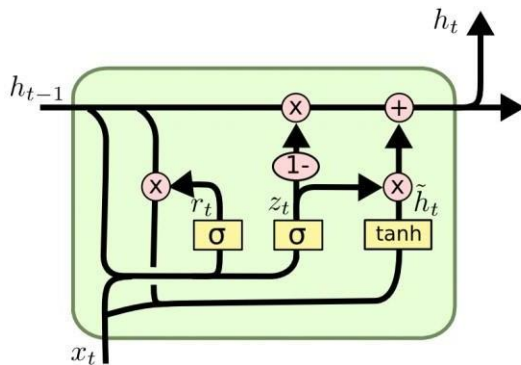
The above diagram adds peepholes to all the gates, but many papers will give some peepholes and not others.

Another variation is to use coupled forget and input gates. Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when we're going to input something in its place. We only input new values to the state when we forget something older.



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

A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014). It combines the forget and input gates into a single “update gate.” It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

These are only a few of the most notable LSTM variants. There are lots of others, like Depth Gated RNNs by Yao, et al. (2015). There’s also some completely different approach to tackling long-term dependencies, like Clockwork RNNs by Koutnik, et al. (2014).

Which of these variants is best? Do the differences matter? Greff, et al. (2015) do a nice comparison of popular variants, finding that they’re all about the same. Jozefowicz, et al. (2015) tested more than ten thousand RNN architectures, finding some that worked better than LSTMs on certain .

Design and Implementation Constraints

Constraints in Analysis

- Constraints as Informal Text
- Constraints as Operational Restrictions
- Constraints Integrated in Existing Model Concepts
- Constraints as a Separate Concept
- Constraints Implied by the Model Structure

Constraints in Design

- Determination of the Involved Classes
- Determination of the Involved Objects
- Determination of the Involved Actions
- Determination of the Require Clauses
- Global actions and Constraint Realization

Constraints in Implementation

A hierarchical structuring of relations may result in more classes and a more complicated structure to implement. Therefore, it is advisable to transform the hierarchical relation structure to a simpler structure such as a classical flat one. It is rather straightforward to transform the developed hierarchical model into a

bipartite, flat model, consisting of classes on the one hand and flat relations on the other. Flat relations are preferred at the design level for reasons of simplicity and implementation ease. There is no identity or functionality associated with a flat relation. A flat relation corresponds with the relation concept of entity-relationship modeling and many object oriented methods.

Other Nonfunctional Requirements

Performance Requirements

The application at this side controls and communicates with the following three main general components.

- embedded browser in charge of the navigation and accessing to the web service;
- Server Tier: The server side contains the main parts of the functionality of the proposed architecture. The components at this tier are the following.

Web Server, Security Module, Server-Side Capturing Engine, Preprocessing Engine, Database System, Verification Engine, Output Module.

Safety Requirements

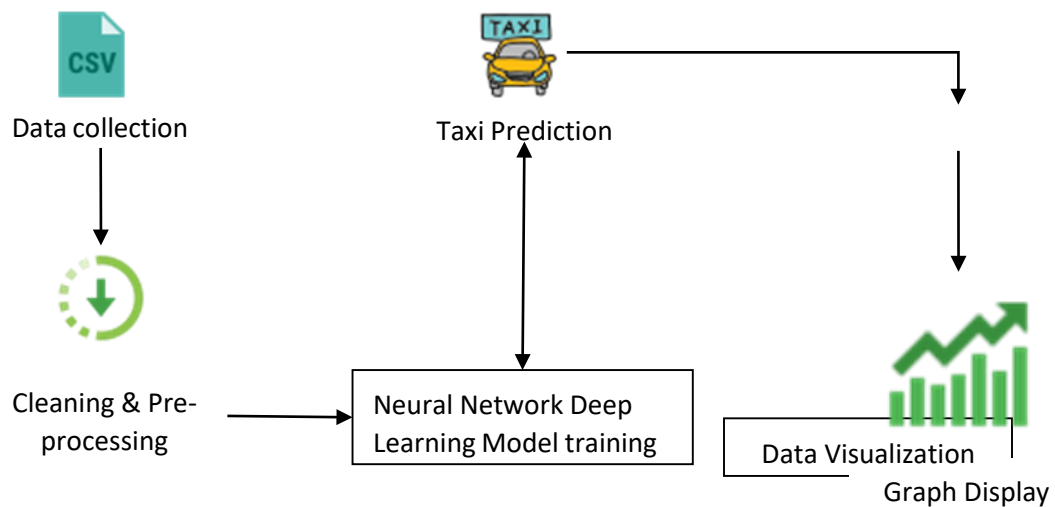
- The software may be safety-critical. If so, there are issues associated with its integrity level
- The software may not be safety-critical although it forms part of a safetycritical system. For example, software may simply log transactions.

- If a system must be of a high integrity level and if the software is shown to be of that integrity level, then the hardware must be at least of the same integrity level.
- There is little point in producing 'perfect' code in some language if hardware and system software (in widest sense) are not reliable.
- If a computer system is to run software of a high integrity level then that system should not at the same time accommodate software of a lower integrity level.
- Systems with different requirements for safety levels must be separated.
- Otherwise, the highest level of integrity required must be applied to all systems in the same environment.

CHAPTER 4

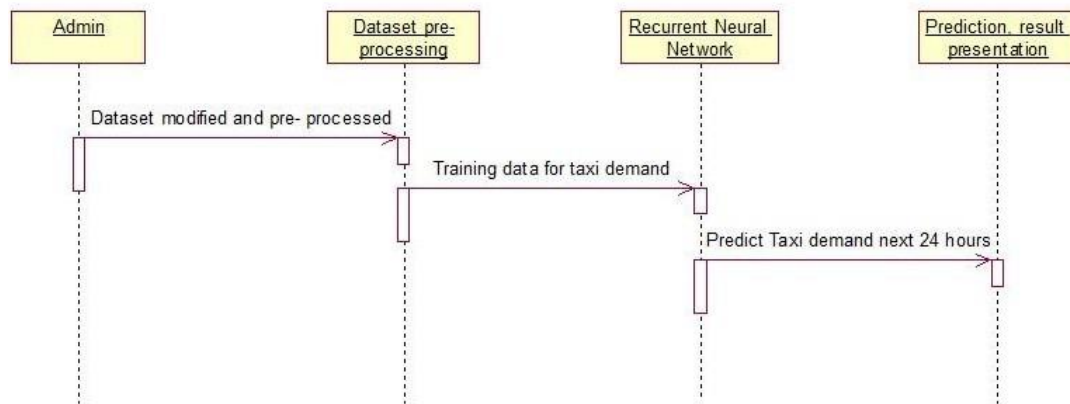
Architecture Diagram:

Fig: 4.1



Sequence Diagram:

A Sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of Message Sequence diagrams are sometimes called event diagrams, event sceneries and timing diagram.



