A Development Environment to Integrate Big Data with Deep Learning

Daljeet Kaur Kaushal

Follow this and additional works at: https://repository.fit.edu/etd

Part of the Computer Sciences Commons
A Development Environment to Integrate Big Data with Deep Learning

by
Daljeet Kaur Kaushal

Master of Science
Computer Science
Florida Institute of Technology
2018

A thesis submitted to
Florida Institute of Technology
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science

Melbourne, Florida
July 2018
We, the undersigned committee, hereby approve the attached thesis.

A Development Environment to Integrate Big Data with Deep Learning

by

Daljeet Kaur Kaushal

__________________________________________

Thomas Eskridge, Ph.D.
Associate Professor
Major Advisor
Harris Institute for Assured Information

__________________________________________

William Allen, Ph.D.
Associate Professor
Committee Member
Harris Institute for Assured Information

__________________________________________

Carlos Otero, Ph.D.
Associate Professor
Committee Member
Electrical and Computer Engineering

__________________________________________

Philip Bernhard, Ph.D.
Director of School of Computing
Abstract

Title: A Development Environment to Integrate Big Data with Deep Learning
Author: Daljeet Kaur Kaushal
Committee Chair: Thomas Eskridge, Ph.D.

This thesis describes a development environment that integrates big data architectures and deep learning models to facilitate rapid experimentation. The thesis makes three major contributions: First, it describes a big-data architecture that supports big data collection and organization supporting deep learning models. Second, it describes a language used to create a data view that converts the various big data streams into a view that can be used by a deep learning system. Third, it demonstrates the system’s effectiveness by applying the tool to several different deep learning applications.
Contents

1 Introduction 1

2 Background 7

2.1 Big Data Processing Frameworks 7

2.1.1 Apache Hadoop 8

2.1.1.1 Hadoop Distributed File System (HDFS) 8

2.1.2 Apache Spark 9

2.2 Big Data Processing Tools 10

2.2.1 Map-Reduce 11

2.2.2 Pig 13

2.2.3 Hive 14

2.3 Overview of Deep Learning 15

2.3.1 Deep Belief Network 16
3 Literature Review

3.1 Distributed Deep Learning for Big Data

3.1.1 Data Parellelism:

3.1.1.1 Mobile Big Data Analytics using Deep Learning and Apache Spark:

3.1.1.2 A Big Data Analysis Framework Using Apache Spark and Deep Learning:

3.1.2 Model Parallelism:

3.1.2.1 On Model Parallelization and Scheduling Strategies for Distributed Machine Learning
3.1.3 Data-Model Parallelism ............................................. 37

3.1.3.1 Spark Based Distributed Deep Learning Framework For Big Data Applications ............................................. 38

3.2 Large Scale Deep Learning for Big Data .......................... 40

3.2.1 Deep Learning using CPUs ........................................ 40

3.2.1.1 Large Scale Distributed Deep Networks: ................. 40

3.2.2 Large-Scale Deep Learning using GPUs ........................ 42

3.2.2.1 Large-scale Deep Unsupervised Learning using Graphics Processors ................................................. 43

3.2.3 Deep learning with COTS HPC systems ....................... 44

3.3 Data Pre-processing Tools ............................................. 46

3.3.1 WEKA: ................................................................. 46

3.3.2 RapidMiner: .......................................................... 48

3.3.3 KNIME ................................................................. 49

3.4 Shortcomings of the existing systems ............................... 50

4 Proposed Approach ......................................................... 52

4.1 Our Proposed Approach: ................................................ 52

4.1.1 User View ............................................................. 53
4.1.2 Controller ............................................. 54
4.1.3 Data Selector ......................................... 55
4.1.4 Data Pre-processor ................................. 56
4.1.5 Data Trainer .......................................... 58
4.1.6 Automate Steps ..................................... 58

5 Prototype .................................................. 60

5.1 Our Prototype ........................................... 60

5.1.1 Data Selector ........................................ 61

5.1.1.1 User Credentials .................................. 61

5.1.1.2 Data Selection ................................... 62

5.1.2 Data Pre-processor .................................. 62

5.1.2.1 Operations for Text Data ....................... 62

5.1.2.1.1 Data Reduction ............................... 63

5.1.2.1.2 Data Cleaning ................................. 63

5.1.2.1.3 Data Integration .............................. 63

5.1.2.1.4 Data Transformation ......................... 64

5.1.2.2 Operations for Image Data ..................... 64

5.1.3 Data Trainer .......................................... 64
5.1.4 Automate Steps

6 Case Study

6.1 Scenario 1
   6.1.1 Setup
   6.1.2 Dataset Used
   6.1.3 Experiment
   6.1.4 Evaluation

6.2 Scenario 2
   6.2.1 Setup
   6.2.2 Dataset Used
   6.2.3 Experiment
   6.2.4 Evaluation

6.3 Scenario Observations

7 Conclusion
List of Figures

1.1 Performance of Deep Learning with respect to amount of data [27] 3
1.2 An example for performing twitter sentiment analysis 5
1.3 Proposed Approach 5

2.1 Map-Reduce Architecture [44] 12
2.2 A high-level view of the data processing is done using Pig Scripts. 13
2.3 Restricted Boltzmann Machine Network [19] 17
2.4 Deep Belief Network architecture [8] 17
2.5 An unrolled recurrent neural network 19
2.6 Long Short-Term Memory Network Architecture [41] 20
2.7 Memory Block of LSTM [2] 21
2.8 Convolutional Neural Network for image classification [8] 22

3.1 Data Parallelism [10] 29
3.2 A Spark-based framework for distributed deep learning in MBD analytics [5]. .......................................................... 30

3.3 Schematic Representation of framework using Apache Spark and Deep Learning [17]. .................................................. 32

3.4 Model Parallelism [10] ............................................................. 35

3.5 High-level architecture of STRADS system interface for dynamic model parallelism [25]. ........................................ 36

3.6 Data-Model Parallelism .......................................................... 38

3.7 Overall architecture of Spark Based Distributed Deep Learning Framework for Big Data Applications [22]. .................. 39

3.8 Nvidia GeForce GTX 280 graphics card [33] .......................... 43

4.1 Proposed Approach Architecture .......................................... 54

4.2 Data Selection ................................................................. 55

4.3 Operations in Data Pre-processor ......................................... 57

6.1 Overview of Scenario 1 .......................................................... 67

6.2 Selection screen for experiment ........................................... 68

6.3 Selection screen for Big Data Stream ................................... 69

6.4 Selection screen for Data Source ......................................... 69

6.5 Accepts the IP Address and user credentials of the Hadoop machine. 70

x
6.6 Directory list of Hadoop machine. ......................................... 71
6.7 Operations on text file. .......................................................... 72
6.8 Selection Screen for Deep Learning task ................................. 72
6.9 Selection Screen for Deep Learning Model ............................. 73
6.10 Selection of Model Hyper-parameters and output of DL model .. 74
6.11 Comparison of Manual Approach and Proposed Approach for Twitter sentiment analysis ......................................................... 76
6.12 Comparison of Manual Approach and Proposed Approach for MNIST handwritten digit classification ................................................. 79
List of Tables

2.1 Comparison of Deep Learning Frameworks [37]. . . . . . . . . . . . 27
I would like to extend immense gratitude to my advisor, Dr. Thomas Eskridge for always believing in me. His constant motivation, valuable advice during the crucial times and mentorship during my entire research helped me to finish my thesis in time.

I would like to thank my parents, Balbir Kaur and Surinderpal Singh Kaushal who have pushed me throughout to challenge myself. My sisters Richa and Gagandeep, for giving me emotional and intellectual support in ways only sisters can provide. My brother, Rajveer Singh Kaushal for his encouragement throughout my thesis.

I would also like to thank Vineet Palan and Smit Desai for listening to my ideas, reviewing my thesis and providing timely feedback which helped me to put my thesis together.
This thesis is dedicated to my parents

Balbir Kaur and Surinderpal Singh Kaushal

for constantly motivating me

and having faith in me.
Chapter 1

Introduction

With the advent of digital technology and smart devices, a huge amount of data is generated at astonishing rates. According to the National Security Agency, the Internet is processing 1,826 Peta-bytes of data per day [4]. With the emerging technologies and the data generated with all IoT (Internet of Things) devices, it is predicted that more amounts of data will be generated, than currently being processed, in the next few years. In fact, about 90% of the current data was created in the last couple of years, and it will continue to grow in the foreseeable future [4].

The exponential growth of digital data generated from numerous and disparate sources makes it impossible to store, process, and analyze using traditional techniques. These limitations have resulted in the evolution of technologies around Big Data. Big Data, defined by rapid growth of volume, variety, and velocity of data, typically deals with unstructured data that require large batch or real-time analysis.
It has also created an enormous paradigm shift in data-driven discovery and analytical data processing. It has been proven to be beneficial for various sectors such as health-care industry manufacturing, enterprises, educational services and rapidly expanding in all science and engineering domains, including physical, biological and bio-medical sciences [43].

