Fault Identification And Diagnosis Of Large Virtualized Engineering Computing Infrastructure Using Tensorflow Deep Machine Learning

Submitted By:

ADITYA PAL
B.E (4th Sem, CSE)
BIT Mesra, Ranchi

Certificate of Approval

*This is to certify that **ADITYA PAL**, a student of 4th semester B.E enrolled in C.S.E. Branch of **BIT Mesra, Ranchi** has worked under my supervision and guidance from **9th JUNE 2017** – **9th JULY 2017** and has completed the project for **"Fault Identification and diagnosis of large virtualized engineering using tensorflow deep machine learning (classification)"** successfully. We appreciate his enthusiasm and commitment during the project tenure and wish his success in all his future endeavours.*

Dr. Pralay Pal *(Senior Program Manager)*	Mr. Kuntal Chowdhury *(Manager, Human Resource)*	Mr. Bidhayak Majumdar *(Senior Team Lead)*

Acknowledgment

It has been a great honour and privilege to undergo training at Tata Technologies Ltd., Jamshedpur. I have been able to complete this project only due to the support and guidance of many individuals. However, it would be wrong on my part to not express my sincere thanks to all of them.

I would like to take an opportunity to express my humble gratitude to Dr. Pralay Pal and Mr. Bidhayak Majumdar who has helped me execute this project. His constant guidance and willingness to share his vast knowledge made me understand this project and its manifestations in great depth. I am highly obliged to them without whose support this work would not have been accomplished. Their invaluable guidance helped me understand the project better.

I am grateful to Mr. Kuntal Chowdhury, the Human resources manager for providing all the facilities to meet my project requirements and giving me a chance to work with this organization.

ADITYA PAL

TABLE OF CONTENTS

CHAPTER NO	***PARTICULARS***	***PAGE N0***
1	Introduction	5
2	Problem Statement	8
3	Requirements	10
4	Theory	14
5	Classifier Implementation	25
6	Neural Net implementation	30
7	Conclusion	37
8	References	38

Introduction

Containers -

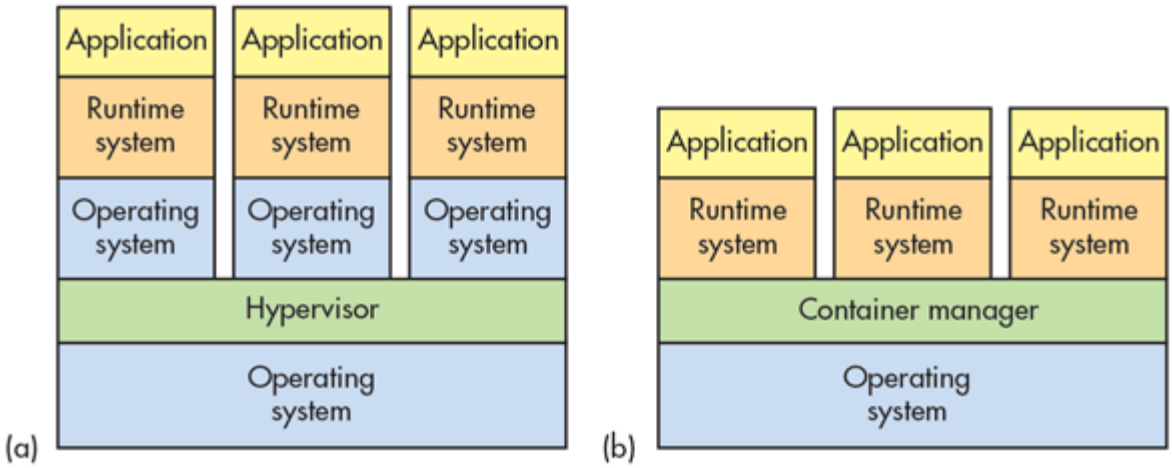
Containers are a method of operating system virtualization that allow you to run an application and its dependencies in resource-isolated processes. Containers allow you to easily package an application's code, configurations, and dependencies into easy to use building blocks that deliver environmental consistency, operational efficiency, developer productivity, and version control. Containers can help ensure that applications deploy quickly, reliably, and consistently regardless of deployment environment. Containers also give you more granular control over resources giving your infrastructure improved efficiency.

Virtual Machines -

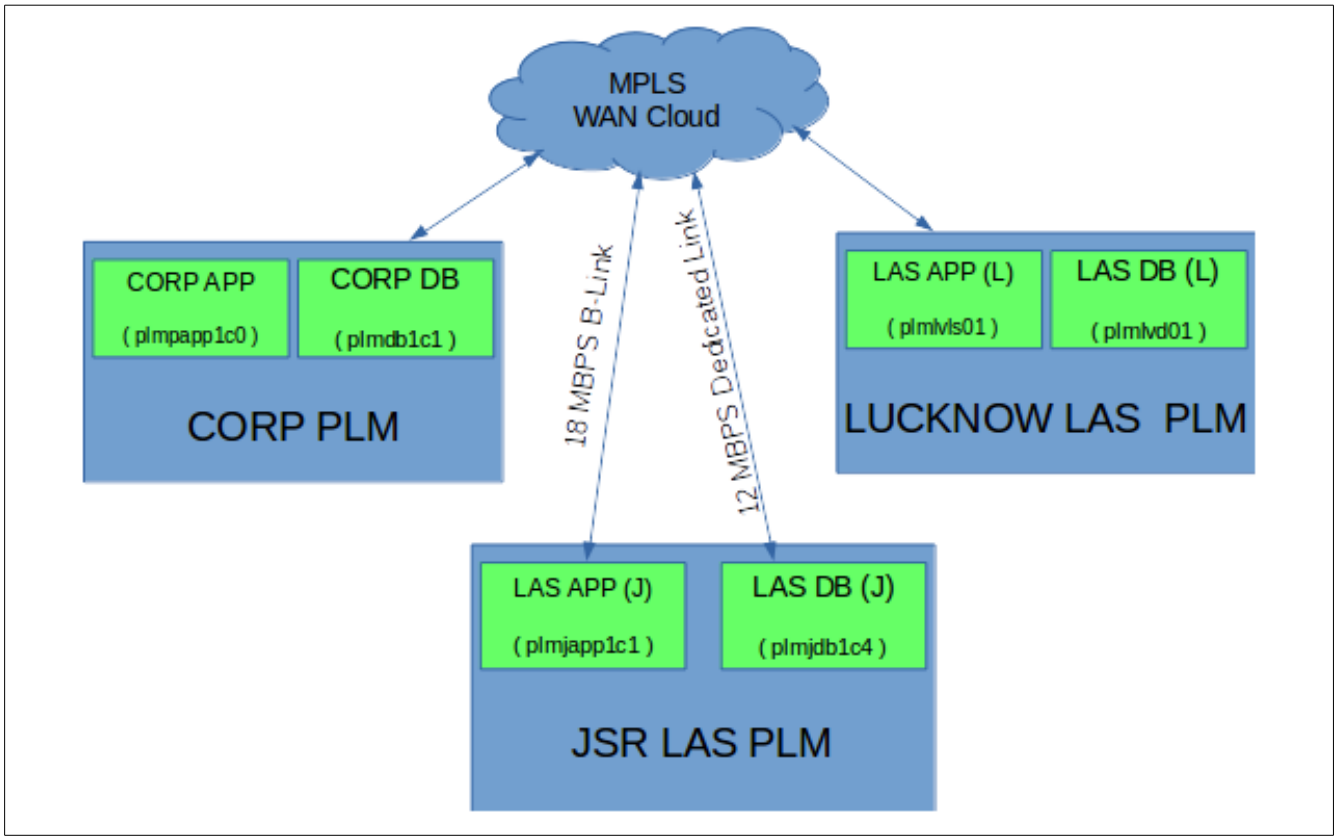
A virtual machine is a computer file, typically called an image, which behaves like an actual computer. In other words, creating a computer within a computer. It runs in a window, much like any other programme, giving the end user the same experience on a virtual machine as they would have on the host operating system itself. The virtual machine is sandboxed from the rest of the system, meaning that the software inside a virtual machine cannot escape or tamper with the computer itself. This produces an ideal environment for testing other operating systems including beta releases, accessing virus-infected data, creating operating system backups and running software or applications on operating systems for which they were not originally intended.