Use Case Diagram:

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. The standard is managed and was created by the Object Management Group. UML includes a set of graphic notation techniques to create visual models of software intensive systems. This language is used to specify, visualize, modify, construct and document the artifacts of an object oriented software intensive system under development.

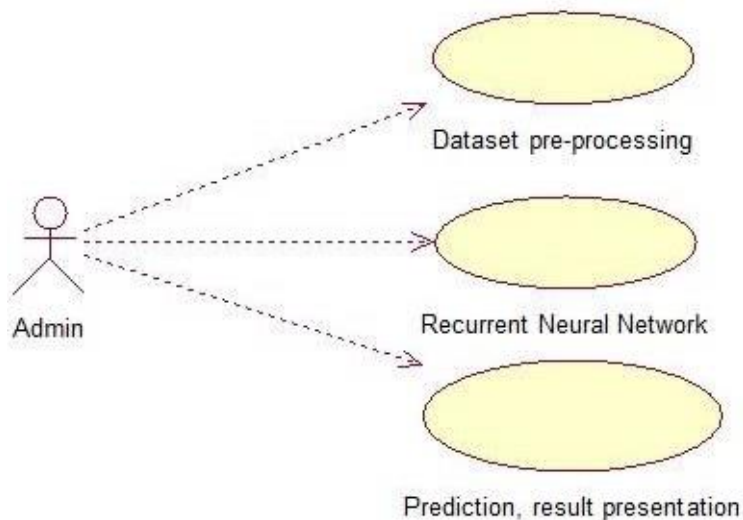
USECASE DIAGRAM

A Use case Diagram is used to present a graphical overview of the functionality provided by a system in terms of actors, their goals and any dependencies between those use cases.

Use case diagram consists of two parts:

Use case: A use case describes a sequence of actions that provided something of measurable value to an actor and is drawn as a horizontal ellipse.

Actor: An actor is a person, organization or external system that plays a role in one or more interaction with the system.



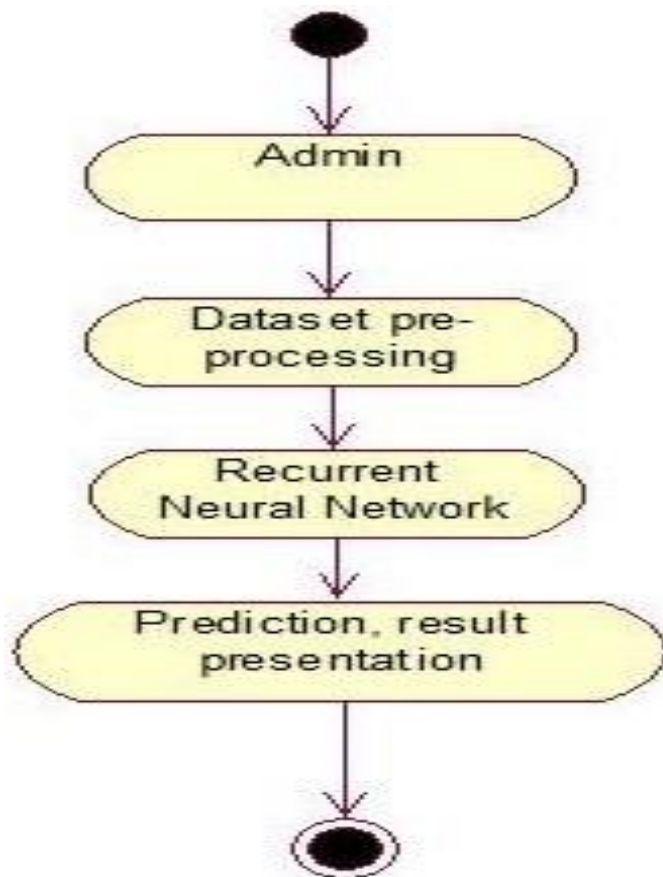
Activity Diagram:

Activity diagram is a graphical representation of workflows of stepwise activities and actions with support for choice, iteration and concurrency. An activity diagram shows the overall flow of control.

The most important shape types:

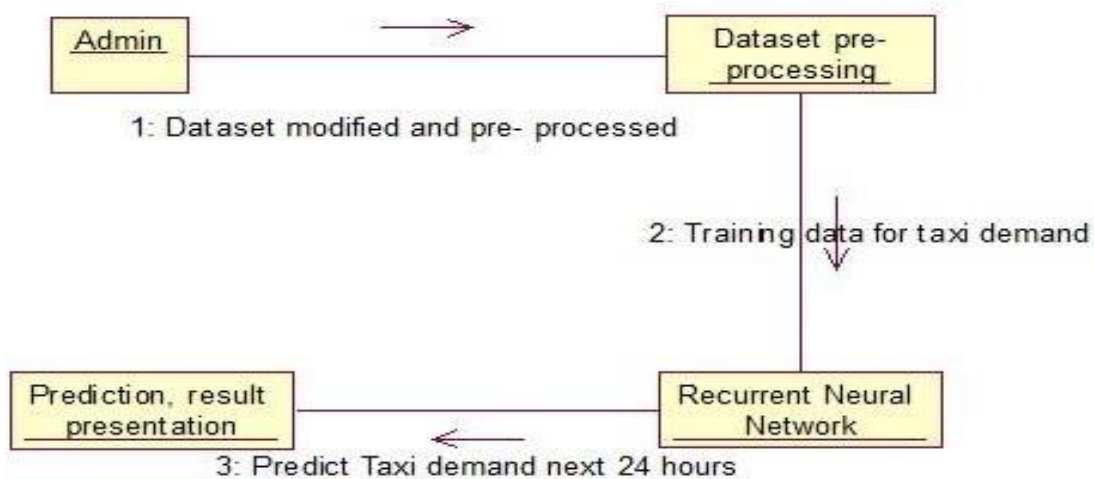
- Rounded rectangles represent activities.
- Diamonds represent decisions.
- Bars represent the start or end of concurrent activities.

- A black circle represents the start of the workflow.
- An encircled circle represents the end of the workflow.



Collaboration Diagram:

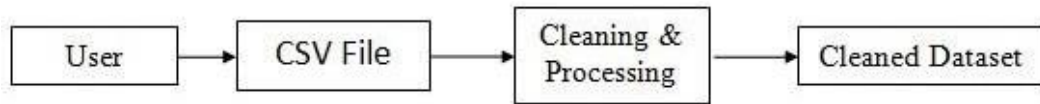
UML Collaboration Diagrams illustrate the relationship and interaction between software objects. They require use cases, system operation contracts and domain model to already exist. The collaboration diagram illustrates messages being sent between classes and objects.