For inferring meaningful results from such volumes of data, an enormous capacity is required in terms of storage and computing resources and also requires parallel processing techniques to process the huge datasets.

Machine learning (ML) is a branch of artificial intelligence that discovers patterns by analyzing large datasets. The insights gained using ML techniques are more profound and derives results at accelerated pace & scale that humans cannot match [12]. Deep Learning (DL), a subset of machine learning, is a technique used to analyze and process a humongous amount of data to help us find abstract and useful patterns from it [14]. Deep Learning learns from multiple levels of representations and abstractions of any data such as text, images, sound, video, time-series, etc.
If we apply Deep Learning to Big Data, we can find unknown and useful patterns that were previously impossible to find [14]. From figure 1.1, we see that the traditional machine learning approaches show better performance for lesser amounts of input data. As the amount of data increases beyond a certain amount, the performance of the traditional machine learning approaches become steady, i.e., it reaches a plateau. However, the performance of deep learning approaches increases with respect to the increment in the amount of data [27]. Hence, we use Deep Learning techniques to extract meaningful information from Big Data.

Using deep learning to infer meaningful information from Big Data often becomes quite challenging due to the following:
- Volume: DL model running time and complexity increases with a large number of examples (inputs), large varieties of class types (outputs), high dimensionality (attributes), and the presence of noisy and incomplete data.

- Variety: Data comes in all types of formats from disparate sources and with different distributions.

- Velocity: Data is generated at very high speeds, so batch and real-time data streams need to be handled.

A typical example of deep learning using Big Data is Twitter Sentiment Analysis. Twitter publishes millions of tweets per hour containing all kinds of information, disseminating views on many different topics such as politics, business, economics and so on. Twitter users regularly publish their opinions on a particular news item, a recently purchased product or service, and ultimately on everything that happens around them. Hence, nowadays, many companies are doing sentiment analysis on user tweets to evaluate the value of their products.

To achieve this, we need to collect all the product tweets to perform sentiment analysis. Then we pre-process the collected tweets to remove all the irrelevant data. After the pre-processing is done, we finally train deep learning models to get information about a product/political views/business. Figure 1.2 gives an overview of steps to use Twitter data with deep learning.
This thesis aims at proposing a user-friendly tool that provides a development environment to allow users to integrate big data management & processing frameworks with deep learning systems. It allows the users to perform all the required data preprocessing operations to get the data in the format as required by deep learning models and train them. Our tool handles different data streams from big data architectures and lets users experiment with different deep learning tasks. Figure 1.3 shows the proposed approach of our thesis.
We define the following requirements for our proposed tool:

- It should support different types of data streams such as text, image, video, audio, time series, etc.

- It should handle processing of batch/real-time data from different big data architectures such as HDFS (Hadoop Distributed File System), Apache Spark\(^1\), and Apache Storm\(^2\).

- It should allow the users to interact and decide what data pre-processing operations need to be performed.

- It should support the construction of big data - deep learning systems to address different tasks such as classification, prediction, recommendation, etc. to infer meaning information from the data.

The rest of this thesis is organized as follows: Chapter 2 contains the background of all big data management & processing frameworks and deep learning systems. Chapter 3 contains literature review of big data and deep learning and available tools for data pre-processing. Chapter 4 describes the tool we propose to integrate different big data management systems and deep learning systems. Chapter 5 explains the prototype implementation of our proposed tool. Chapter 6 explores how this tool can be used with the help of two case studies. Chapter 7 concludes and summarizes our proposed tool.

\(^1\)https://spark.apache.org/
\(^2\)http://storm.apache.org/
Chapter 2

Background

In this chapter, we look into the background and different terminologies used in our thesis related to big data and deep learning. We discuss different big data management frameworks and various tools available for processing big data. We also discuss different deep learning models and different open source frameworks available to implement them.

2.1 Big Data Processing Frameworks

With the increase in large amounts of data generates from a variety of sources, existing data processing technologies are not suitable to cope with it. Big data frameworks help in storing, analyzing and processing such massive amounts of data [20]. In this section, we discuss different Big Data Frameworks and we categorize
them according to their key features such as: (1) the programming model (2) supported programming languages (3) the type of data-sources they require.

### 2.1.1 Apache Hadoop

Apache Hadoop\(^1\) (High-availability distributed object-oriented platform) is a Java-based programming framework that supports the storage, management, and processing of large data sets in a distributed computing environment. It is an Apache project, founded in 2008 by Doug Cutting at Yahoo and Mike Cafarella at the University of Michigan, inspired by published papers of Google [31]. Since then, Hadoop has become a standard for storing, processing and managing hundreds of terabytes and even petabytes of data.

Apache Hadoop consists of two main components:

- Hadoop Distributed File System (HDFS)
- Map-Reduce programming model

#### 2.1.1.1 Hadoop Distributed File System (HDFS)

HDFS is a scalable and distributed file system designed to store a large amount of data in distributed servers/clusters. HDFS partitions the data into small blocks of data (default block size is 64 MB) and distributes the blocks of data across its clusters [18].

\(^{1}\)http://hadoop.apache.org/
HDFS consists of two main components, namely, Name Nodes and Data Nodes.

- **Name Node:**
  The Name Node is the main component of HDFS as it is responsible for allocation of physical space to store files. It maintains metadata for all files and directories in the file system as well as manages file system operations. Whenever the client wants to retrieve data from HDFS, the request is sent to the name node. The Name Node then searches for the file location in its indexing system and sends the metadata (filename, file location, etc.) back to the HDFS client. The availability of HDFS is enhanced using Secondary Name Node. The Secondary Name Node stores the state of the Name Node periodically and takes over automatically in-case of Name Node failure.

- **Data Node:**
  The Data Nodes are the places where all the data reside. They are responsible for the storage and retrieval of large files in the form of blocks as directed by Name Node. The Data Nodes report back to the Name Node with lists of blocks that they store periodically.

### 2.1.2 Apache Spark

Apache Spark\(^2\), a high-level, general-purpose cluster computing platform designed to be fast and fault-tolerant, is used by many companies such as Yahoo, Baidu, and Tencent\(^[20]\). The key concept that differentiates Spark from other Big Data frame-

\(^2\)https://spark.apache.org/
works like Hadoop is that it works in-memory using Resilient Distributed Datasets (RDDs), speeding up processing times of the data. An RDD is an immutable collection of objects spread across spark cluster. Spark is also designed to be compatible with Hadoop.

There are two types of operations performed on RDDs [20]:

- **Transformations**: Transformations consist of the creation of new RDDs from the existing RDDs using functions such as map, filter, union and join.

- **Actions**: Actions are the final results of RDD computations.

Spark provides various Application Programming Interface (APIs) such as Spark Core that allows the users to perform distributed programming, Spark Streaming for real-time processing of live data streams, Spark SQL for querying data via SQL, Spark MLLib that provides users with different machine learning algorithms, and GraphX for manipulating graphs and performing graph-parallel computations [18].

## 2.2 Big Data Processing Tools

Big Data is used for a collection of data sets so large and complex that it is difficult to process using traditional applications/tools. One challenge is how to handle unstructured data and get the data in a format before we attempt to understand and capture relevant information from it. To make sense out of the data requires the use of sophisticated big data processing tools. In this section, we discuss a few big data processing tools.
2.2.1 Map-Reduce

Hadoop MapReduce is a programming model that processes large datasets in parallel on different clusters in a reliable and fault-tolerant manner. MapReduce was proposed by Google in 2004 and performs computations on large datasets while hiding the complex underlying details of parallelization, distributed storage, load balancing and fault tolerance [20].

Figure 2.1 shows components of Map-Reduce. The two main components of Map-Reduce programming model are [35]:

**Map Phase:** This phase usually processes the input data, i.e., file or directory stored in the Hadoop File System (HDFS). Similar to primary keys used in traditional relational databases, the Map phase emits key-value pairs.

**Reduce Phase:** The Reduce phase is a combination of the Shuffle, Sort and Reduce Stages. The Shuffle stage shuffles the data from the mapper function. The Sort stage sorts the output from the Shuffle phase. The Reduce stage is used to combine all the intermediate values associated with an intermediate key that comes from the sort stage. After processing, the reduce stage generates a new output which is stored in HDFS.
To summarize, a map-reduce job splits the input data into small blocks of data and the map phase processes every block of data in parallel. The output of the map task is shuffled and sorted and given as an input to the reduce task. The input as well as the output, of the Map-Reduce jobs, are stored in a Hadoop File System (HDFS). The Map-Reduce framework schedules and monitors individual tasks running in parallel and also re-executes task in-case of any failures.

*Figure 2.1: Map-Reduce Architecture [44]*
2.2.2 Pig

Pig\(^3\) is an Apache open source project that was initially developed by Yahoo \([18]\). It runs on top of Hadoop by making use of HDFS and Map-Reduce, which are the two main components of Hadoop \([35]\). Using pig, users can use Map-Reduce library and run one or more Map-Reduce jobs on Hadoop. Pig enables analysts to write complex data transformations easily without knowledge of Java programming.