Multiple virtual machines can run simultaneously on the same physical computer. For servers, the multiple operating systems run side-by-side with a piece of software called a hypervisor to manage them, while desktop computers typically employ one operating system to run the other operating systems within its programme windows. Each virtual machine provides its own virtual hardware, including CPUs, memory, hard drives, network interfaces and other devices. The virtual hardware is then mapped to the real hardware on the physical machine which saves costs by reducing the need for physical hardware systems along with the associated maintenance costs that go with it, plus reduces power and cooling demand.

Virtual



machines (VM) are managed by a hypervisor and utilize VM hardware (a), while container systems provide operating system services from the underlying host and isolate the applications using virtual-memory hardware (b).



Cloud Architecture of MPLS Cloud

Machine Learning

Machine learning is the subfield of computer science that, according to Arthur Samuel in 1959, gives "computers the ability to learn without being explicitly programmed." Evolved from the study of pattern recognition and computational learning theory in

artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions,:2 through building a model from sample inputs. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or infeasible; example applications include email filtering, detection of network intruders or malicious insiders working towards a data breach, optical character recognition (OCR), learning to rank, and computer vision.

Machine learning is closely related to (and often overlaps with) computational statistics, which also focuses on prediction-making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field. Machine learning is sometimes conflated with data mining, where the latter subfield focuses more on exploratory data analysis and is known as unsupervised learning.:vii Machine learning can also be unsupervised and be used to learn and establish baseline behavioral profiles for various entities and then used to find meaningful anomalies.

Deep learning

Deep learning (also known as deep structured learning or hierarchical learning) is the application to learning tasks of artificial neural networks (ANNs) that contain more than one hidden layers. Deep learning is part of a broader family of machine learning methods based on learning data representations, as opposed to task specific algorithms. Learning can be supervised, partially supervised or unsupervised.

Some representations are loosely based on interpretation of information processing and communication patterns in a biological nervous system, such as neural coding that attempts to define a relationship between various stimuli and associated neuronal responses in the brain. Research attempts to create efficient systems to learn these representations from large-scale, unlabeled data sets.

Deep learning architectures such as deep neural networks, deep belief networks and recurrent neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation and bioinformatics where they produced results comparable to and in some cases superior to human experts.

Problem Statement

In Tata Technologies, Product Life Cycle Management (PLM) applications are hosted on a private cloud server which contains several virtual machines and containers. Different application and database servers are present in different locations in India such as Jamshedpur, Pune, Lucknow, Hinjawadi, etc. These servers control the working of PLM client applications. The working of these servers are dependent on multiple parameters such as Ping Response, TNS Ping, CPU Load Average, CPU utilization, RAM utilization, etc. The servers located at different locations communicate through network links dependent on the mentioned parameters. If any of the parameter values goes beyond a certified range, the working of the servers as well as the applications is affected. The diagnosis of the occurred event is done through manual process. The manual process takes up time and is subject to inaccuracy.

The above system changes give rise to several different type of faults that can slow down the normal execution of the system. Some of the faults that occur commonly are :

- Remote Application Server Down
- Remote Database Server Down
- Local Database Server Down
- Local Application Container Hang
- Remote Application Container Hang

Objectives:

Our project focusses on automating the process of detection and diagnosis of the faults that occur commonly. We have used the approach of machine learning and deep learning for classifying different categories of faults according to their parametric values.

Benefits:

Basic benefit to this project will be the amount of time and effort saved in the process of detecting faults through manual steps. Also the faults can be detected with more accuracy. Our project will also aim to reduce the operational cost of the process.

	JSR	PUNE	LKN	JSR-PUNE	JSR-LKN
Tns Ping	✗	✗	✗	✓	✓
Ping Response	✗	✗	✗	✓	✓
Load Average	✓	✓	✓	✗	✗
CPU Utilization	✓	✓	✓	✗	✗

Figure 1 : Local and Wide Area Based Monitoring Parameters

Requirements

Dataset :

The dataset that is fed into the model records parametric values every minute. This dataset consists of eight parameters listed as (in order) :

- Ping Local
- Ping Local Application-Corporate Application
- Ping Local Application-Corporate DataBase
- Load Average Local Application
- Load Average Local DataBase
- Load Average Corporate Application
- TNS ping Local Application-Local DataBase
- TNS ping Local Application-Corporate DataBase

The first five days data is used as training set to train the design model which is then implemented on test set. The test set contains two days data.