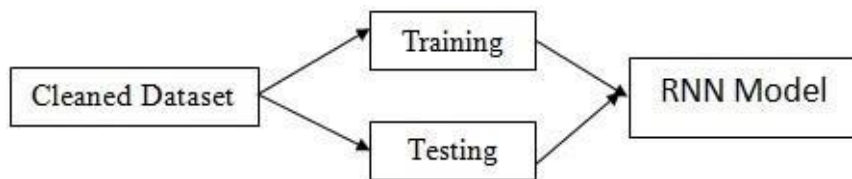
DATA FLOW DIAGRAM:

A Data Flow Diagram (DFD) is a graphical representation of the “flow” of data through an information system, modeling its aspects. It is a preliminary step used to create an overview of the system which can later be elaborated DFDs can also be used for visualization of data processing.

Level 1:



Level 2:



Level 3:



CHAPTER 5

MODULES & FUTURE WORK

- Dataset pre-processing • Recurrent Neural Network
- Prediction, result presentation

MODULE EXPLANATION:

Dataset pre-processing:

We like to use Python language because it has vast collection of machine learning libraries. First to read live tweet from twitter, we need to use twitter API for python. The dataset might contain empty values, negative values or error. Dataset is cleaned in the pre-processing. The preprocessing methods involve of removing records which is not complete. Once the clean dataset is available we have to prepare it to feed to the machine learning algorithm.

Recurrent Neural Network:

The network input is the current taxi demand and other relevant information while the output is the demand in the next time-step. The reason we use a recurrent neural network is that it can be trained to store all the relevant information in a sequence to predict particular outcomes in the future. In addition, taxi demand prediction is a time series forecasting problem in which an intelligent sequence analysis model is required. We divide the entire city into small areas. It is desired to predict taxi demand in small areas so that the drivers know exactly where to go. We train our system with dataset and create the model for future prediction.

Prediction, result presentation:

A graph is plotted for the future prediction for the next time slot and the area to be crowded. This machine learning model predicts the future demand area in a city based on NN and the drivers were taken to wait in the area where the system identified as demand area.

CODING AND TESTING

CODING

Once the design aspect of the system is finalized the system enters into the coding and testing phase. The coding phase brings the actual system into action by converting the design of the system into the code in a given programming language. Therefore, a good coding style has to be taken whenever changes are required it easily screws into the system.

CODING STANDARDS

Coding standards are guidelines to programming that focus on the physical structure and appearance of the program. They make the code easier to read, understand and maintain. This phase of the system actually implements the blueprint developed during the design phase. The coding specification should be in such a way that any programmer must be able to understand the code and can bring about changes whenever felt necessary. Some of the standards needed to achieve the abovementioned objectives are as follows:

Program should be simple, clear and easy to understand.

1. Naming conventions

2. Value conventions

3. Script and comment procedure

4. Message box format

5. Exception and error handling

NAMING CONVENTIONS

Naming conventions of classes, data member, member functions, procedures etc., should be **self-descriptive**. One should even get the meaning and scope of the variable by its name. The conventions are adopted for **easy understanding** of the intended message by the user. So, it is customary to follow the conventions. These conventions are as follows:

Class names

Class names are problem domain equivalence and begin with capital letter and have mixed cases.

Member Function and Data Member name

Member function and data member name begins with a lowercase letter with each subsequent letters of the new words in uppercase and the rest of letters in lowercase.

VALUE CONVENTIONS

Value conventions ensure values for variable at any point of time. This involves the following:

- Proper default values for the variables.
- Proper validation of values in the field.
- Proper documentation of flag values.

SCRIPT WRITING AND COMMENTING STANDARD

Script writing is an art in which indentation is utmost important. Conditional and looping statements are to be properly aligned to facilitate easy understanding. Comments are included to minimize the number of surprises that could occur when going through the code.

MESSAGE BOX FORMAT

When something has to be prompted to the user, he must be able to understand it properly. To achieve this, a specific format has been adopted in displaying messages to the user. They are as follows:

- X – User has performed illegal operation.
- ! – Information to the user.

TEST PROCEDURE SYSTEM TESTING

Testing is performed to identify errors. It is used for quality assurance. Testing is an integral part of the entire development and maintenance process. The goal of the testing during phase is to verify that the specification has been accurately and completely incorporated into the design, as well as to ensure the correctness of the design itself. For example, the design must not have any logic faults in the design is detected before coding commences, otherwise the cost of fixing the faults will be considerably higher as reflected. Detection of design faults can be achieved by means of inspection as well as walkthrough.

Testing is one of the important steps in the software development phase. Testing checks for the errors, as a whole of the project testing involves the following test cases:

- Static analysis is used to investigate the structural properties of the Source code.

- Dynamic testing is used to investigate the behavior of the source code by executing the program on the test data.

TEST DATA AND OUTPUT

UNIT TESTING:

Unit testing is conducted to verify the functional performance of each modular component of the software. Unit testing focuses on the smallest unit of the software design (i.e.), the module. The white-box testing techniques were heavily employed for unit testing.

FUNCTIONAL TESTING:

Functional test cases involved exercising the code with nominal input values for which the expected results are known, as well as boundary values and special values, such as logically related inputs, files of identical elements, and empty files.

Three types of tests in Functional test:

- Performance Test
- Stress Test
- Structure Test

PERFORMANCE TEST:

It determines the amount of execution time spent in various parts of the unit, program throughput, and response time and device utilization by the program unit.

STRESS TEST:

Stress Test is those test designed to intentionally break the unit. A Great deal can be learned about the strength and limitations of a program by examining the manner in which a programmer in which a program unit breaks.

STRUCTURED TEST:

Structure Tests are concerned with exercising the internal logic of a program and traversing particular execution paths. The way in which White-Box test strategy was employed to ensure that the test cases could Guarantee that all independent paths within a module have been have been exercised at least once.

- Exercise all logical decisions on their true or false sides.
- Execute all loops at their boundaries and within their operational bounds.
- Exercise internal data structures to assure their validity.
- Checking attributes for their correctness.
- Handling end of file condition, I/O errors, buffer problems and textual errors in output information

INTEGRATION TESTING:

Integration testing is a systematic technique for construction the program structure while at the same time conducting tests to uncover errors associated with interfacing. i.e., integration testing is the complete testing of the set of modules which makes up the product. The objective is to take untested modules and build a program structure tester should identify critical modules. Critical modules should be tested as early as possible. One approach is to wait until all the units have passed testing, and then combine them and then tested. This approach is evolved from unstructured testing of small programs. Another strategy is to construct the product in increments of tested units. A small set of modules are integrated together and tested, to which another module is added and tested in combination. And so on. The advantages of this approach are that, interface dispenses can be easily found and corrected.

The major error that was faced during the project is linking error. When all the modules are combined the link is not set properly with all support files. Then we checked out for interconnection and the links. Errors are localized to the new module and its intercommunications. The product development can be staged, and modules integrated in as they complete unit testing. Testing is completed when the last module is integrated and tested.