![Diagram](https://example.com/diagram.png)

Figure 2.2: A high-level view of the data processing is done using Pig Scripts.

Pig consist of two main components, Pig Latin and Run-Time Environment:

1. Pig Latin: Pig consists of a high-level language, called Pig Latin, that provides a higher level of abstraction for writing Map-Reduce tasks \([35]\).

   The input and output of a Pig Latin Script is stored in HDFS. The syntax of Pig Latin is similar to SQL, for example, data is loaded into a table

\(^3\)https://pig.apache.org/
like structure consisting of different field types. As shown in figure 2.2 Pig Latin Scripts can perform operations like filtering, projecting a subset of data, statistical operations such as minimum, average, maximum, count, etc., joining of multiple datasets and also supports complex data types such as bags, tuples, and maps.

2. Run Time Environment: Pig has two modes of executing scripts:

   (a) Local Mode: Pig Scripts runs on a single local machine, Hadoop or Map-Reduce is not required.

   (b) Hadoop or Map-Reduce Mode: Pig Scripts runs on Hadoop clusters.

Pig provides three modes of user interaction to execute Pig Latin commands: Interactive mode, Embedded Mode and Batch Mode [7].

Interactive Mode: The operator uses an interactive shell to enter Pig Latin commands and inspect the results.

Batch Mode: The user submits a file consisting of Pig Latin commands.

Embedded Mode: The Pig Latin commands are embedded in a host language and submitted through method invocation from the program.

2.2.3 Hive

Apache Hadoop has a data warehouse system developed by Facebook called Hive\(^4\) [18]. Hive provides easy data summarization, ad-hoc queries and the analysis of large datasets stored in Hadoop Distributed File System (HDFS).

\(^4\)https://hive.apache.org/
Hive provides a mechanism to query the stored data using a SQL-like language called HiveQL. This language also allows traditional map-reduce programmers to plug their custom map-reduce scripts in their queries when it is inconvenient or inefficient to express their query logic. Hive also includes Hive-MetaStore, a system catalog, that contains schemas and statistics useful for optimization of the queries and data exploration.

HiveQL provides traditional SQL statements like Data Definition (DDL), and Data Manipulation (DML) statements to query the data. DDL statements are used to create tables with specific serialization formats, partitioning, and bucketing columns. DML statements are used to load data from external sources and insert query results into Hive Tables using load and insert statements respectively. HiveQL also supports various types of joins - inner, outer, left, right and outer joins, cartesian products, aggregations, union all, group bys, create table as select and many useful functions on primitive and complex types [40].

2.3 Overview of Deep Learning

In recent years, massive amounts of data are generated from various fields including medical informatics, social media, cybersecurity and an increasing number of electronic devices. Traditional data processing techniques have limitations of processing such large amounts of data. Big Data requires new and sophisticated algorithms based on machine learning and deep learning techniques to process data in real-time.
with high accuracy and efficiency [20]. However, recent research has shown deep
learning with hybrid learning and training mechanism for processing Big Data with
high speed.

Deep learning techniques have provided powerful tools to handle large amounts
of labeled/unlabeled data as they extract higher level features from them to get
hierarchical representations.

Deep learning has been applied to the fields of speech recognition, acoustic mod-
eling for audio classification, image processing such as handwritten classification,
high-resolution remote sensing scene classification, natural language processing,
computer vision, pattern recognition, etc.[26].

2.3.1 Deep Belief Network

Deep belief network (DBN) uses a deep architecture that alleviates the problems
of conventional neural networks such as they cannot take advantage of unlabeled
data and often gets trapped in local optima [34]. DBNs incorporate supervised and
unsupervised learning as it can learn feature representations from both labeled and
unlabeled datasets.

A DBN is composed of a stack of Restricted Boltzmann machines (RBMs). RBMs
consists of two layers, i.e., the visible layer, and hidden layer [19]. In every layer,
odes are fully-connected to every other node of previous and subsequent layers
and are independent of all the nodes in the same layer. Figure 2.3 shows an RBM
network with four visible nodes, i.e., $v_0, v_1, v_2$ and $v_3$ and three hidden
nodes, i.e.,
Figure 2.3: Restricted Boltzmann Machine Network [19]

h0, h1 and h2.

Figure 2.4: Deep Belief Network architecture [8]

Figure 2.4 shows typical Deep Belief Network architecture. The DBN consists of three hidden layers each with three neurons; one input layer with five neurons and one output layer also with five neurons. Any two adjacent layers can form an RBM.
trained with unlabeled data. The outputs from one RBM are the inputs to the next RBM.

Deep-belief networks are used to recognize, cluster and generate images, video sequences and motion-capture data, image/face recognition, and natural language processing.

### 2.3.2 Recurrent Neural Network

The idea behind the Recurrent Neural Network (RNN) is to make use of sequential information. For Example, if we want to predict the next word in a sentence we should know what word came before it. RNNs are called recurrent because for every element of a sequence they perform the same task, as they use the output of a step as the input for the next step. They have a memory which captures information about what has been calculated so far. A simple recurrent neural network can be viewed as a fully connected neural network if we unroll the time axes. RNNs use information in arbitrarily long sequences, but they work well only for short-term memory, and it suffers from a basic problem of not learning relationships separated by significant periods of time [36]. Figure 2.5 gives an overview of unrolled recurrent neural network.
2.3.2.1 Long Short-Term Memory Units

As discussed, a simple recurrent neural network has a problem of not being able to store long-term dependencies in a sequence. To overcome this, in late 90’s Sepp Hochreiter and Jurgen Schmidhuber proposed Long Short-Term Memory Units (LSTMs). LSTMs are insensitive to gap length over alternatives RNNs and other sequence learning methods.

Figure 2.6 shows the architecture of LSTM that consists of multiple connected memory blocks (where A represents a memory block). A typical LSTM model usually contains recurrently connected subnets [29], also called memory blocks, i.e., it remembers values over arbitrary time intervals. Each block contains one or more self-connected memory cells and three following gates:

- Input gate: It controls which new values flow into the cell.
- Output gate: It controls the extent to which values remain in the cell.
- Forget gate: It controls the extent to which the values in the cell can compute the output activation of the LSTM unit.
Figure 2.6: Long Short-Term Memory Network Architecture [41]

Figure 2.7 shows structure of memory block in LSTM. The cells learn and make decisions via the gates about when to allow data to enter, leave or be deleted through the iterative process of making guesses, back-propagating error, and adjusting weights.

The few applications of Long Short-Term Memory networks include time-series prediction, speech recognition, machine translation, language, and prediction, generate image descriptions, anomaly detection, etc.

2.3.3 Convolutional Neural Network

Convolutional Neural Networks or ConvNets or CNNs, consists of an input layer, output layer, and hidden layers. They receive input data as single vector and may contain single or multiple hidden layers. Every hidden layer contains a set of neurons, wherein every neuron is fully connected to all the previous layer neurons. Within a single layer, every neuron is independent and do not share any connections with the other neurons present within the same layer. The last fully connected layer
Figure 2.7: Memory Block of LSTM [2]

contains the output of the network. For instance, for image classification problem the output layer consists of scores for each classification class.

In a CNN network, every neuron has weights and biases. Each neuron receives some input, takes the weighted sum over them, passes it through an activation function and computes an output. The whole network has a loss function on the last fully connected layer. The three main layers of CNNs are:

- **Convolutional Layer**: This layer does most of the computation in CNN. It applies convolution operation on the input with several feature filter maps of equal sizes.
Figure 2.8: Convolutional Neural Network for image classification [8]

- **Pooling Layer:** The input to this layer is a convolutional layer. It is used to combine the results of a group of neurons at one layer into a single neuron in the next layer. The most commonly used pooling technique is max pooling and also includes average pooling. This layer is also called as subsampling layer.

- **Fully Connected Layer:** In this layer, every neuron is connected to every other neuron in the previous and subsequent layer. Within this layer, neurons are independent and not connected to other neurons present in the same layer.
Figure 2.8 depicts a typical CNN architecture. The input data is passed through the convolutional layer and then through subsampling layer to reduce the dimensionality of the data. The sequence of convolution and subsampling layer can be repeated many times and is usually pre-determined by the users [8].

Convolutional Neural Networks are very popular in the field of image recognition and classification. They are proven to be very successful in identifying faces, objects, and gestures, and in natural language processing tasks such as text/sentence classification, image, speech, audio, etc.

2.4 Deep Learning Frameworks

Deep learning techniques enable us to find solutions for complex problems very efficiently. Today, there are dozens of deep learning frameworks available such as Tensorflow, Caffe, Theano, etc., which provides training methods used to train deep learning models fast and easy. In this section, we will be discussing an overview of some of the deep learning frameworks.