	Ping Local	Ping LocalApp-CorpApp	Ping LocalApp-CorpDB	Load Average LocalApp	Load Average LocalDB	Load Average CORPApp	TNS ping LocalApp-LocalDB	TNS ping LocalApp-CorpDB
0	0.412	68.3	55.2	0.21	0.03	0.39	0	120
1	0.474	67.4	54.9	0.22	0.09	0.92	0	120
2	0.422	66.7	55.4	0.47	0.02	0.81	0	130
3	0.586	66.4	55.3	0.88	0.20	0.47	0	130
4	0.489	74.4	55.2	0.30	0.12	0.46	0	130
5	0.383	67.4	56.5	1.26	0.12	0.55	0	120
6	0.434	66.7	56.4	0.20	0.05	0.24	0	130
7	0.304	65.1	56.9	0.53	0.03	1.98	0	120
8	0.363	66.1	56.6	0.58	0.09	0.64	0	130
9	0.398	65.6	56.1	0.25	0.08	0.47	0	120
10	0.454	65.5	56.2	0.19	0.02	0.30	0	120
11	0.405	65.9	56.6	0.72	0.24	0.46	0	120
12	0.359	66.4	56.8	0.34	0.02	0.44	10	120
13	0.330	65.2	59.2	0.57	0.14	0.82	0	130

Dataset Used

Softwares Used

Programming Language : Python 3.6

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale.
Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library.

IPython

IPython is a command shell for interactive computing in multiple programming languages, originally developed for the Python programming language, that offers introspection, rich media, shell syntax, tab completion, and history. IPython provides the following features:

- Interactive shells (terminal and Qt-based).
- A browser-based notebook with support for code, text, mathematical expressions, inline plots and other media.
- Support for interactive data visualization and use of GUI toolkits.
- Flexible, embeddable interpreters to load into one's own projects.
- Tools for parallel computing.

> In 2014, Fernando Pérez announced a spin-off project from IPython called Project Jupyter. IPython will continue to exist as a Python shell and a kernel for Jupyter, while the notebook and other language-agnostic parts of IPython will move under the Jupyter name.

Atom IDE

Atom is a free and open-source text and source code editor for macOS, Linux, and Microsoft Windows with support for plug-ins written in Node.js, and embedded Git Control, developed by GitHub. Atom is a desktop application built using web technologies. Most of the extending packages have free software licenses and are community-built and maintained. Atom is based on Electron (formerly known as Atom Shell), a framework that enables cross-platform desktop applications using Chromium and Node.js. It is written in CoffeeScript and Less. It can also be used as an integrated development environment (IDE). Atom was released from beta, as version 1.0, on June 25, 2015. Its developers call it a "hackable text editor for the 21st Century".

Libraries Used

TensorFlow:

TensorFlow is an open source software library for machine learning across a range of tasks, and developed by Google to meet their needs for systems capable of building and training neural networks to detect and decipher patterns and correlations, analogous to the learning and reasoning which humans use. It is currently used for both research and production at Google products, often replacing the role of its closed-source predecessor, DistBelief. TensorFlow was originally developed by the Google Brain team for internal Google use before being released under the Apache 2.0 open source license on November 9, 2015.

Scikit-learn:

Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Pandas:

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. Pandas is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for multidimensional structured data sets.

NumPy:

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

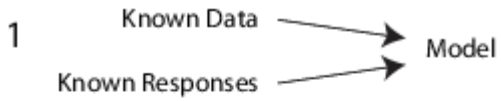
Theory

Supervised Machine Learning

It is the machine learning task of inferring a function from *labeled training data*. The training data consist of a set of *training examples*. In supervised learning, each example is a *pair* consisting of an input object (typically a vector) and a desired output value. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a reasonable.

The aim of supervised, machine learning is to build a model that makes predictions based on evidence in the presence of uncertainty. As adaptive algorithms identify patterns in data, a computer "learns" from the observations. When exposed to more observations, the computer improves its predictive performance.

Specifically, a supervised learning algorithm takes a known set of input data and known responses to the data (output), and *trains* a model to generate reasonable predictions for the response to new data.



For example, suppose you want to predict whether someone will have a heart attack within a year. You have a set of data on previous patients, including age, weight, height, blood pressure, etc. You know whether the previous patients had heart attacks within a year of their measurements. So,

the problem is combining all the existing data into a model that can predict whether a new person will have a heart attack within a year.

You can think of the entire set of input data as a heterogeneous matrix. Rows of the matrix are called *observation and* each contain a set of measurements for a subject (patients in the example). Columns of the matrix are called *predictors* and each are variables representing a measurement taken on every subject (age, weight, height, etc. in the example). You can think of the response data as a column vector where each row contains the output of the corresponding data (whether patient had a heart attack).

Supervised learning splits into two broad categories: classification and regression.

- In *classification*, the goal is to assign a class (or *label*) from a finite set of classes to an observation. That is, responses are categorical variables. Applications include spam filters, advertisement recommendation systems, and image and speech recognition. Predicting whether a patient will have a heart attack within a year is a classification problem, and the possible classes are true and false. Classification algorithms usually apply to nominal response values. However, some algorithms can accommodate ordinal classes.

- In *regression*, the goal is to predict a continuous measurement for an observation. That is, the responses variables are real numbers. Applications include forecasting stock prices, energy consumption, or disease incidence.

Unsupervised Machine Learning

It is the machine learning task of inferring a function to describe hidden structure from "unlabeled" data (a classification or categorization is not included in the observations). Since the examples given to the learner are unlabeled, there is no evaluation of the accuracy of the structure that is output by the relevant algorithm—which is one way of distinguishing unsupervised learning from supervised learning.

Approaches to unsupervised learning include:

- Clustering
- Anomaly Detection
- Neural Networks

Classifiers

A classifier sometimes abbreviated as “clf”, called a counter word. It is to illustrate the nature of decision boundaries on dataset of different classifiers particularly in high-dimensional spaces, data can more easily be separated linearly and the simplicity of classifiers such as Naive Bayes and linear SVMs might lead to better generalization than is achieved by other classifiers.

The classical example of unsupervised learning in the study of both natural and artificial neural networks is subsumed by Donald Hebb’s principle, that is, neurons that fire together wire together. In Hebbian learning the connection is reinforced irrespective of an error, but is exclusively a function of the coincidence between action potentials between the two neurons.

In machine learning and statistics classification is the problem of identifying to which set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. An example would be assigning a given email into "spam" or "non-spam" classes or assigning a diagnosis to a given patient as described by observed characteristics of the patient (gender, blood pressure, presence or absence of certain symptoms, etc.). Classification is an example of pattern recognition.