TESTING TECHNIQUES / TESTING STRATEGIES

TESTING :

Testing is a process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an as-yet –undiscovered error. A successful test is one that uncovers an as-yet- undiscovered error. System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently as expected before live operation commences. It verifies that the whole set of programs hang together. System testing requires a test consists of several key activities and steps for run program, string, system and is important in adopting a successful new system. This is the last chance to detect and correct errors before the system is installed for user acceptance testing.

The software testing process commences once the program is created and the documentation and related data structures are designed. Software testing is essential for correcting errors. Otherwise the program or the project is not said to be complete. Software testing is the critical element of software quality assurance and represents the ultimate the review of specification design and coding. Testing is the process of executing the program with the intent of finding the error. A good test case design is one that as a probability of finding an yet undiscovered error. A successful test is one that uncovers an yet undiscovered error. Any engineering product can be tested in one of the two ways:

WHITE BOX TESTING

This testing is also called as Glass box testing. In this testing, by knowing the specific functions that a product has been design to perform test can be conducted that demonstrate each function is fully operational at the same time searching for errors

in each function. It is a test case design method that uses the control structure of the procedural design to derive test cases. Basis path testing is a white box testing.

Basis path testing:

- Flow graph notation
- Cyclometric complexity
- Deriving test cases
- Graph matrices Control

BLACK BOX TESTING

In this testing by knowing the internal operation of a product, test can be conducted to ensure that “all gears mesh”, that is the internal operation performs according to specification and all internal components have been adequately exercised. It fundamentally focuses on the functional requirements of the software.

The steps involved in black box test case design are:

- Graph based testing methods
- Equivalence partitioning
- Boundary value analysis
- Comparison testing

SOFTWARE TESTING STRATEGIES:

A software testing strategy provides a road map for the software developer.

Testing is a set activity that can be planned in advance and conducted systematically. For this reason a template for software testing a set of steps into which we can place specific test case design methods should be strategy should have the following characteristics:

- Testing begins at the module level and works “outward” toward the integration of the entire computer based system.
- Different testing techniques are appropriate at different points in time.
- The developer of the software and an independent test group conducts testing.
- Testing and Debugging are different activities but debugging must be accommodated in any testing strategy.

INTEGRATION TESTING:

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with. Individual modules, which are highly prone to interface errors, should not be assumed to work instantly when we put them together. The problem of course, is “putting them together”- interfacing. There may be the chances of data lost across on another’s sub functions, when combined may not produce the desired major function; individually acceptable impression may be magnified to unacceptable levels; global data structures can present problems.

PROGRAM TESTING:

The logical and syntax errors have been pointed out by program testing. A syntax error is an error in a program statement that in violates one or more rules of the language in which it is written. An improperly defined field dimension or omitted keywords are common syntax error. These errors are shown through error messages generated by the computer. A logic error on the other hand deals with the incorrect data fields, out-off-range items and invalid combinations. Since the compiler s will not deduct logical error, the programmer must examine the output. Condition testing exercises the logical conditions contained in a module. The possible types of elements in a condition include a Boolean operator, Boolean variable, a pair of Boolean parentheses A relational operator or on arithmetic expression. Condition testing method focuses on testing each condition in the program the purpose of condition test is to deduct not only errors in the condition of a program but also other a errors in the program.

SECURITY TESTING:

Security testing attempts to verify the protection mechanisms built in to a system well, in fact, protect it from improper penetration. The system security must be tested for invulnerability from frontal attack must also be tested for invulnerability from rear attack. During security, the tester places the role of individual who desires to penetrate system.

VALIDATION TESTING

At the culmination of integration testing, software is completely assembled as a package. Interfacing errors have been uncovered and corrected and a final series of software test-validation testing begins. Validation testing can be defined in many ways,

but a simple definition is that validation succeeds when the software functions in manner that is reasonably expected by the customer. Software validation is achieved through a series of black box tests that demonstrate conformity with requirement. After validation test has been conducted, one of two conditions exist.

- * The function or performance characteristics confirm to specifications and are accepted.
- * A validation from specification is uncovered and a deficiency created.

Deviation or errors discovered at this step in this project is corrected prior to completion of the project with the help of the user by negotiating to establish a method for resolving deficiencies. Thus, the proposed system under consideration has been tested by using validation testing and found to be working satisfactorily. Though there were deficiencies in the system they were not catastrophic

USER ACCEPTANCE TESTING

User acceptance of the system is key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system and user at the time of developing and making changes whenever required. This is done in regarding to the following points.

- Input screen design
- Output screen design.

SOURCE CODE:

```
import os
```

```
import pickle
```

```
import copy
```

```
import datetime as dt

from datetime import timedelta

import pandas as pd

from flask import Flask

import dash

from dash.dependencies import Input, Output, State

import dash_core_components as dcc

import dash_html_components as html

import pandas as pd

import numpy as np

from pandas.tseries.holiday import USFederalHolidayCalendar as calendar

from sklearn.externals import joblib

df = pd.read_csv('C:/Python/Real-Time Prediction of Taxi Demand
FULL/clean_data/FinalData_for_Models.csv')

loaded_model = joblib.load("model_rfgbtxgb_la.pkl")

hourly_biases = pd.read_csv('C:/Python/Real-Time Prediction of Taxi
Demand FULL/clean_data/biases_per_hour.csv', index_col = 0)

hourly_biases = hourly_biases.values
```

```
app = dash.Dash()
```

```
app.css.append_css({'external_url': 'https://cdn.rawgit.com/plotly/dash-app-  
stylesheets/2d266c578d2a6e8850ebce48fdb52759b2aef506/styleSheet-oil-  
and- gas.css'})
```

```
app.title = "LGA Taxi Demand"
```

```
layout = dict(
```

```
    autosize=True,
```

```
    height=500,
```

```
    font=dict(color='#CCCCCC'),
```

```
    titlefont=dict(color='#CCCCCC', size='14'),
```

```
    margin=dict(
```

```
        l=35,
```

```
        r=35,
```

```
        b=35,
```

```
        t=45
```

```
    ),
```

```
    hovermode="closest",
```

```
    plot_bgcolor="#191A1A",
```

```

paper_bgcolor="#020202",

legend=dict(font=dict(size=10), orientation='h'),

title='Satellite Overview',

)

app.layout = html.Div(

    [

        html.Div(

            [

                html.H1(

                    'Estimating Taxi Demand at LaGuardia Airport',

                    className='ten columns', style = {'font-weight':'bold'}

                ),

            ],

            className='row', style={'margin-bottom':'20px', 'color' : '#000000'}

        ),

        html.Div(

            [

```

```

html.H6('Date', className='one columns'),

html.Div([dcc.DatePickerSingle(

    id='my-date-picker-single',

    min_date_allowed=dt.datetime(2014, 1, 1),

    max_date_allowed=dt.datetime(2020, 12, 31),

    initial_visible_month=dt.datetime(2017, 12, 11),

    date=dt.datetime(2017, 12, 11)

)], className='two columns'),

],

className='row'

),

html.Div(

    [

        html.P('Select Hour:'),

        dcc.Slider(

            id='hour',

            min=0,