2.4.1 TensorFlow

TensorFlow\footnote{https://www.tensorflow.org/} is an open source machine learning platform developed by Google Brain Team for research used in the fields of machine learning and deep neural nets [37].
To achieve excellent performance, TensorFlow expresses all numerical computations using a single data flow graph. A data flow graph is a computation graph where each node of a graph represents a mathematical computation (like matrix multiplication, convolution, etc.) and the edges of the graph represent the arguments to the computation or the output values that flow on the edges in between the nodes called tensors [37]. The communication between the sub-computation is executed by the data flow graph, which makes it possible to execute independent computations in parallel or to use multiple devices to execute partition computations[3]. The mapping of data flow graph nodes is achieved across many machines within a single cluster and across multiple computational devices such as multi-core CPUs, GPUs, and custom designed ASICs known as Tensor Processing Units (TPUs) within a single machine.

For constructing and executing data flow graphs, Tensorflow provides APIs in several languages such as C++, Java, and Python. The resilient architecture of TensorFlow allows users as well as developers to train and experiment with a wide variety of deep neural network models and is used for deploying into production machine learning models for different fields including speech recognition, computer vision, NLP, robotics, and computational drug discovery [37].

2.4.2 Caffe

Caffe\(^6\) is an open-source deep learning framework developed and maintained by the Berkeley Vision and Learning Center (BVLC) and has an active community support

\(^6\)http://caffe.berkeleyvision.org/
on GitHub. Yangqing Jia, a Ph.D. student of UC Berkeley, created this project. It is written in BSD-licensed C++ with well-supported bindings to Python/NumPy and MATLAB for training and deploying general-purpose convolutional neural networks efficiently and other deep learning models on commodity architectures. It supports parallelization in CUDA and a third party implementation of OpenCL but does not currently support OpenMP [32].

Caffe provides state-of-the-art image classification with convolutional networks. It also has a huge repository of pre-trained neural networks, called Model Zoo, that is suited for a variety of image classification tasks. It also supports Recurrent Nets and provides pre-trained models such as ResNet-50, ResNet-101, ResNet-152 on ImageNet that can be used for transfer learning [32].

Some of the advantages of using Caffe framework such as [21]:

- **Modularity**
- **Separation of representation and implementation**
- **Test Coverage**
- **Python and MATLAB bindings**
- **Pre-trained reference models**
2.4.3 Theano

Theano\(^7\) is a cross-platform open-source python library that is developed by the LISA group at the University of Montreal, Canada [32]. Theano represents neural networks and data as matrices and operations on data as matrix calculations. Due to vectorization, the code is executed fast as multiple values are processed in parallel.

Theano allows users to code every aspect of a neural net such as nodes, layers, activation, and training rate. Hence, all types of deep nets can be trained easily. It is a general use machine learning library and not just deep learning library making it highly extensible.

Block\(^8\), Lasagne\(^9\), and Keras\(^10\) have been developed to build on top of Theano’s non-intuitive interface with a higher-level user interface to train different architectures of deep learning models [32]. Theano also supports CUDA and OpenMP.

2.4.4 Comparison of Deep Learning Frameworks

In this subsection, we will be comparing the above discussed deep learning frameworks for implementing large-scale deep learning models on few parameters such as CPU support, support for multi-threaded CPU, GPU support, Multi-GPU support, NVIDIA cuDNN support, and platforms supported. Table 2.1 gives the comparisons

\(^7\)http://deeplearning.net/software/theano/
\(^8\)http://blocks.readthedocs.io/en/latest/
\(^9\)https://lasagne.readthedocs.io/en/latest/
\(^10\)https://keras.io/
Table 2.1: Comparison of Deep Learning Frameworks [37].

<table>
<thead>
<tr>
<th>Property</th>
<th>TensorFlow</th>
<th>Caffe</th>
<th>Theano</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>C++</td>
<td>C++</td>
<td>Python</td>
</tr>
<tr>
<td>CPU</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multi-Threaded CPU</td>
<td>Yes, Blas</td>
<td>Yes, Eigen</td>
<td>Yes, Blas, conv2D, limited OpenMP</td>
</tr>
<tr>
<td>GPU</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multi-GPU</td>
<td>Yes (Only parallel data)</td>
<td>Most Flexible</td>
<td>No, Experimental Version available</td>
</tr>
<tr>
<td>Nvidia cuDNN</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Platform</td>
<td>Linux, Mac OS X</td>
<td>Ubuntu, OS X, AWS, unofficial port for windows and android</td>
<td>Cross-Platform</td>
</tr>
</tbody>
</table>

of the discussed frameworks. By evaluating the frameworks on these properties, we choose tensorflow to implement deep learning models on large-scale systems.
Chapter 3

Literature Review

In this chapter, we discuss the existing approaches to train/implement deep learning models for big data problems. We review each approach, summarize it and also discuss the available tools for performing data pre-processing.

3.1 Distributed Deep Learning for Big Data

Using large datasets for the training of neural network architectures has an enormous impact in many domains but requires massive amounts of computational power. Since training neural networks on a single machine often takes impractically long time, given the advances in network architectures, GPU hardware and training methods, a considerable amount of work and research has been carried out for the distributed training of neural networks. The following two approaches can be considered for parallelizing/distributing training of neural nets [6]:
- Data Parallelism: The data is distributed across nodes

- Model Parallelism: Model parameters are divided and assigned across nodes.

### 3.1.1 Data Parallelism:

In distributed deep learning with data parallelism, the training data is divided into small chunks of data and distributed across the nodes and model parameters are replicated across every node, as shown in figure 3.1. Every node updates the model local to it by conducting training of one small batch of the local training data. A new model is then created from the updated models (or gradients used to update the model) and broadcasted to all the nodes.

![Data Parallelism Diagram](image)

**Figure 3.1:** Data Parallelism [10]
3.1.1.1 Mobile Big Data Analytics using Deep Learning and Apache Spark:

The authors in [5] propose an approach to perform analytics of Mobile Big Data (MBD) using Deep Learning and Apache Spark Framework. The proposed framework intends to speed up the Mobile Big Data decision-making process by parallelizing the training and learning of deep learning models in a high-performance computing cluster.

The proposed framework uses Apache Spark to address the problem of volume, velocity, and variety of MBD. The deep learning model addresses the variety and value problem of MBD.

![Figure 3.2: A Spark-based framework for distributed deep learning in MBD analytics [5].](image)

In this approach, the mobile big data is divided into small batches of data and distributed across every Spark Worker node. The data is stored in Resilient Distributed Dataset (RDD) of the Spark engine. The deep learning model is executed
on many spark workers as iterative Map-Reduce computing. Every spark worker trains their respective local model with the partition of the overall mobile big data. By averaging the parameters of all the partial models obtained from every worker, a master deep model is then built.

The Spark framework consists of two main components:

- **Spark Master**: It is responsible for initializing an instance of the Spark driver that manages the execution of many partial models in a group of Spark workers.

- **One or more Spark workers**: The Spark driver, initialized by the master machine, manages the execution of partial deep learning models in a group of Spark workers. Each Spark worker node on every iteration of the deep learning algorithm learns a partial deep model on the small partitions of MBD and sends the computed parameters back to the master node. The master node averages the partial models of all executor nodes and reconstructs the master deep model.

The learning of deep models is done in two steps:

- **Gradient Computation**.

- **Parameter Update**: Parameter Update consists of two steps. In the first step, the learning algorithm computes the gradients of model’s parameters locally by iterating through all the distributed data batches independently. The second step calculates the average of all the local gradient model’s parameter
obtained from the first step and updates the learning model parameters accordingly.

These two steps describe deep learning model in Map-Reduce programming model wherein the map function carries out the gradient computation across all data batches in parallel, and the reduce function updates the model’s parameters globally.

### 3.1.1.2 A Big Data Analysis Framework Using Apache Spark and Deep Learning:

The authors in [17] aim to provide an approach that proves to be an improvement over traditional big data analysis methods that use either Hadoop/Spark or deep learning as individual elements. They propose a framework that uses distributive computational abilities of Spark and deep learning architecture of Multi-Layer Perceptron (MLP) using Cascade learning.

**Proposed Framework**

![Proposed Framework Diagram](image)

**Figure 3.3:** Schematic Representation of framework using Apache Spark and Deep Learning [17].
3.1.1.2.1 Framework

The framework is structured into three stages forming the core of research and experiments. The three stages are explained below:

• **Stage 1:**
  This stage is a binary-learning stage where a regression model, created by passing pre-processed dataset through the regression algorithms, presents the probability of each data-point belonging to a binary class. This stage consists mainly of Apache Spark as it uses the MLLib library of Spark to implement regression algorithms such as decision tree, random forests and logistic regression.

• **Stage 2:**
  Stage 2 is Cascading Learning stage. The process of Cascading involves using the results that we obtain from one model to train another model. In this framework, stage 2 modifies the original dataset by appending the probabilities obtained from Stage 1. The probabilities turned out to be a strong distinguishing feature in the modified dataset and used as the ground-truth value for every data-point. This modified dataset is used as an input for stage 3.