In the terminology of machine learning, classification is considered an instance of supervised learning, i.e. learning where a training set of correctly identified observations is available. The corresponding unsupervised procedure is known as clustering, and involves grouping data into categories based on some measure of inherent similarity or distance.

An algorithm that implements classification, especially in a concrete implementation, is known as a classifier. The term "classifier" sometimes also refers to the mathematical function, implemented by a classification algorithm that maps input data to a category.

Classification and clustering are examples of the more general problem of pattern recognition, which is the assignment of some sort of output value to a given input value. Other examples are regression, which assigns a real-valued output to each input; sequence labeling, which assigns a class to each member of a sequence of values.

A common subclass of classification is probabilistic classification. Algorithms of this nature use statistical inference to find the best class for a given instance. Unlike other algorithms, which simply output a "best" class, probabilistic algorithms output a probability of the instance being a member of each of the possible classes. The best class is normally then selected as the one with the highest probability. However, such an algorithm has numerous advantages over non-probabilistic classifiers.

Neural Networks and Deep Learning

- Neural networks, a beautiful biologically-inspired programming paradigm which enables a computer to learn from observational data.
- Deep learning, a powerful set of techniques for learning in neural networks.

Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition and natural language processing.

Artificial neural networks (ANNs)

Artificial neural networks (ANNs) are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn to do tasks by considering examples, generally without task-specific

programming. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the analytic results to identify cats in other images. They have found most use in applications difficult to express in a traditional computer algorithm using rule-based programming.

Typically, neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, possibly after traversing the layers multiple times .The original goal of the neural network approach was to solve problems in the same way that a human brain would. Over time, attention focused on matching specific mental abilities, leading to deviations from biology such as back propagation or passing information in the reverse direction and adjusting the network to reflect that information.

Neural networks have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games, medical diagnosis and in many other domains.

As of 2017, neural networks typically have a few thousand to a few million units and millions of connections. Their computing power is similar to a worm brain several orders of magnitude simpler than a human brain. Despite this, they can perform functions (e.g., playing chess) that are far beyond a worm's capacity.

Why use neural networks?

Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer "what if" questions.
Other advantages include:

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.

2. Self-Organisation: An ANN can create its own organisation or representation of the information it receives during learning time.

3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.

4. Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

Neural networks -- also called artificial neural networks -- are a variety of deep learning technologies. Commercial applications of these technologies generally focus on solving complex signal processing or pattern recognition problems.

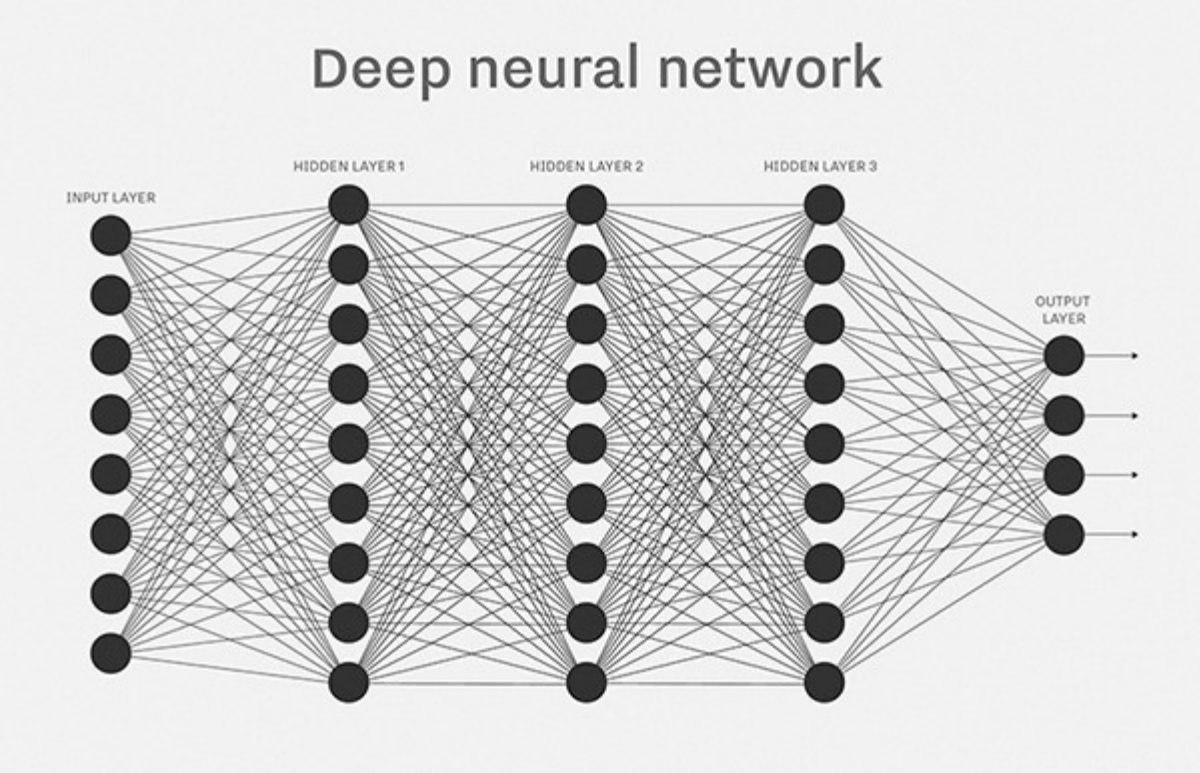
A neural network usually involves a large number of processors operating in parallel and arranged in tiers .The first tier receives the raw input information -- analogous to optic nerves in human visual processing. Each successive tier receives the output from the tier preceding it, rather than from the raw input -- in the same way neurons further from the optic nerve receive signals from those closer to it. The last tier produces the output of the system.

Each processing node has its own small sphere of knowledge, including what it has seen and any rules it was originally programmed with or developed for itself. The tiers are highly interconnected, which means each node in tier *n* will be connected to many nodes in tier *n-1* -- its inputs -- and in tier *n+1,* which provides input for those nodes. There may be one or multiple nodes in the output layer, from which the answer it produces can be read.

Neural networks are notable for being adaptive, which means they modify themselves as they learn from initial training and subsequent runs provide more information about the world. The most basic learning model is

centered on weighting the input streams, which is how each node weights the importance of input from each of its predecessors. Inputs that contribute to getting right answers are weighted higher.