```

```

        max=23,

        marks={i: '{}'.format(i) for i in range(0, 24)},

        value=10

    ),

], className='five columns',

style={'margin-top': '20', 'margin-bottom': '30px', 'position':'relative'}

),

html.Div(

    [

        html.Div(

            [

                html.P('Filter by Precipitation:'),

                dcc.Dropdown(

                    id='precipitation',

                    options=[

                        {'label': 'None', 'value': 0},

                        {'label': 'Light', 'value': 0.04},

```

```

        {'label': 'Medium', 'value': 0.13},

        {'label': 'Heavy', 'value': 0.35}

    ],

    value=0

),

],

    className='five columns', style={'margin-left':'600px', 'margin-
top': '-90px', 'display':'block'}

),

html.Div(

    [

        html.P('Input Temperature (deg F):'),

        dcc.Input(id='temp', value=70, type='int'),

    ],

    className='six columns', style={'margin-bottom':'80px'}

),

],

```

```

className='row', style= {'margin-top': '110px'}

),

html.Div(

    [

        html.P('Filter by Weather Category:'),

        dcc.Dropdown(

            id='weather',

            options=[

                {'label': 'Clear', 'value': 'clear'},

                {'label': 'Clouds', 'value': 'clouds'},

                {'label': 'Fog', 'value': 'fog'},

                {'label': 'Rain', 'value': 'rain'},

                {'label': 'Snow', 'value': 'snow'},

                {'label': 'Thunderstorm', 'value': 'thunderstorm'},

            ],

            value=['clear'],

            multi=True

```

```

        )

    ],

    className='five columns', style={'margin-left':'600px', 'margin-
top': '- 130px', 'display':'block', 'margin-bottom': '40px'}

    ),

    html.Div(

        [

            html.P('Estimated Number of Pickups', style={'font-size': '3.0rem',
'margin- bottom':'-400px', 'position':'absolute', 'font-weight':'bold'}),

            html.Div(id='Prediction',

                className='four columns',

                style={'margin-top': '20', 'font-size': '16rem', 'color': '#003406',
'align':'center', 'margin-left':'50px'}

            ),

            html.Div(

                [

                    dcc.Graph(id='predict_graph')

                ],

```

```

        className='seven columns',

        style={'margin-top': '20'}

    ),

],

    className='row'

),

],

    className='ten columns offset-by-one'

)

@app.callback(

    dash.dependencies.Output('Prediction', 'children'),

    [dash.dependencies.Input('my-date-picker-single', 'date'),

    dash.dependencies.Input('precipitation', 'value'),

    dash.dependencies.Input('weather', 'value'),

    dash.dependencies.Input('temp', 'value'),

    dash.dependencies.Input('hour', 'value')])

def update_prediction(date, precipitation, weather, temp, hour):

```

```

new_date = pd.to_datetime(date, format='%Y-%m-%d')

month = new_date.month

months = np.array([month])

day = new_date.dayofweek

days = np.array([day])

weather_dummies = weather_format(weather)

if temp == "":

    temp = 0.0

temp_K = np.array([(float(temp) + 459.67)*(5.0/9.0)])

hours = np.array([hour])

hol = is_holiday(date)

humidity = np.array([df.loc[(df.Month == month) & (df.Day == day) &
(df.Hour == hour), 'humidity'].mean()])

wind_speed = np.array([df.loc[(df.Month == month) & (df.Day == day) &
(df.Hour == hour), 'wind_speed'].mean()])

flight_info = df.loc[(df.Hour == hour) & (df.Day == day) & (df.holiday ==
hol), ['Passengers', 'Avg_Delay_Arriving',
'Cancelled_Departing_Flights']].mean()

```

```
passengers, delays, cancellations = flight_info.values
```

```
last_day = day
```

```
last_2day = day
```

```
last_hour = hour - 1
```

```
last_2hour = hour - 2
```

```
if hour == 0:
```

```
    last_hour = 23
```

```
    last_2hour = 22
```

```
    if day == 0:
```

```
        last_day = 6
```

```
        last_2day = 6
```

```
    else:
```

```
        last_day = day-1
```

```
        last_2day = day-1
```

```
if hour == 1:
```

```
    last_2hour = 23
```

```
    if day == 0:
```

```

        last_2day = 6

    else:

        last_2day = day-1

    prev_hour_pass = np.array([df.loc[(df.Hour == last_hour) & (df.Day ==
last_day), 'Passengers'].mean()])

    prev_2hour_pass = np.array([df.loc[(df.Hour == last_2hour) & (df.Day ==
last_2day), 'Passengers'].mean()])

    hol_array = np.array([int(hol)])

    precipitation = np.array([precipitation])

    feature_vect = np.concatenate((temp_K, humidity, wind_speed,
np.array([passengers]), months, hours, days, hol_array, precipitation,

                                np.array([delays]), np.array([cancellations]),
weather_dummies, prev_hour_pass,

                                prev_2hour_pass))

    feature_vect = feature_vect.reshape(1, len(feature_vect))

    return ('{}'.format(int(max(loader_model.predict(feature_vect),
0)+hourly_biases[hour])))

@app.callback(

    dash.dependencies.Output('predict_graph', 'figure'),

```

```

[dash.dependencies.Input('my-date-picker-single', 'date'),

dash.dependencies.Input('precipitation', 'value'),

dash.dependencies.Input('weather', 'value'),

dash.dependencies.Input('temp', 'value'),

dash.dependencies.Input('hour', 'value'))

def update_prediction_graph(date, precipitation, weather, temp, hour):

    layout_pred = copy.deepcopy(layout)

    new_date = pd.to_datetime(date, format='%Y-%m-%d')

    axis_hours = ["{} hour".format((hour+i)%24) for i in range(7) ]

    prediction_list = []

    my_hours = np.zeros(7)

    for i in range(7):

        my_hours[i] = hour + i

        if my_hours[i] > 23:

            my_hours[i] = my_hours[i] - 24

            prediction_list.append(update_featurevect(new_date +
pd.DateOffset(years=1), precipitation, weather, temp, int(my_hours[i])))

```

```

else:

    prediction_list.append(update_featurevect(new_date, precipitation,
weather, temp, int(my_hours[i])))

    data= [dict(x=axis_hours, y=prediction_list, type= 'line')]

    layout_pred['title'] = 'Estimated Number of Taxi Pickups for Next 6 Hours'

    layout_pred['showlegend'] = False

    layout_pred['yaxis'] = dict(range=[0,1000])

    figure = dict(data=data, layout=layout_pred)

    return figure

def update_featurevect(date, precipitation, weather, temp, hour):

    new_date = date

    month = new_date.month

    months = np.array([month])

    day = new_date.dayofweek

    days = np.array([day])

    weather_dummies = weather_format(weather)

    if temp == "":