• **Stage 3:**
  This stage uses the modified dataset obtained from stage 2 to train a Multilayer Perceptron(MLP) architecture. The architecture of MLP varies ac-
According to the requirements of an application, this stage can be used for binary classification or multi-class classification. The network depth of MLP architecture is dependent on the complexity of the problem and the system's computational complexity.

Apart from the benefits of using Spark and deep-learning models together, this approach also offers other important advantages such as enhanced feature set obtained from stage 2 improves the overall accuracy of the model. Also, using MLLib library of Spark and deep learning model together decreased the computational time required as compared to using a two-layer approach using two deep learning models simultaneously. The framework uses back-propagation based deep learning model resulting in continuous learning and improvements. Hence, this makes the framework more reliable.

One challenge faced by this approach is that when both stages are multi-class classifiers, the framework produced a lower performance. The experiments showed when stage 1 multi-class classifier was used in combination with multi-class deep learning model in stage 3 resulted in lower performance compared to when both the stages using binary classifiers.

### 3.1.2 Model Parallelism

In distributed deep learning with model parallelism, model parameters are divided and assigned to every compute node and data to train the models is replicated on
every node, as shown in figure 3.4. The information gathered from the model of
every other node updates the local model.

![Model Parallelism](image)

**Figure 3.4:** Model Parallelism [10]

### 3.1.2.1 On Model Parallelization and Scheduling Strategies for Distributed Machine Learning

The authors in [25] develop a system for model parallelism, Structure Aware Dynamic Scheduler (STRADS), that performs automatic scheduling as well as parameter prioritization for implementing Model parallelism.

To exploit model-parallelism, STRADS provides a programming interface where users can write three functions for any machine learning problem: schedule, push
Figure 3.5: High-level architecture of STRADS system interface for dynamic model parallelism [25].

and pull. STRADS creates an iterative model-parallelism algorithm by repeating scheduling and executing these three functions. The description of these three functions are:

- **Schedule**: The schedule function selects model parameters to be dispatched for updates. It selects model parameters uniformly or according to a fixed sequence. While selecting model parameters, it ensures that only fastest-converging parameters are selected to avoid already converged parameters and avoids parallelization errors by not dispatching parameters with interdependencies in parallel.

- **Push and Pull**: Push and Pull functions describe the flow of model parameters from the scheduler to the workers. Push functions dispatch the selected model parameters from the scheduler to each worker. Every worker will then compute a partial update for the model parameters. The Push function is
used to collect the partial updates from all workers and commit them for the full parameter update.

STRADS uses multiple master/scheduler machines, worker machines and one master machine that coordinates the flow between the schedulers and workers. STRADS executes the schedule-push-pull functions in round-order fashion.

The steps of implementation are as shown in figure 3.5: (1) the master machine executes the schedule function to select the model parameters to be used in parallel. (2) The push function is executed to dispatch the model parameters on the worker machines and compute the partial updates for each parameter. (3) the key-value stores execute the pull function to aggregate the partial updates and keep the newly updated parameters.

STRADS provides scalability and efficient memory utilization by allowing larger models to run with additional machines. It also provides the ability to invoke dynamic schedules that reduce model parameter dependencies across workers resulting in lower parallelization error and thus faster and correct convergence.

3.1.3 Data-Model Parallelism

Model and Data Parallelism uses the concept of Centralized Parameter Server that is responsible for holding and updating the model parameters. Each node contains a partition of training data and replica of the neural network model as shown in figure 3.6. Every node independently performs training of one batch of the local training
data and updates local model. It then sends the model updates to the parameter server. Each node receives the model updates of other nodes from parameter server.

![Data-Model Parallelism](image)

**Figure 3.6**: Data-Model Parallelism

### 3.1.3.1 Spark Based Distributed Deep Learning Framework For Big Data Applications

The authors in [22] have designed and implemented a framework to train deep learning models using Apache Spark. This framework speeds up the training time of large-scale deep learning networks by distributing the same model, via stochastic gradient descent, among data node clusters in HDFS.

This framework exploits the advantages of both, data and model parallelism. The partition of training data across spark cluster machines and model replication on
Figure 3.7: Overall architecture of Spark Based Distributed Deep Learning Framework for Big Data Applications [22].

Each machine implements data parallelism. Each model trains in parallel with its data partition. The distribution of layers of each deep neural network model replica across the spark cluster implements model parallelism.

The framework consists of three main components:

- Spark Master: It has two main components, namely, Spark Driver which is responsible for maintenance, coordination, and scheduling of applications on the cluster, and Worker Nodes which are responsible for running applications. It starts the training process of the neural network by initializing the parameters on the Parameter Server and neural network layers.

- Parameter Server: Parameter server is partitioned across machines in spark cluster, and every partition corresponds to a layer of the neural network model.
• Data Shard: Data shards are data nodes. According to the network layer size, every data shard generates neural network layers and for each data partition it computes forward and backward passes for each data example until it processes all examples. It updates the master model, after each iteration, by sending the newly computed gradients to the Parameter Server master model.

3.2 Large Scale Deep Learning for Big Data

While deep learning has shown extraordinary results in many applications, its training is not easy for Big Data applications since it is very computationally intensive. These computations become more complex with growing datasets. Thus, with the unprecedented growth of commercial datasets in recent years, there is a surge in interest in effective and scalable parallel algorithms for training deep models. In this section, we discuss high-performance computing devices and architectures such as graphics processing units (GPUs) and CPU clusters that enables the training of large-scale deep learning models for big data feature learning.

3.2.1 Deep Learning using CPUs

3.2.1.1 Large Scale Distributed Deep Networks:

Dean et al. [10] developed a software framework Distbelief that could exploit the computing power of clusters with thousands of machines to train large-scale deep
learning algorithms. The Distbelief framework provides model parallelism within a node via multi-threading and across node via message passing [22].

Within this framework, two algorithms were developed for large-scale training of deep learning algorithms:

- Downpour Stochastic Gradient Descent (SGD): an asynchronous stochastic gradient descent procedure supporting a large number of model replicas with adaptive learning rates.

- Sandblaster: a framework that supports distributed batch optimization procedures, including a distributed implementation of L-BFGS (Limited-Memory Broyden Fletcher Goldfarb Shanno) [30], that uses both model and data parallelism.

Both of the algorithms are designed in a manner that they can tolerate variation in the execution speed of different model replicas and are more robust to machine failures such as a machine is offline or restarted at random. Both of the optimized algorithms implement an intelligent version of data parallelism. They also enable simultaneous execution of distinct training examples in each of the model replicas and gather their results periodically to optimize the model’s objective function.

DistBelief has obtained a high speedup for training several large-scale deep learning models. For instance, it achieved a speedup of 12 times for a convolutional neural network with 1.7 billion parameters and 16 million images on 81 machines. Furthermore, it has also achieved a significant improvement for training another deep learning architecture with 14 million images of size 200 x 200 pixels on 1000
machines each with 16 CPU cores. Distbelief framework performs extraordinarily to train deep learning models for big data feature learning since it can scale up over many computers.

### 3.2.2 Large-Scale Deep Learning using GPUs

Nowadays, modern desktop systems are shipped with a typical graphics card that contains over a hundred processing cores and has a memory bandwidth several times higher than their CPUs. The hardware is designed to work concurrently on thousands of threads with very little overhead. This makes GPUs increasingly attractive for general-purpose computation that is hard to parallelize on other distributed architectures. Some experiments have been performed to evaluate large-scale deep learning frameworks such as Caffe, CNTK¹ (Microsoft Cognitive toolkit), MXNet², and tensorflow, over single GPU and multi-GPU environments [38].

While a thorough introduction to graphics processor architecture is beyond the scope of this thesis, we discuss some basic ideas behind GPU’s successful computation.

Figure 3.8 shows a simplified schematic of a typical Nvidia GPU. A typical GPU consists of several multiprocessors (MPs) and each multiprocessor contains several stream processors (SPs) that are responsible for the actual computation. Every MP runs scheduled group of threads called blocks. Within each MP, each block is

²https://mxnet.apache.org/
Figure 3.8: Nvidia GeForce GTX 280 graphics card [33]

scheduled to run on a SP. Every GPU has a global memory with very high bandwidth and high latency when accessed by the CPU (host).

The GPU hardware allows for two levels of parallelism: instruction (memory) level (i.e., MPs) and thread level (SPs). All threads within a block have shared access to a small amount of local shared memory (16KB). They also have access to GPU-wide global memory (upto 4GB).

3.2.2.1 Large-scale Deep Unsupervised Learning using Graphics Processors

Raina et al. [33] propose a GPU-based framework to parallelize unsupervised learning models including DBNs and sparse coding. The primary objective was to train unsupervised learning models for large-scale applications with millions of free pa-
rameters, which forces the researchers to use smaller-scale models or reduce the training examples.