Typically, a neural network is initially trained or fed large amounts of data. Training consists of providing input and telling the network what the output should be. For example, to build a network to identify the faces of actors, initial training might be a series of pictures of actors, non-actors, masks, statuary, animal faces and so on. Each input is accompanied by the matching identification, such as actors' names, "not actor" or "not human" information. Providing the answers allows the model to adjust its internal weightings to learn how to do its job better.



Neural networks are sometimes described in terms of their depth, including how many layers they have between input and output, or the model's so-called hidden layers. They can also be described by the number of hidden nodes the model has or in terms of how many inputs and outputs each node has. Variations on the classic neural-network design allow various forms of forward and backward propagation of information among tiers.

Artificial neural networks were first created as part of the broader research effort around artificial intelligence and they continue to be important in that space, as well as in research around human recognition and consciousness.

Backpropagation and Forward Propagation

It is a method to calculate the gradient of the loss function with respect to the weights in an artificial neural network. It is commonly used as a part of algorithms that optimize the performance of the network by adjusting the weight.

The optimization algorithm repeats a two phase cycle, propagation and weight update. When an input vector is presented to the network, it is propagated forward through the network, layer by layer, until it reaches the output layer. The output of the network is then compared to the desired output, using a loss function and an error value is calculated for each of the neurons in the output layer. The error values are then propagated backwards, starting from the output, until each neuron has an associated error value which roughly represents its contribution to the original output.

Backpropagation uses these error values to calculate the gradient of the loss function. In the second phase, this gradient is fed to the optimization method, which in turn uses it to update the weights, in an attempt to minimize the loss function.

The importance of this process is that, as the network is trained, the neurons in the intermediate layers organize themselves in such a way that the different neurons learn to recognize different characteristics of the total input space. After training, when an arbitrary input pattern is present which contains noise or is incomplete, neurons in the hidden layer of the network will respond with an active output if the new input contains a pattern that resembles a feature that the individual neurons have learned to recognize during their training. Other typical problems of the back-propagation algorithm are the speed of convergence and the possibility of ending up in a local minimum of the error function. Today there are practical methods that

make back-propagation in multi-layer perceptrons the tool of choice for many machine learning tasks.

A feedforward neural network is an artificial neural network wherein connections between the units do *not* form a cycle. The feedforward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

Single Layer Perceptron

The simplest kind of neural network is a *single-layer perceptron* network, which consists of a single layer of output nodes; the inputs are fed directly to the outputs via a series of weights. In this way it can be considered the simplest kind of feed-forward network. The sum of the products of the weights and the inputs is calculated in each node, and if the value is above some threshold (typically 0) the neuron fires and takes the activated value (typically 1); otherwise it takes the deactivated value (typically -1). Neurons with this kind of activation function are also called *artificial neurons*. A perceptron can be created using any values for the activated and deactivated states as long as the threshold value lies between the two.

Perceptron can be trained by a simple learning algorithm that is usually called the *delta rule*. It calculates the errors between calculated output and sample output data, and uses this to create an adjustment to the weights, thus implementing a form of gradient descent.

Although a single threshold unit is quite limited in its computational power, it has been shown that networks of parallel threshold units can approximate any continuous function from a compact interval of the real numbers into the interval [-1,1]. With this choice, the single-layer network is identical to the logistic regression model, widely used in statistical modeling. The logistic function is also known as the sigmoid function. It has a continuous derivative, which allows it to be used in back propagation. This function is also preferred because its derivative is easily calculated.

Multi Layer Perceptron

A two-layer neural network capable of calculating XOR .The numbers within the neurons represent each neuron's explicit threshold (which can be factored out so that all neurons have the same threshold, usually the numbers that annotate arrows represent the weight of the inputs. This net assumes that if the threshold is not reached, zero (not -1) is output. Note that the bottom layer of inputs is not always considered a real neural network layer .This class of networks consists of multiple layers of computational units, usually interconnected in a feed-forward way. Each neuron in one layer has directed connections to the neurons of the subsequent layer. In many applications the units of these networks apply a *sigmoid function* as an activation function.

The *universal approximation theorem* for neural networks states that every continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated arbitrarily closely by a multi-layer perceptron with just one hidden layer. This result holds for a wide range of activation functions, e.g. for the sigmoidal functions. Multi-layer networks use a variety of learning techniques, the most popular being *back-propagation*. Here, the output values are compared with the correct answer to compute the value of some predefined error-function. By various techniques, the error is then fed back through the network. Using this information, the algorithm adjusts the weights of each connection in order to reduce the value of the error function by some small amount. After repeating this process for a sufficiently large number of training cycles, the network will usually converge to some state where the error of the calculations is small. In this case, one would say that the network has *learned* a certain target function. To adjust weights properly, one applies a general method for non-linear optimization that is called gradient descent. For this, the network calculates the derivative of the error function with respect to the network weights, and changes the weights such that the error decreases (thus going downhill on the surface of the error function). For this reason, back-propagation can only be applied on networks with differentiable activation functions.

Learning Rate

The learning rate is a common parameter in many of the learning algorithms, and affects the speed at which the Artificial Neural Network arrives at the minimum solution. In back propagation, the learning rate is analogous to the step-size parameter from the gradient-descent algorithm. If the step-size is too high, the system will either oscillate about the true solution, or it will diverge completely. If the step-size is too low, the system will take a long time to converge on the final solution. Basically it is the training parameter that controls the size of weight and bias changes in learning of the training algorithm.

Learning rate is a decreasing function of time. Two forms that are commonly used are a linear function of time and a function that is inversely proportional to the time *t*. These are illustrated in the Figure 2.7. Linear alpha function (a) decreases to zero linearly during the learning from its initial value whereas the inverse alpha function (b) decreases rapidly from the initial value Both the functions in the Figure have the initial value of 0.9. The initial values for a must be determined. Usually, when using a rapidly decreasing inverse alpha function, the initial values can be larger than in the linear case. The learning is usually performed in two phases. On the first round relatively large initial alpha values are used (a = 0.3 … 0.99) whereas small initial alpha values (a = 0.01 … a = 0.1) are used during the other round.