```

```

temp = 0.0

temp_K = np.array([(float(temp) + 459.67)*(5.0/9.0)])

hours = np.array([hour])

hol = is_holiday(date)

humidity = np.array([df.loc[(df.Month == month) & (df.Day == day) &
(df.Hour == hour), 'humidity'].mean()])

wind_speed = np.array([df.loc[(df.Month == month) & (df.Day == day) &
(df.Hour == hour), 'wind_speed'].mean()])

flight_info = df.loc[(df.Hour == hour) & (df.Day == day) & (df.holiday ==
hol), ['Passengers', 'Avg_Delay_Arriving',
'Cancelled_Departing_Flights']].mean()

passengers, delays, cancellations = flight_info.values

last_day = day

last_2day = day

last_hour = hour - 1

last_2hour = hour - 2

if hour == 0:

    last_hour = 23

```

```

last_2hour = 22

if day == 0:

    last_day = 6

    last_2day = 6

else:

    last_day = day-1

    last_2day = day-1

if hour == 1:

    last_2hour = 23

    if day == 0:

        last_2day = 6

    else:

        last_2day = day-1

    prev_hour_pass = np.array([df.loc[(df.Hour == last_hour) & (df.Day ==
last_day), 'Passengers'].mean()])

    prev_2hour_pass = np.array([df.loc[(df.Hour == last_2hour) & (df.Day ==
last_2day), 'Passengers'].mean()])

    hol_array = np.array([int(hol)])

```

```

precipitation = np.array([precipitation])

feature_vect = np.concatenate((temp_K, humidity, wind_speed,
np.array([passengers]), months, hours, days, hol_array, precipitation,

                                np.array([delays]), np.array([cancellations]),
weather_dummies, prev_hour_pass,

                                prev_2hour_pass))

feature_vect = feature_vect.reshape(1, len(feature_vect))

print(feature_vect.shape)

print(feature_vect.shape)

return
int(max(loader_model.predict(feature_vect),0)+hourly_biases[hour])

def day_format(day):

    days = np.zeros(7)

    days[day] = 1

    return days

def month_format(month):

    months = np.zeros(12)

    months[month-1] = 1