Since transferring data between the host and GPU global memory is a time-consuming process, there is a need to minimize the transfers between host and device by taking advantage of shared memory. One strategy to reduce the transfer times is to store all parameters and a large chunk of training examples in global memory during training. This strategy allows parameter updates to be carried out entirely inside GPUs. To take advantage of MP/SP levels of parallelism as shown in figure 3.8, a few training examples from global memory will be selected on each iteration to compute the update concurrently across blocks, which helps in achieving data parallelism.

Experimental results show that with 45 million parameters in an RBM and one million examples, the GPU-based implementation increases the speed of DBN learning by a factor of up to 70, compared to a dual-core CPU implementation (around 29 minutes for GPU-based implementation versus more than one day for CPU-based implementation) [33].

### 3.2.3 Deep learning with COTS HPC systems

The Distbelief framework in [10] can learn with huge models of more than one billion parameters. However, it uses 16,000 CPU cores for training, which is not commonly available for most researchers. Coates et al [9] presents an alternative approach. The Commodity Off-The-Shelf High-Performance Computing (COTS
HPC) technology can train deep learning models with more than 11 billion parameters using three machines, instead of a considerable number of CPU cores.

The COTS HPC system is comprised of a cluster of 16 GPU servers with Infiniband adapter for interconnects and MPI for data exchange in a cluster. Every server contains four NVIDIA GTX680 GPUs, each having 4GB of memory. COTS HPC is capable of running very large-scale deep learning with a well-balanced number of GPUs and CPUs.

COTS HPC systems effectively utilize memory and perform efficient computations since implementation includes carefully designed CUDA\(^3\) kernels. CUDA is a parallel computing platform and programming model developed by NVIDIA \([15]\). For instance, to efficiently compute a matrix multiplication \(Y=WX\) (W is the filter matrix, and X is the input matrix), the proposed approach presented by Coates et al. \([9]\) fully takes advantage of matrix sparseness and local receptive field by extracting non-zero columns in W for neurons that share identical receptive fields, which are then multiplied by the corresponding rows in X. This strategy avoids the situation when the requested memory is larger than the shared memory of the GPU.

GPUs are also being used to implement a model-parallel scheme. Using the same set of input examples, each GPU is used for different part of model optimization and communication occurring through the MVAPICH\(^4\) MPI (Message Passing Interface). The resulting approach is a huge deep learning system that is capable of training with more than 11 billion parameters, but with many fewer machines than Distbelief.

\(^3\)https://developer.nvidia.com/cuda-zone
\(^4\)http://mvapich.cse.ohio-state.edu/
3.3 Data Pre-processing Tools

To integrate the big data streams with DL models requires some pre-processing on the data to get data in proper format. Big Data can be very susceptible to be inconsistent, incomplete, redundant, unreliable, noisy and irrelevant for further analysis. The success of any DL model depends on the quality of the data used. The result of pre-processing is a dataset that can be used as the final training set for machine learning algorithms [24]. There are many Data Pre-processing tools available today. We will be discussing some of them.

3.3.1 WEKA:

The WEKA\(^5\) (Waikato Environment for Knowledge Analysis) is a platform-independent system that provides a comprehensive suite of Java libraries that supports several standard data mining tasks, including data pre-processing, clustering, classification, regression, visualization and feature selection.

WEKA provides data pre-processing capabilities in the form of filters that enable data processing at the instance and attribute value levels. Following are the filters that WEKA uses on the attributes of a dataset [42]:

- General manipulations of attributes: Following is the list of filters for attribute manipulations:

  1. Add/Delete Filter: To insert and delete attributes.

\(^5\)https://www.cs.waikato.ac.nz/ml/weka/
2. Make Indicator Filter: It transforms a nominal attribute to a binary indicator attribute. It is used when a multi-class attribute should be represented by a two-class attribute.

3. Merge Attribute Values Filter: It concatenates attributes values into a single value.


5. Select Filter: It deletes all instances from a dataset that exhibit one of a particular set of nominal attribute values below or above a certain threshold.

6. Replace Missing Values Filter: Each missing value is replaced with the mean (for numeric attributes) or the mode (for nominal attributes).

7. Swap Attribute Values Filter: It swaps the attribute values.

- Transforming numeric attributes: Some data pre-processing filters are specifically for numeric attributes, such as:

  1. Numeric Transform Filter: It transforms all the numeric attributes using a user-specified transformation function.

  2. Discretise Filter: It implements supervised and unsupervised discretization method.

- Feature Selection: It provides three feature selection system, i.e., a locally produced correlation-based technique, the wrapper method, and Relief to select relevant attributes to be included in model induction.
The authors in [23] describe a parallel distributed Weka framework for Big data mining using Spark called DistributedWekaSpark. The framework aims at overcoming the disadvantage of Weka as it only supports sequential single-node execution, which imposes significant limitations in handling Big Data.

The framework is a scalable Big Data Mining toolkit that combines the power of Spark’s processing power with standard Weka’s usability. It utilizes both parallel and distributed execution as it is built on top of Spark that provides fast-in-memory iterative processing.

### 3.3.2 RapidMiner:

RapidMiner\(^6\) is a software platform developed by company Rapid Miner and provides an integrated environment for data preparation, machine learning, text mining, and predictive analysis. It was developed by Ralf Klinkenberg, Ingo Mierswa, and Simon Fischer at Artifical Intelligence Unit of the Technical University of Dortmund in 2001 and was initially known as Yet Another Learning Environment (YALE).

Rapid Miner is a cross-platform software and uses a client/server model with the server offered as Software as a Service or on cloud infrastructures. It supports twenty-two file formats and contains more than 100 learning schemes for regression, classification, and clustering analysis [13].

Radoop\(^7\) is an extension for the RapidMiner data mining tool which provides easy-to-use operators for running distributed processes on Hadoop. RapidMiner provides the

---
\(^6\)https://rapidminer.com/
\(^7\)https://rapidminer.com/products/radoop/
user with a graphical user interface for performing operations like loading, mining, and visualization tasks. Radoop reads data from HDFS and executes various data mining functions. The operators used in Radoop are similar to Apache Mahout\textsuperscript{8}, a library for machine-learning algorithms in Hadoop, and hence, both have the same performance issues. At runtime, the operators are translated to Mahout tasks and executed on Hadoop.

Radoop uses Hive\textsuperscript{9} to perform all data transformations which execute as Map-Reduce jobs on Hadoop. All data transformations are expressed in HiveQL scripts. It includes selecting attributes, generating new attributes, iterating examples, sorting, renaming, type conversions, etc. It also supports aggregations, joining of tables and user-defined functions (UDF) for more complex transformations [1].

### 3.3.3 KNIME

Berthold et al. [28] developed a modular environment called KNIME\textsuperscript{10} (Konstanz Information Miner) that provides easy visual assembly and interactive execution of a data pipeline. It is designed as a teaching, research and collaboration platform, that facilitates simple integration of new algorithms and tools as well as data manipulation or visualization methods in the form of new modules or nodes.

KNIME is written using JAVA and its graphical workflow editor is implemented as an Eclipse plug-in. The architecture of KNIME was designed using three principles:

\textsuperscript{8}https://mahout.apache.org/
\textsuperscript{9}https://hive.apache.org/
\textsuperscript{10}https://www.knime.com/
• Visual, interactive framework: Drag & Drop allows combining data flows from a variety of processing units.

• Modularity: Processing units and data containers should not be dependent on each other to enable easy distribution of computation and allow for the development of different algorithms.

• Easy Expandability: It should be easy to add new processing nodes or views and distribute them through a simple plugin mechanism without the need for complicated install/de-install procedures.

KNIME also has a text plugin [39] to process text and natural language data. It is a combination of natural language processing, text mining and information retrieval. Using this plugin, one can read text data, represent data internally as documents and terms and apply several tagging, filtering, frequency computation, data structure transformation and other pre-processing and analysis tasks on it. It is an intermediary plugin that enables reading, pre-processing and transforming textual data into a numerical representation.

### 3.4 Shortcomings of the existing systems

By understanding the functionalities of the existing system, we will address the following issues:

• The existing data-processing tools do not support all the data types. In our proposed approach, we will be handling various data-types and data formats.
• The existing systems support data from either Hadoop machine or Spark. Our proposed approach will handle data from both the systems.

• The existing systems do not integrate big data systems with deep learning system. In our proposed approach, we will fill the gap and connect both the systems.

• The existing systems are designed to handle only specific tasks such as text classification or image classification. Our proposed approach is code-free and allows the user to quickly setup and perform various experiments.

• The existing systems do not allow users to change the model hyper-parameters. The proposed approach allows the user to modify default values of model hyper-parameters before training to perform various experiments.
Chapter 4

Proposed Approach

In this chapter, we describe our proposed approach to create a development environment to integrate big data architecture frameworks and deep learning models. By using the proposed approach, users can transform the data coming from different big data streams into a format as needed to train deep learning models.