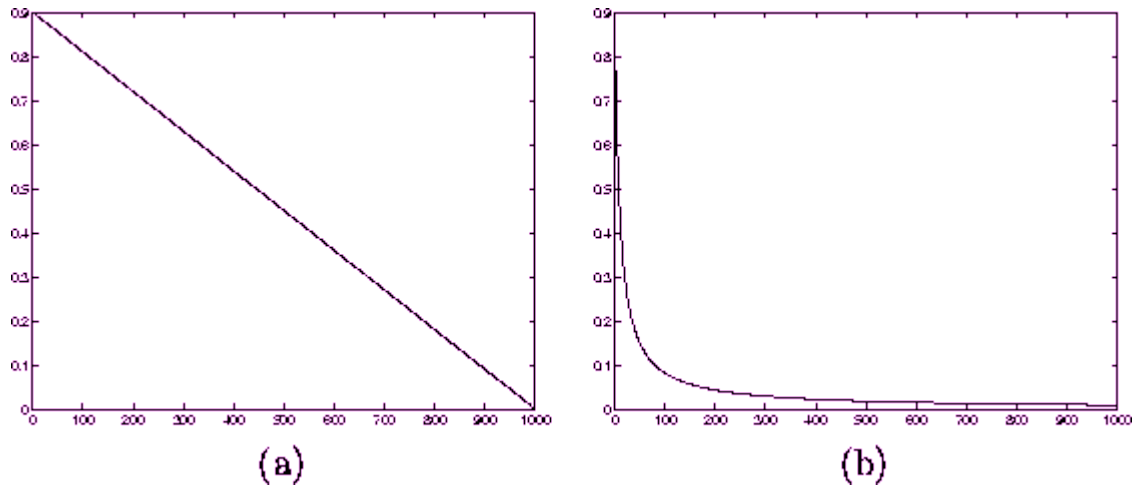


Figure 2.7: Learning rates as functions of time

Solution Implementation Using Classifier

Classifier Used :
Random Forest Classifier (RF)

Importing tools Needed

```
from sklearn.model_selection import train_test_split as tt
import numpy as np
import pandas as pd
import csv
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score as acc
```

- **Scikit-learn:**

Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Scikit-learn Functions used:

RandomForestClassifier:
Random Forest is a trademark term for an ensemble of decision trees. In Random Forest, we've collection of decision trees (so known as "Forest"). To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Each tree is planted & grown as follows:

1. If the number of cases in the training set is N, then sample of N cases is taken at random but *with replacement*. This sample will be the training set for growing the tree.

2. If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. There is no pruning.

accuracy_score:

In multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must *exactly* match the corresponding set of labels in y_true.

train_test_split:

The data we use is usually split into training data and test data. The training set contains a known output and the model learns on this data in order to be generalized to other data later on. We have the test dataset (or subset) in order to test our model's prediction on this subset. This can be done easily using scikit-learn's train_test_split modeule.

- **NumPy:**

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

- **Pandas:**

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures

and operations for manipulating numerical tables and time series. Pandas is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for multidimensional structured data sets.

- **CSV:**

The so-called CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases. The csv module implements classes to read and write tabular data in CSV format. It allows programmers to say, “write this data in the format preferred by Excel,” or “read data from this file which was generated by Excel,” without knowing the precise details of the CSV format used by Excel. Programmers can also describe the CSV formats understood by other applications or define their own special-purpose CSV formats.

Data Preprocessing:

```
file = open("dummy.csv", "r")
file1 = open("data1.csv", "w")
file1.write('"Ping Local" "Ping LocalApp-CorpApp" "Ping LocalApp-CorpDB" "Load Average LocalApp" "Load Average LocalDB" "Lo
for line in file:
    l = len(line)
    dat = list(line.split())
    if (dat[1] == 'x' or dat[5] == 'x'):
        line = ""
        for da in dat:
            if (da == 'x'):
                da = '9999999999'
            line+=da+" "
        line += "1\n"
    elif (dat[2] == 'x' or dat[7] == 'x'):
        line = ""
        for da in dat:
            if (da == 'x'):
                da = '9999999999'
            line+=da+" "
        line += "2\n"
    elif (dat[3] == 'x' or dat[4] == 'x' or dat[6] == 'x'):
        line = ""
        for da in dat:
            if (da == 'x'):
                da = '9999999999'
            line+=da+" "
        line += "3\n"
    else:
        line = line[:-1] + " 0" + line[-1]
    file1.write(line)
file.close()
file1.close()
```

First we open the input file ‘dummy.csv’ to read. Then we add the parameter headers to the output csv file ‘data1.csv’. After that we process the input to check what type of fault might be present in the system based on the input data row. The input is then classified into the four types of category faults and the fault type is added as a new column:

- Remote Application Down – Fault Class 1
- Remote DataBase Down – Fault Class 2
- Local DataBase Container Hang – Fault Class 3
- No fault – Class 0

```
data = pd.read_csv("data1.csv",delimiter = ' ')
```

Then we read the csv data in a pandas dataframe named data.

Training The Classifier:

```
x = data.ix[:,:8]
y = data.ix[:,8:]
x_train,x_test,y_train,y_test = tt(x,y,test_size=0.4,random_state=1)
y_train
```

Then
we split the data into two sets – a train set and a test set. The train set consists 60% of the data while the test set consists 40% of the data.

```
clf = RandomForestClassifier(n_estimators = 500)
clf.fit(x_train,y_train)
```

Then we initialize a Random Forest Classifier with number of estimators 500. After running the initialization, we try to fit the training input data with the training target. The best function is then fitted in the variable clf.

Implementing The Trained Model on Test Set

```
result = clf.predict(x_test)
prob = clf.predict_proba(x_test)
print (result)
print (str((acc(y_test,result)*100).astype(int),) + "% accurate")
np.savetxt("output.csv", result , delimiter=" ",fmt= "%0.0f")
print (prob)
```

The trained model is then implemented on the test set. The result is predicted in two forms – one is a matrix consisting of four columns which state the probability of each class for all the inputs. The other output gives the class which has the maximum probability, which is our required prediction.
The output is then saved in a file “output.csv” and the accuracy of the prediction is measured with the original target of the test set.

Result:
The test results generated had a 100% match with the test target given as input.

```
100% accurate


[[ 0.192  0.     0.11   0.698]
 [ 1.     0.     0.     0.   ]
 [ 0.     0.     1.     0.   ]
 ...,
 [ 0.99   0.01   0.     0.   ]
 [ 0.     0.     1.     0.   ]
 [ 0.98   0.018  0.002  0.   ]]
```

Neural Network Implementation

The above problem statement is adressed again through the implementation of neural networks.