```

```
return months
```

```
def weather_format(weather):
```

```
    array = np.zeros(6)
```

```
    if 'clear' in weather:
```

```
        array[0] = 1
```

```
    if 'clouds' in weather:
```

```
        array[1] = 1
```

```
    if 'fog' in weather:
```

```
        array[2] = 1
```

```
    if 'rain' in weather:
```

```
        array[3] = 1
```

```
    if 'snow' in weather:
```

```
        array[4] = 1
```

```
    if 'thunderstorm' in weather:
```

```
        array[5] = 1
```

```
    return array
```

```
def hour_format(hour):
```

```

hours = np.zeros(4)

if hour < 7:

    hours[0] = 1

elif hour < 13:

    hours[1] = 1

elif hour < 19:

    hours[2] = 1

else:

    hours[3] = 1

return hours

def is_holiday(date):

    dr = pd.date_range(pd.Timestamp('2014-01-01'), pd.Timestamp('2020-
12-31'))

    cal = calendar()

    holidays = cal.holidays(start=dr.min(), end=dr.max())

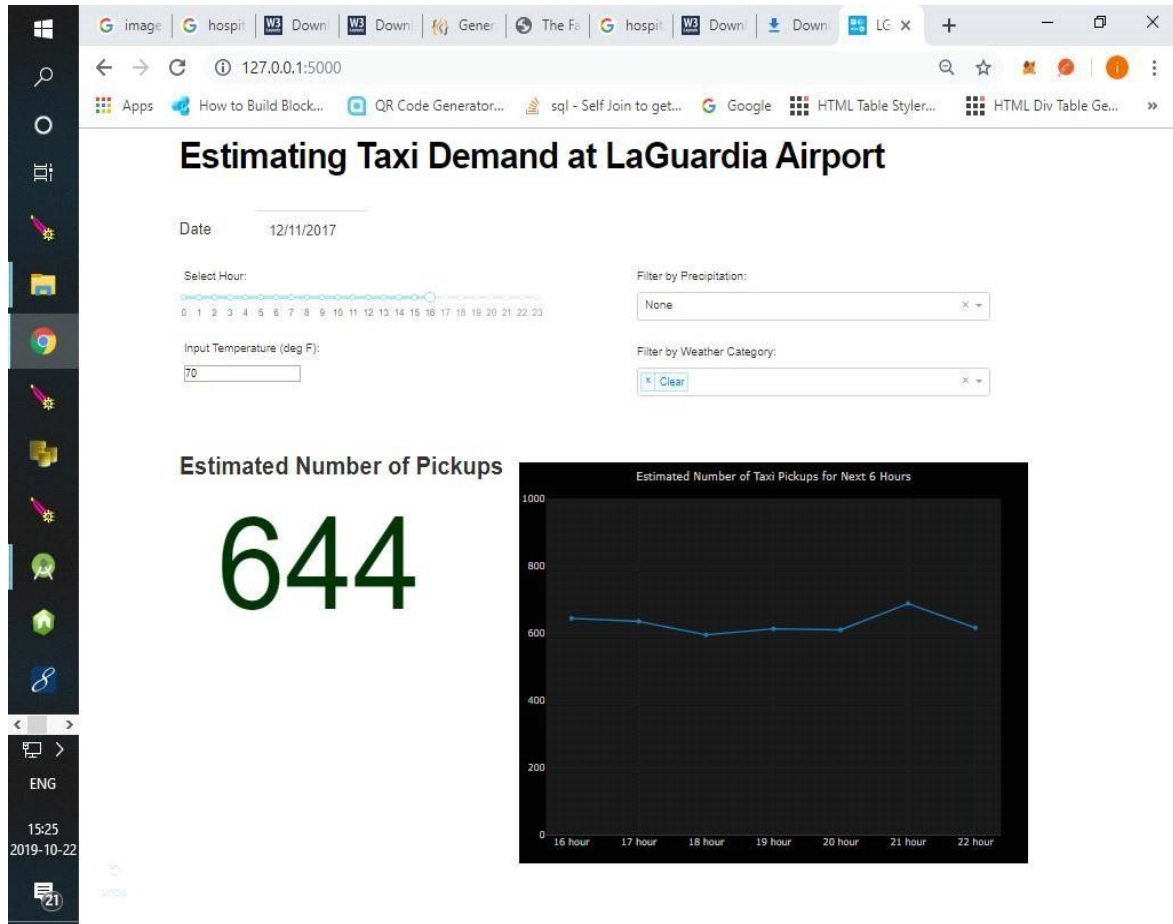
    return date in holidays

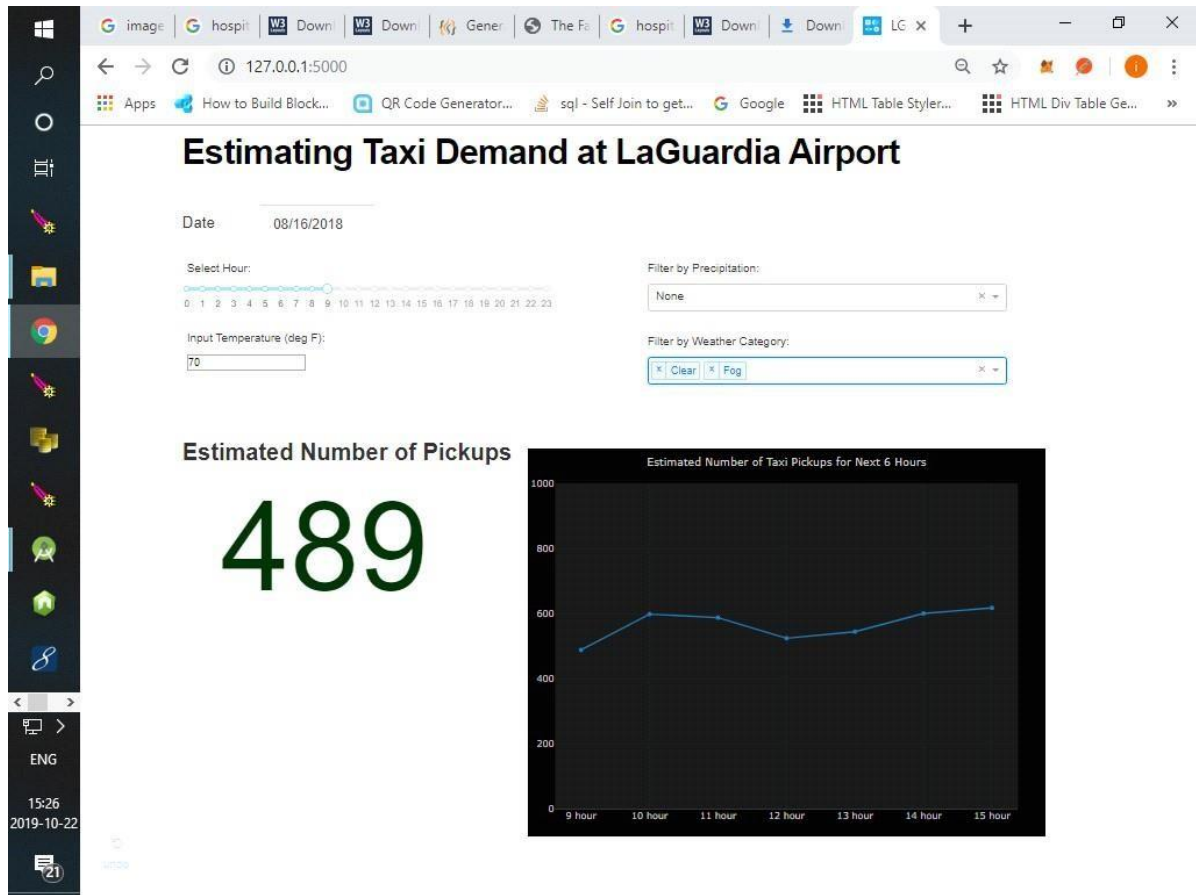
if __name__ == '__main__':

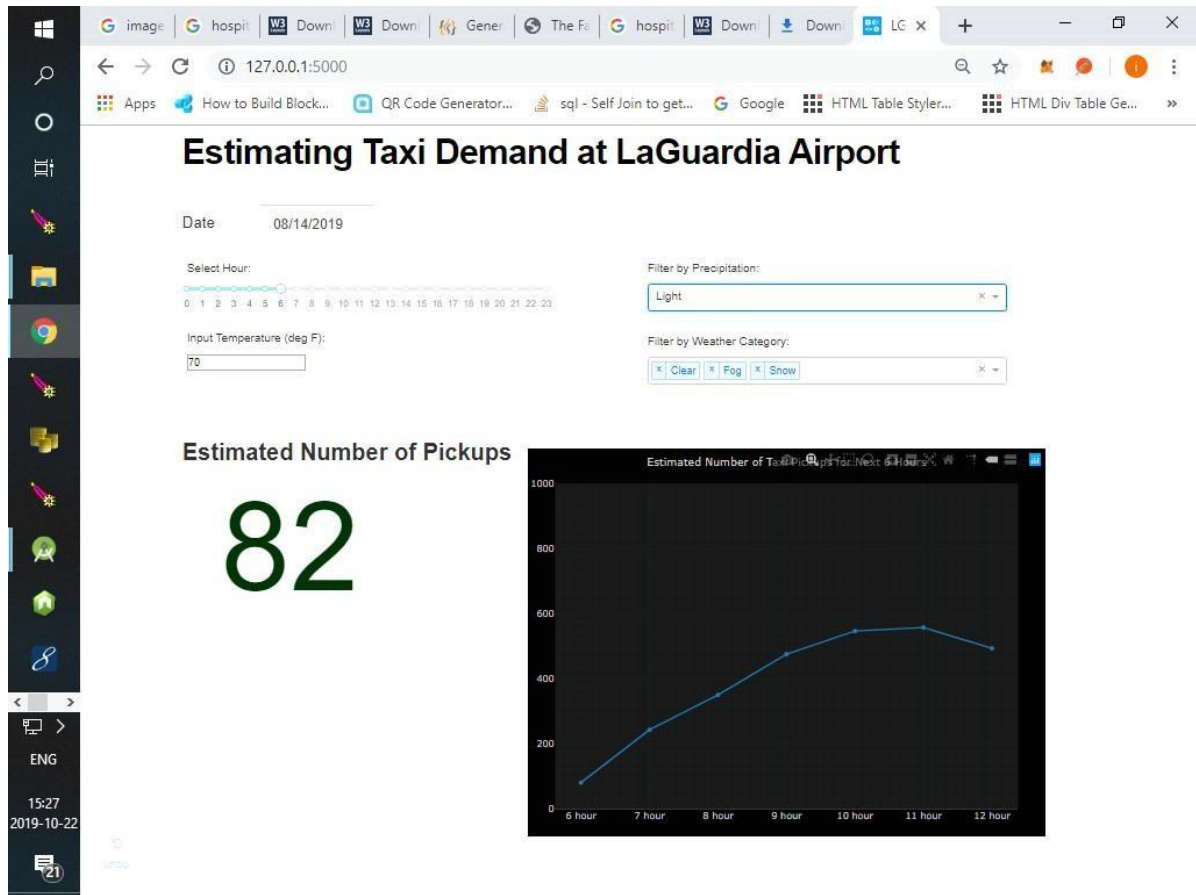
```

```
app.server.run(debug=True, threaded=True)
```

SCREENSHOTS:







CONCLUSION:

Earlier, I mentioned the remarkable results people are achieving with RNNs. Essentially all of these are achieved using LSTMs. They really work a lot better for most tasks!