4.1 Our Proposed Approach:

Our proposed approach provides a framework that integrates different big data streams and deep learning models and also enables users to perform various manipulations on the data streams. Using this framework the users can quickly construct and perform various experiments on a subset of the dataset to derive meaningful results from it.
The development environment provides options to the user to select data (batch data/real-time data) from different big data architectures, pre-processing data to transform data into the required format and then training the deep learning models using pre-processed data. The environment also provides an option to record all the steps performed and later allows the user to run them on the entire dataset.

Figure 4.1 shows the architecture of the proposed approach that comprises of the following components:

- **User view**: It provides user interface that enables the users to select and perform different operations.
- **Controller**: It handles different components of the architecture.
- **Data Selector**: It allows the selection of different big data streams.
- **Data Pre-Processor**: It performs different data pre-processing operations.
- **Data Trainer**: It performs the training of deep learning models.

### 4.1.1 User View

User View is an interface that displays the data from different components of the proposed approach architecture and also enables the users to perform various actions such as data selection, data manipulations, and training deep learning models.
4.1.2 Controller

The controller is the crux of our proposed environment. It acts as a bridge between the user view and components (such as Data Selector, Data Pre-processor, and Data Trainer). It interprets the user actions and invokes the corresponding component.

First, the controller calls the Data Selector component to handle the Big data stream selected by the user in the user view for further processing.
Next, the controller forwards the user actions to the Data Pre-processor component to perform the corresponding operations on the data. These data manipulations help to transform the data into the required format for next step.

Later, the controller sends the pre-processed dataset to the Data Trainer component. This dataset is the training dataset to the deep learning model.

### 4.1.3 Data Selector

The Data Selector provides options to select different big data streams from various data sources. This component allows the user to select different types of data such as text, image, video, sound, and time-series. It also handles big data streams from different data sources such as Hadoop Distributed File System (HDFS) for batch data and Apache Storm and Apache Spark for real-time data streams.

![Data Selection Diagram](image)

**Figure 4.2:** Data Selection

The primary tasks of the Data Selector are:
1. It allows selection of different types of data.

2. It allows data streams from different data sources.

3. It allows users to select file or multiple files, to perform further steps.

The final selected data file(s) are then used as input for the data pre-processing step.

4.1.4 Data Pre-processor

The Data Pre-processor provides the user with a library of operations to perform manipulations and transformations on the selected data. Using the Data Pre-processor, the user can convert raw data into a clean dataset in a specific format. It includes tasks such as data-cleaning, normalization, transformation, feature subset reduction, etc.

Data Pre-processor supports the following operations on data as shown in figure 4.3:

1. Data Cleaning: The primary aim of this function is remove inconsistencies, redundant and irrelevant data. It allows to remove missing values, smoothing out noisy data and correct inconsistent data.

2. Data Integration: It enables integration of data from multiple sources (data files) into a single dataset by joining on attribute values.
3. Data Reduction: It uses feature subset selection and sampling to reduce the dataset i.e. generate a compressed version of the entire dataset. It provides the reduction of the dataset i.e. generate a compressed version of entire dataset. It involves feature subset selection and sampling.

4. Data Transformation: It transforms the data into a format as required by deep learning models. It involves normalization, i.e., converting a attribute value (numeric attribute) into a specified range, aggregation, i.e., combining the attributes into one attribute and generalization, i.e., replacing lower-level attribute or primitive(raw) data by higher level concepts.

The result of pre-processing is a dataset that can be used as the final training set for deep learning models.


### 4.1.5 Data Trainer

The Data Trainer component provides the user with options to select the deep learning model to perform different tasks such as classification, prediction, and recommendation. For training deep learning models, data training uses a pre-processed dataset from the data pre-processor. It also provides the user to select values for different hyper-parameters for optimization.

The main tasks of the Data Trainer are:

1. It allows users to select values of model different hyper-parameters such as the number of epochs, batch size, activation function, learning rate, momentum, number of hidden layers, number of neurons.

2. It allows users to select from different deep learning models for different tasks such as classification, prediction, and recommendation.

### 4.1.6 Automate Steps

Once the user is done with all the steps to infer meaningful results from the big data streams, the framework provides an option to the user to save all the steps performed. The saved file contains the start-to-end steps performed by the user such as which big data stream to select, what operations to perform, which task to perform, which deep learning model to select along with model hyper-parameters. Using this stored file, the user will have the option to perform all the steps with just
a single click on the entire dataset, i.e., all the steps will be performed automatically in the background without any user-intervention.
Chapter 5

Prototype

In this chapter, we describe a prototype implementation of our framework. Although the prototype is based on a simplified interpretation of this framework and only offers a subset of the features we described in chapter 4, it demonstrates the potential of the approach.

5.1 Our Prototype

Our prototype is a user-driven web-based application that allows the users to integrate different big data streams with deep learning system. Since it is just a prototype of our proposed development environment, it supports the integration of batch data streams from Apache Hadoop and deep learning models for classification only. Our prototype is a step-by-step system that gives the user an option to select a data stream, perform required manipulations and integrate with the deep
learning system. On every step, the user has the power to select which operation to perform next.

The Data Selector allows the user to select different big data streams from different sources. After selecting the data, the Data Pre-Processor allows the user to perform various pre-processing operations on the data. The pre-processed data is used as a training dataset by the Data Trainer to train deep learning models.

The advantage of the prototype is that it can be used to show a proof of concept (POC) to determine whether data from the selected stream with the performed data transformations will yield meaningful results or not after training deep learning models with it.

5.1.1 Data Selector

The different steps of Data Selector are as follows:

5.1.1.1 User Credentials

The first step of Data Selector is to enable access to the user to select big data streams. The prototype first asks the user to provide the credentials to get the big data stream from the machine. The user provides his user-name, password or private SSH Key along with the IP address of the machine.

Using the credentials provided by the user, we connect to the Hadoop master node machine using Secure Shell (SSH) to get the data stream.
5.1.1.2 Data Selection

After logging into the Hadoop machine, the prototype displays the list of all the file directories of the Hadoop machine. The user can select the files and preview the file contents. The user then selects the file for further steps.

To display the file directory list from the Hadoop System, we query HDFS and fetch all the files and directories and display it to the user.

Once the user clicks on a file, we display the contents of the file by querying HDFS for that file and display limited number records to the user.

When the user selects the file for further processing, we dynamically generate HIVE table using HIVEQL script and load the file contents into the created HIVE table.

5.1.2 Data Pre-processor

The prototype then displays a list of operations, depending on the data-type, that the user can perform on the data as described in section 4.1.4.

5.1.2.1 Operations for Text Data

The different operations supported by the prototype are as follows:
5.1.2.1.1 Data Reduction

The prototype provides an option to reduce the number of attributes, i.e., features from the dataset and also reduce the number of dataset records.

We dynamically generate hive script to drop the selected column(s) from the hive table. We use hive’s LIMIT command to achieve this.

5.1.2.1.2 Data Cleaning

The prototype provides options to change attribute values, adding a new column, sorting, type-conversions, renaming a column and filtering.

For every operation selected by the user, on executing the hive script, a Map-Reduce job is executed on the Hadoop machine.

5.1.2.1.3 Data Integration

The prototype provides an option to integrate attributes from different files into a single file.

We create a hive table that contains selected attributes from the selected files to form a single dataset.
5.1.2.1.4 Data Transformation

The prototype provides options to normalize, aggregate and transform the attributes of the dataset.

We store the data from the hive table, created to perform other operations, in a file. We use scikit-learn\(^1\) library to perform data transformation operations on the file.

All the data manipulations for text data streams will be done on the HIVE table created for the selected file. We dynamically generate HIVE scripts using HIVEQL to perform all the user-specified operation. The HIVE scripts are executed as Map-Reduce jobs on Hadoop machine to perform the required tasks.

5.1.2.2 Operations for Image Data

After selecting the image files for further processing, the prototype provides the user with different operations such as cropping image, rotating image, adjusting brightness, saturation, and hue.

We use tensorflow to perform all transformations on the images.

5.1.3 Data Trainer

After all pre-processing operations are done, the user is given options to build and train deep learning model as discussed in section 4.1.5. The user can select the

\(^1\)http://scikit-learn.org/stable/
deep learning model for different tasks such as classification, prediction, and recommendation. The user can also modify default values of model hyper-parameters. After training is done, we display training loss and validation accuracy graphs.

We use tensorflow\(^2\) to build and train deep learning models. The prototype provides templates for different deep learning models. The model hyper-parameters specified by the user will be used for training deep learning model.

### 5.1.4 Automate Steps

After the user is done with training the deep learning model, the user is given an option to record all the steps performed as discussed in section 4.1.6. Selected file(s), all the data pre-processing operations, model hyper-parameters and selected deep learning model is recorded. The same user-experiment is performed on the entire dataset automatically using these recorded steps.

The prototype records all the steps in JSON file. The values for every user step is extracted from the JSON file and executed in the background without any user intervention.

---

\(^2\)https://www.tensorflow.org/
Chapter 6

Case Study

In this chapter, we will evaluate the prototype of our proposed development environment. We will go through two different scenarios.