Importing the tools needed:

```
import pandas as pd
import numpy as np
import tensorflow as tf
%matplotlib inline
```

The above modules are imported as needed.

Data Preprocessing:

```
f = raw_input()
file = open(f, "r")
file1 = open("data_nn.csv", "w")
file1.write('"Time" "Ping Local" "Ping LocalApp-CorpApp" "Ping LocalApp-CorpDB" "Load Average LocalApp"
             "Load Average LocalDB" "Load Average CORPApp" "TNS ping LocalApp-LocalDB" "TNS ping LocalApp-CorpDB"
            "Class0" "Class1" "Class2" "Class3"\n')
hr = 12
m = 0
for line in file:
    l = len(line)
    dat = list(line.split())
    if (dat[1] == 'x' or dat[5] == 'x'):
        line = str(hr)+":"+str(m)+" "
        for da in dat:
            if (da == 'x'):
                da = '9999999999'
            line+=da+" "
        line += "0 1 0 0\n"
    elif (dat[2] == 'x' or dat[7] == 'x'):
        line = str(hr)+":"+str(m)+" "
        for da in dat:
            if (da == 'x'):
                da = '9999999999'
            line+=da+" "
        line += "0 0 1 0\n"
    elif (dat[3] == 'x' or dat[4] == 'x' or dat[6] == 'x'):
        line = str(hr)+":"+str(m)+" "
        for da in dat:
            if (da == 'x'):
                da = '9999999999'
            line+=da+" "
        line += "0 0 0 1\n"
    else:
        line = str(hr)+":"+str(m)+" "
        for da in dat:
            if (da == 'x'):
                da = '9999999999'
            line+=da+" "
        line += "1 0 0 0\n"
    file1.write(line)
    m=m+1
    if(m==60):
       m=0
       hr=hr+1
file.close()
```

Training Set is then labelled using file manipulation techniques in Python to add labels/classes to the data set according to the diagnosis conditions. This manipulation adds four columns to our dataset which represents the class of the fault to which the instance of the data belongs.

Neural Net Architecture:

```
#computation graph
x = tf.placeholder(tf.float32,[None,8])
#weights
W = tf.Variable(tf.random_normal([8,4]))

#biases
b = tf.Variable(tf.random_normal([4]))

#calculations
y_values = tf.add(tf.matmul(x,W),b)

y = tf.nn.softmax(y_values)

y_ = tf.placeholder(tf.float32,[None,4])
```

The above code depicts the architecture of the neural network implemented. The net consist of the basic 3 layers:

- INPUT layer
- HIDDEN layer
- OUTPUT layer

The input layer is depicted by a variable ‘x’. It is a matrix of shape [?,8], where ‘?’ stands for any number and 8 is the number of parameters.

The dataframe loaded from the datafile is converted into a matrix containing the values of the 8 input parameters.

The hidden layer consist of 3 variables, namely ‘W’ which is for the weights of the individual nodes of the layer. ‘b’ stands for the bias values and ‘y_value’ which computes the value of a node by the linear equation:

y_values= xW+b

Another function layer accumulated within the hidden layer is the ‘Softmax Layer’ which computes the softmax values of the y_values.

Softmax Function :

The softmax function is often used in the final layer of a neural network-based classifier. Such networks are commonly trained under a log loss (or cross-entropy) regime, giving a non-linear variant of multinomial logistic regression.

The Softmax layer outputs a matrix of shape [?,4]. 4 represents the number of classes.

The softmax ouput is stored in variable ‘y’.

The output layer of the net is represented by a matrix of shape

[?,4]. This is feeded by the hidden layer. This depicts the class for the particular instance of input parameters.

```
cost = tf.reduce_sum(tf.pow(y_ - y, 2))/(2*n_samples)

optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

Optimization:

The above code is for optimizing the graph. The ‘cost’ variable calculates the average of the difference between the original ouput and the predicted output.

The optimizer is used to minimize the cost of the prediction.

tf.train.GradientDescentOptimizer is an in-built optimizer in the tensorflow module that runs the algorithm of Gradient Descent. The only parameter used to initialize the optimizer is the learning rate.

The minimize funtion of the optimizer is used to minimize the cost of the prediction.

Initializing the Graph:

```
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
```

The above step is very important in the execution of a tensorflow graph. tf.Session() initiaizes a session of the graph whereas tf.initialize_all_variables is used initialize all the varaibles that has been used in the graph. By ‘sess.run(init)’,

we run the initialization step within our session.

Training The Model:

```
#print ("Training cost=", training_cost, "W=", sess.run(W), "b=", sess.run(b), '\n')
for i in range(training_epochs):
    sess.run(optimizer, feed_dict={x: inputX, y_: inputY}) # Take a gradient descent step using our inputs and labels

    if (i) % display_step == 0:
        cc = sess.run(cost, feed_dict={x: inputX, y_:inputY})
        print ("Training step:", '%04d' % (i), "cost=", "{:.9f}".format(cc)) #, \"W=", sess.run(W), "b=", sess.run(b)

print ("Optimization Finished!")
training_cost = sess.run(cost, feed_dict={x: inputX, y_: inputY})
print ("Training cost=", training_cost, "W=", sess.run(W), "b=", sess.run(b), '\n')
```

The above block of code is used to train the model.

Training_epochs is the number of iterations for which the classifeir will be trained and optimized.

feed_dict is used to assign values to the varaibles used in the graph.

For every display_step, the cost of the model is ouptut.

After all the iterations are finished, a message is output

along with the final weights and biases of the nodes.

Ouput of the Training Step:

```
Training step: 0000 cost= 0.020783532
Training step: 1000 cost= 0.020783532
Training step: 2000 cost= 0.020783532
Training step: 3000 cost= 0.020783531
Training step: 4000 cost= 0.020783531
Training step: 5000 cost= 0.020783531
Training step: 6000 cost= 0.020783531
Training step: 7000 cost= 0.020783529
Training step: 8000 cost= 0.020783529
Training step: 9000 cost= 0.020783529
Training step: 10000 cost= 0.020783529
Training step: 11000 cost= 0.020783529
Training step: 12000 cost= 0.020783529
Training step: 13000 cost= 0.020783529
Training step: 14000 cost= 0.020783529
Training step: 15000 cost= 0.020783529
Training step: 16000 cost= 0.020783527
Training step: 17000 cost= 0.020783527
Training step: 18000 cost= 0.020783527
Training step: 19000 cost= 0.020783527
Optimization Finished!
Training cost= 0.0207835 W= [[ 0.15463702 -1.21701765  0.52780521 -1.4738549 ]
 [ 0.42893121 -0.04660674  0.82068664  0.09615696]
 [ 0.62039953  0.93298858 -0.19308956 -1.06650257]
 [-0.05696351 -1.84619224 -0.93848866 -0.38569927]
 [-0.67261547  0.19050606 -0.29760489 -0.62254256]
 [-1.1083765  -2.41506457  0.25277814 -2.16972232]
 [ 0.35176882  0.16467018  2.88074422 -0.33394173]
 [ 1.98250139 -0.09985847 -0.04978497 -0.10028272]] b= [ 0.13422392 -0.29879299  0.13994822  1.51326454]
```

Testing Step:

```
x = tf.placeholder(tf.float32,[None,8])
#weights
W = tf.Variable([[0.15463702, -1.21701765, 0.52780521, -1.4738549],
                 [0.42833516, -0.04660674, 0.82068664, 0.09615696],
                 [0.62039953, 0.93298858, -0.19275554, -1.06650257],
                 [-0.05696351, -1.84619224, -0.93848866, -0.38569927],
                 [-0.67261547, 0.19050605, -0.29760489, -0.62254256],
                 [-1.1083765, -2.41506457, 0.25277814, -2.16972232],
                 [0.35110912, 0.16467018, 2.88074422, -0.33394173],
                 [1.98250139, -0.09985847, -0.04889834, -0.10028272]])
#W2 = tf.Variable(tf.random_normal([16,4]))
#biases
b = tf.Variable([0.13422392, -0.29879299, 0.13994822, 1.51326454])
#b2 = tf.Variable(tf.random_normal([4]))

#calculations
y_values = tf.add(tf.matmul(x,W),b)

y = tf.nn.softmax(y_values)

y_ = tf.placeholder(tf.float32,[None,4])
```

The final model is shown above.

Three different test files are provided as input to the model. The different test files are test1, test2 and test3.

```
with tf.Session() as sess:
    init = tf.initialize_all_variables()
    sess.run(init)
    result = sess.run(y,feed_dict={x:inputX})
```

The above block of code computes the result of individual files.

Output of test1

```
At ['Day_0:13:10']Remote DB is down
At ['Day_0:13:34']Remote DB Back Up and Running

At ['Day_0:14:4']Remote DB is down
At ['Day_0:14:6']Remote DB Back Up and Running

At ['Day_1:4:27']Remote DB is down
At ['Day_1:6:26']Remote DB Back Up and Running

At ['Day_1:9:10']Remote DB is down
At ['Day_1:9:12']Remote DB Back Up and Running

At ['Day_1:9:26']Remote DB is down
At ['Day_1:9:28']Remote DB Back Up and Running

At ['Day_1:9:30']Remote DB is down
At ['Day_1:9:32']Remote DB Back Up and Running

At ['Day_1:9:54']Remote DB is down
At ['Day_1:9:56']Remote DB Back Up and Running

At ['Day_1:13:56']Remote DB is down
At ['Day_1:13:58']Remote DB Back Up and Running

At ['Day_2:4:26']Remote DB is down
At ['Day_2:6:28']Remote DB Back Up and Running

At ['Day_3:3:35']Remote DB is down
At ['Day_3:3:37']Remote DB Back Up and Running

At ['Day_3:3:53']Remote DB is down
At ['Day_3:3:55']Remote DB Back Up and Running

```

Output of test2

```
At ['Day_0:17:54']Remote DB is down
At ['Day_0:18:58']Remote DB Back Up and Running

At ['Day_0:18:59']Remote DB is down
At ['Day_0:19:55']Remote DB Back Up and Running

At ['Day_0:21:29']Remote App is down
At ['Day_0:21:31']Remote App Back Up and Running

At ['Day_1:10:38']Remote App is down
At ['Day_1:10:40']Remote App Back Up and Running

At ['Day_1:17:53']Remote DB is down
At ['Day_1:19:52']Remote DB Back Up and Running

At ['Day_2:17:49']Remote DB is down
At ['Day_2:19:50']Remote DB Back Up and Running

```

Output of test3

```
At ['Day_0:14:48']Remote DB is down
At ['Day_0:16:50']Remote DB Back Up and Running

At ['Day_0:23:4']Remote DB is down
At ['Day_0:23:6']Remote DB Back Up and Running

At ['Day_1:14:44']Remote DB is down
At ['Day_1:16:17']Remote DB Back Up and Running

At ['Day_1:16:18']Remote DB is down
At ['Day_1:16:43']Remote DB Back Up and Running

At ['Day_1:18:38']Remote DB is down
At ['Day_1:18:40']Remote DB Back Up and Running

At ['Day_2:14:42']Remote DB is down
At ['Day_2:16:41']Remote DB Back Up and Running

At ['Day_2:18:45']Remote DB is down
At ['Day_2:18:47']Remote DB Back Up and Running

At ['Day_2:19:16']Remote DB is down
At ['Day_2:19:18']Remote DB Back Up and Running

At ['Day_2:19:25']Remote DB is down
At ['Day_2:19:27']Remote DB Back Up and Running

At ['Day_2:19:28']Remote DB is down
At ['Day_2:19:30']Remote DB Back Up and Running

At ['Day_2:19:48']Remote DB is down
At ['Day_2:19:50']Remote DB Back Up and Running

At ['Day_3:14:28']Remote DB is down
```

CONCLUSION

This project, made by me, is a small effort made towards the development of larger programs, and it involves limited aspects which are required in training management. In my project, I hope I have been successful in speeding up the process of detection and diagnosis of faults in the servers. I hope that my project helps in providing a more accurate prediction than the manual process currently being used. The software development although is a very difficult task but it can be carried out successfully. After the completion of this project, I have learned different things about software and its development and hope to employ the skills which I have acquired during this tenure for development of technologies.

REFERENCES

Website reference

1. stackoverflow.com
2. google.com
3. https://en.wikipedia.org/wiki/Artificial_neural_network
4. http://scikit-learn.org/stable/
5. https://cognitiveclass.ai/courses/neural-networks/
6. https://cognitiveclass.ai/courses/introduction-deep-learning/
7. https://cognitiveclass.ai/courses/deep-learning-tensorflow/