Written down as a set of equations, LSTMs look pretty intimidating. Hopefully, walking through them step by step in this essay has made them a bit more approachable.

LSTMs were a big step in what we can accomplish with RNNs. It's natural to wonder: is there another big step? A common opinion among researchers is: "Yes! There is a next step and its attention!" The idea is to let every step of RNN pick information to look at from some larger collection of information.

For example, if you are using an RNN to create a caption describing an image, it might pick a part of the image to look at for every word it outputs. In fact, Xu, *et al.* (2015) do exactly this – it might be a fun starting point if you want to explore attention! There's been a number of really exciting results using attention and it seems like a lot more are around the corner.

REFERENCES:

- [1] N. J. Yuan, Y. Zheng, L. Zhang, and X. Xie, "T-finder: A recommender system for finding passengers and vacant taxis," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 10, pp. 2390–2403, Oct. 2013.
- [2] K. T. Seow, N. H. Dang, and D.-H. Lee, "A collaborative multiagent taxi-dispatch system," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 3, pp. 607–616, Jul. 2010.
- [3] P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. H. Strogatz, and C. Ratti, "Quantifying the benefits of vehicle pooling with shareability networks," *Proc. Nat. Acad. Sci. USA*, vol. 111, no. 37, pp. 13290–13294, 2014.
- [4] X. Ma, H. Yu, Y. Wang, and Y. Wang, "Large-scale transportation network congestion evolution prediction using deep learning theory," *PLoS ONE*, vol. 10, no. 3, p. e0119044, 2015.
- [5] K. Zhang, Z. Feng, S. Chen, K. Huang, and G. Wang, "A framework for passengers demand prediction and recommendation," in *Proc. IEEE SCC*, Jun. 2016, pp. 340–347.
- [6] K. Zhao, D. Khryashchev, J. Freire, C. Silva, and H. Vo, "Predicting taxi demand at high spatial resolution: Approaching the limit of predictability," in *Proc. IEEE BigData*, Dec. 2016, pp. 833–842.
- [7] D. Zhang, T. He, S. Lin, S. Munir, and J. A. Stankovic, "Taxi-passenger demand modeling based on big data from a roving sensor network," *IEEE Trans. Big Data*, vol. 3, no. 1, pp. 362–374, Sep. 2017.
- [8] F. Miao *et al.*, "Taxi dispatch with real-time sensing data in metropolitan areas: A receding horizon control approach," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 2, pp. 463–478, Apr. 2016.
- [9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [10] A. Graves. (2013). "Generating sequences with recurrent neural networks."
[Online]. Available: <https://arxiv.org/abs/1308.0850>
- [11] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. NIPS*, Dec. 2014, pp. 3104–3112.

- [12] C. M. Bishop, *Mixture Density Networks*. Birmingham, U.K.: Aston University, 1994.
- [13] L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, and L. Damas, "Predicting taxi-passenger demand using streaming data," *IEEE Trans. Intell. Transp. Syst.*, vol. 14, no. 3, pp. 1393–1402, Sep. 2013.
- [14] N. Davis, G. Raina, and K. Jagannathan, "A multi-level clustering approach for forecasting taxi travel demand," in *Proc. IEEE ITSC*, Dec. 2016, pp. 223–228.
- [15] J. Yuan, Y. Zheng, L. Zhang, X. Xie, and G. Sun, "Where to find my next passenger," in *Proc. ACM UbiComp*, Sep. 2011, pp. 109–118.
- [16] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *Proc. IEEE ICDE*, Apr. 2013, pp. 410–421.
- [17] H. Rong, X. Zhou, C. Yang, Z. Shafiq, and A. Liu, "The rich and the poor: A Markov decision process approach to optimizing taxi driver revenue efficiency," in *Proc. ACM CIKM*, Oct. 2016, pp. 2329–2334.
- [18] J. Azevedo, P. M. d'Orey, and M. Ferreira, "On the mobile intelligence of autonomous vehicles," in *Proc. IEEE NOMS*, Apr. 2016, pp. 1169–1174.
- [19] P. S. Castro, D. Zhang, C. Chen, S. Li, and G. Pan, "From taxi gps traces to social and community dynamics: A survey," *ACM Comput. Surv.*, vol. 46, no. 2, p. 17, 2013.
- [20] A. de Brébisson, É. Simon, A. Auvolet, P. Vincent, and Y. Bengio. (2015). "Artificial neural networks applied to taxi destination prediction." [Online]. Available: <https://arxiv.org/abs/1508.00021>
- [21] R. Rahmatizadeh, P. Abolghasemi, A. Behal, and L. Bölöni. (2016). "Learning real manipulation tasks from virtual demonstrations using LSTM." [Online]. Available: <https://arxiv.org/abs/1603.03833>
- [22] A. Karpathy, J. Johnson, and L. Fei-Fei. (2015). "Visualizing and understanding recurrent networks." [Online]. Available: <https://arxiv.org/abs/1506.02078>
- [23] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE ICASSP*, May 2013, pp. 6645–6649.
- [24] K. Simonyan and A. Zisserman. (2014). "Very deep convolutional networks for large-scale image recognition." [Online]. Available: <https://arxiv.org/abs/1409.1556>

- [25] G. Niemeyer. (2008). *Tips & Tricks About Geohash*. [Online]. Available: <http://geohash.org/site/tips.html>
- [26] NYC Taxi & Limousine Commission. *Taxi and Limousine Commission (TLC) Trip Record Data*. Accessed: Dec. 2016. [Online]. Available: http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml
- [27] G. J. McLachlan and K. E. Basford, *Mixture Models: Inference and Applications to Clustering*, vol. 84. New York, NY, USA: Marcel Dekker, 1988.
- [28] A. van der Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," in *Proc. ICML*, Jun. 2016, pp. 1747–1756.
- [29] H. Larochelle and I. Murray, "The neural autoregressive distribution estimator," in *Proc. AISTATS*, Jun. 2011, pp. 29–37.
- [30] B. van Merriënboer et al. (2015). "Blocks and fuel: Frameworks for deep learning." [Online]. Available: <https://arxiv.org/abs/1506.00619>
- [31] R. Al-Rfou et al. (2016). "Theano: A python framework for fast computation of mathematical expressions." [Online]. Available: <https://arxiv.org/abs/1605.02688>
- [32] P. Lopez-Garcia, E. Onieva, E. Osaba, A. D. Masegosa, and A. Perallos, "A hybrid method for short-term traffic congestion forecasting using genetic algorithms and cross entropy," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 2, pp. 557–569, Feb. 2016.
- [33] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang, "Traffic flow prediction with big data: A deep learning approach," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 2, pp. 865–873, Apr. 2015.
- [34] M. Yang, Y. Liu, and Z. You, "The reliability of travel time forecasting," *IEEE Trans. Intell. Transp. Syst.*, vol. 11, no. 1, pp. 162–171, Mar. 2010