6.1 Scenario 1

In this scenario, we perform twitter sentiment analysis, a common text classification task. The Twitter data is stored in Hadoop Distributed File System (HDFS). We perform various operations to get the data in a proper format as required by deep learning model. The deep learning architecture selected to perform this task is Convolutional Neural Network (CNN). Figure 6.1 gives an overview of the entire scenario.
6.1.1 Setup

To start with the process, we have installed Hive on the Hadoop system. If Hive is not installed then the framework will automatically install Hive. To train deep learning models, we need tensorflow on the system training them.

6.1.2 Dataset Used

We use Sentiment140 for Twitter sentiment analysis [16]. Sentiment140 started as a class project from Stanford University. The tweets are collected based on positive and negative emoticons. Any tweet with positive emoticon "\:.\)" is considered as a positive tweet and any tweet with negative emoticon "\:\(\)" is considered as a negative tweet.

6.1.3 Experiment

Every step performed for text classification is as follows:
Step 1:
In this step, the prototype provides the user option to select a data stream to begin the experiment or upload a JSON file to execute the user performed operations in the background for the entire dataset without any user intervention. To begin text classification experiment, we select "Select Data Stream" option.

![Selection screen for experiment](image)

**Figure 6.2:** Selection screen for experiment

Step 2:
In this step, the prototype provides options to the user to select the type of big data stream. For twitter sentiment analysis, we select text data stream.
Step 3:
In this step, the prototype provides options to the user to select the data source to get the big data stream. Since the twitter data is stored in HDFS, we select Hadoop.

Step 4:
We provide the IP address and user credentials (username and SSH Key) of the Hadoop machine to establish a connection. The user can either provide a password
or SSH Key for authentication. Once the user credentials are verified, the controller invokes Data Selector to display the contents of the Hadoop machine to select data.

![Hadoop Machine Details](image)

**Figure 6.5:** Accepts the IP Address and user credentials of the Hadoop machine.

**Step 5:**

The prototype provides the list of directories and files present on the Hadoop machine. The user can view the file contents and decide a file to proceed with.
Step 6:

The prototype provides the user with several options to format or manipulate the selected file as explained in section 4.1.4.
Step 7:
In this step, the prototype provides the user with an option to select the task. For text classification, we select classification option.
Step 8:
In this step, the prototype provides the user with an option to select the deep learning model. For twitter sentiment analysis, we select convolutional neural network.

![Selection Screen for Deep Learning Model](image)

**Figure 6.9:** Selection Screen for Deep Learning Model

Step 9:
Once the user is done with all the data pre-processing, the prototype provides the user the option to modify the default model hyper-parameters. The user can then start the process of Twitter sentiment classification. The prototype displays the training loss and the validation accuracy of the deep learning model after the training of deep learning model is completed.
6.1.4 Evaluation

We have performed end-to-end flow for twitter sentiment analysis. The prototype integrates a text stream from Hadoop machine and performs training on deep learning model system.

We can evaluate the prototype by comparing it with manual steps. If we perform, all the experiment steps manually, then we need to perform the following steps:

1. We have to manually scan every file in the Hadoop system and determine the data file to work with for further steps.
2. After selecting the file, we have to perform all the transformations on the data such as reducing the number of records, removing attributes, adding a new attribute, changing attribute values, filtering and sorting.

3. Once all the transformations are performed, we have to design and code the deep learning model with default values for model hyper-parameters.

4. After setting up the deep learning model, we have to get the pre-processed data file for training deep learning model with it.

5. If we need to change anything in any of the performed steps, then we need to repeat the entire process.

The manual steps can be quite time-consuming and error-prone. Using the prototype, the user can quickly setup experiments with only a few clicks and no-coding. The prototype also allows the user to execute all the recorded steps in the background for the entire dataset.

For this scenario, the manual process took roughly around an hour to perform all the steps, excluding the time for training the deep learning model. However, it took approximately 15 minutes using the proposed environment. Figure 6.11 shows the comparison of manual approach and proposed approach.
Figure 6.11: Comparison of Manual Approach and Proposed Approach for Twitter sentiment analysis

6.2 Scenario 2

In this scenario, we perform classification of MNIST handwritten digits images. The MNIST images are stored on Hadoop Distributed File System (HDFS). We perform various operations to get the data in a proper format as required by deep learning model. The deep learning architecture selected to perform classification task is Convolutional Neural Network (CNN).
6.2.1 Setup

For image data streams, we will read images from Hadoop machine and use tensorflow to perform various pre-processing operations on the image dataset.

6.2.2 Dataset Used

The MNIST (Modified National Institute of Standards and Technology) dataset consists of images of handwritten digits [11]. It includes 60,000 examples for training phase and 10,000 examples of handwritten images for the testing phase. The dataset was created by "re-mixing" the samples from NIST’s (National Institute of Standards and Technology) original datasets. The images in the dataset are size normalized to fit in a 20x20 pixel box, and they are centered in a 28x28 image using the center of mass.

6.2.3 Experiment

We followed the same steps as described in Scenario 1 (6.1.3). However, since we are dealing with image data stream, the only difference is the pre-processing operations performed on image dataset. The different pre-processing operations performed adds artificial data to the image dataset for training the deep learning models. We performed pre-processing operations such as rotating the images, adjusting brightness, hue, and saturation, and cropping images to set the same size for all images.
6.2.4 Evaluation

We performed end-to-end flow to classify MNIST handwritten digits images. The prototype integrates image data stream from Hadoop system and performs training on deep learning model system.

We evaluate the prototype for image stream by comparing it with manual steps. The manual steps will be the same for the image data stream as described in section 6.1.4. The manual steps for image classification are very time-consuming and can be error-prone.

Using the prototype, the user can perform all the manual steps with just a few clicks and without writing any code for a subset of data. The prototype also records all the steps performed by the user. It allows the user to execute all the recorded steps in the background for the entire dataset.

For this scenario, the manual process took more than an hour to perform all the steps, excluding the time for training the deep learning model. However, it took approximately 25 minutes using the proposed environment. Figure 6.12 shows the comparison of manual approach and proposed approach.
6.3 Scenario Observations

With the evolution of Big Data Technologies, deep learning models are implemented distributively. However, for the two scenarios discussed, we are not training deep learning models distributively, instead, we are integrating big data technologies with large-scale deep learning systems. The framework provides an environment to quickly setup and performs various experiments on a small subset of data. Our proposed framework provides the following advantages to the user:

- It provides an intuitive and user-friendly GUI for the user to perform steps.
• It is independent of the data-type.

• It is independent of the data source.

• It provides users with various data pre-processing options.

• It allows the users to modify the default model hyper-parameters.

• It provides the users an option to execute the performed steps automatically in the background for the entire dataset.

Thus, our prototype provides an easy and user-friendly approach to integrate big data streams with deep learning models. We can easily extend our prototype to support real-time data streams with deep learning systems.
Chapter 7

Conclusion

In this thesis, we proposed a development environment that enables integration of different big data streams with deep learning systems. It also allows the user to quickly and easily setup and performs various experiments to infer meaningful information from data. We reviewed the existing work done in the fields of big data and deep learning. Many of the existing solutions either use distributed deep learning models on big data architectures or implement large-scale deep learning systems. We also reviewed the tools available for performing data manipulations. Our proposed environment tries to fill the gap between big data systems and deep learning systems. It provides the ability to integrate various big data streams with deep learning systems. Unlike other approaches, our proposed environment is not dependent on the big data stream or deep learning model. To support our proposed approach, we have also implemented a prototype to show the feasibility and effectiveness of our proposed environment.
We presented two case studies of how our prototype can be used and also discussed some advantages that our prototype provides. Our prototype eliminates all the error-prone manual process of connecting and supporting different big data streams and setting up deep learning model for training. The prototype also provides an option to execute all user steps automatically in the background on the larger dataset without any user intervention.

In future, we hope to integrate the following to the prototype implementation of our proposed development environment:

- Provide support for Real-time data streams: The prototype currently supports Apache Hadoop for batch data and in future we can integrate real-time data streams from Apache Spark and Apache Storm.

- Support training of deep learning models on multi-GPU systems: The prototype currently supports processing on a single CPU/GPU system. In future, the prototype will be extended to support data and model level parallelism on multiple CPU/GPU.

- Support for performing multiple experiments simultaneously: The current prototype executes a single experiment to completion. In future, the prototype will allow execution of multiple experiments simultaneously to enable more rapid exploration of the problem domain.

- Support for processing multiple data-types present in a stream: The current prototype will be extended to enable processing of multiple data-types through out the system.
Conduct user study on the complete development environment: Once the development environment is feature complete, we will conduct a user study to identify the strengths and weaknesses of the current user interface design.

The recent successes of deep learning in self-driving cars and in highly complex games like Go are indicative of the future successes that will be found as deep learning is applied in new domains. Combining data captured in big data systems with deep learning techniques will ease the transition to new areas of research and product development. The development environment demonstrated in this thesis lowers the threshold for creating such systems, making the development process practical for programmers and non-programmers alike.
Bibliography


