ibm.com/redbooks

IBM PowerAI: Deep Learning Unleashed on IBM Power Systems Servers

Dino Quintero

Bing He

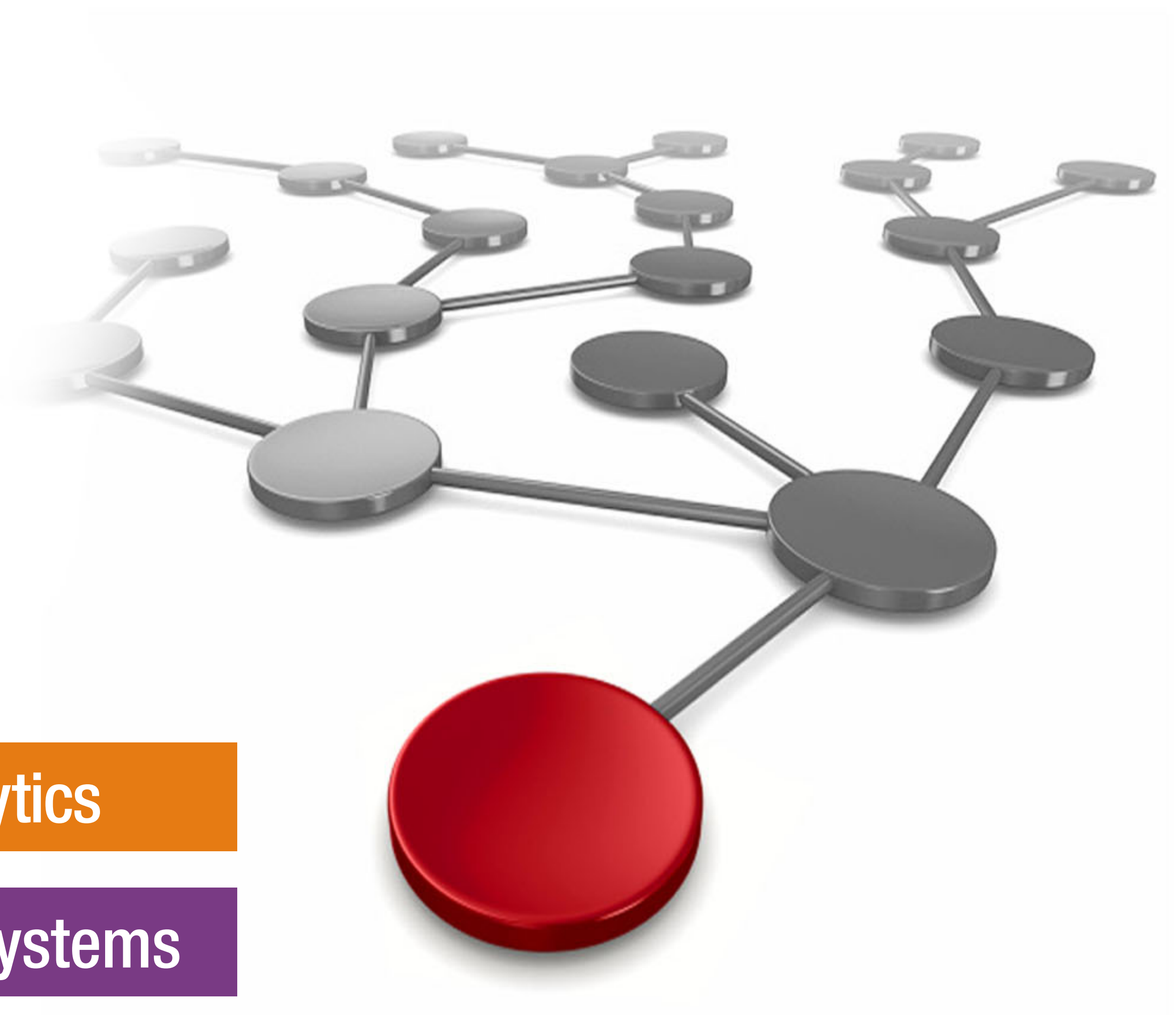
Bruno C. Faria

Alfonso Jara

Chris Parsons

Shota Tsukamoto

Richard Wale



Analytics

Power Systems

Redbooks

International Technical Support Organization

IBM PowerAI: Deep Learning Unleashed on IBM Power Systems Servers

March 2018

SG24-8409-00

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (March 2018)

This edition applies to:
Ubuntu 16.04 for IBM Power Systems
Red Hat Enterprise Linux 7.4 for IBM Power Systems
NVIDIA CUDA Toolkit V8.0.61 and V9.0.176
NVIDIA cuDNN V6.0.20 and V7.1
NVIDIA drivers 384.66
IBM PowerAI V1.4 and V1.5
Berkeley Vision and Learning Center (BVLC) Caffe V1.0.0
IBM Caffe V1.0.0
NVIDIA fork of Caffe V0.15.14
Theano V0.9.0
Torch V7
TensorFlow V1.1.0 and V1.4 RC
Deep Learning GPU Training System (DIGITS) V5
Chainer V1.23.0
IBM Spectrum MPI V10.1.1.0
Anaconda2 V5.0
Python V2

Contents

Notices

Trademarks

Preface

This IBM® Redbooks® publication is a guide about the IBM PowerAI Deep Learning solution. This book provides an introduction to artificial intelligence (AI) and deep learning (DL), IBM PowerAI, and components of IBM PowerAI, deploying IBM PowerAI, guidelines for working with data and creating models, an introduction to IBM Spectrum™ Conductor Deep Learning Impact (DLI), and case scenarios.

IBM PowerAI started as a package of software distributions of many of the major DL software frameworks for model training, such as TensorFlow, Caffe, Torch, Theano, and the associated libraries, such as CUDA Deep Neural Network (cuDNN). The IBM PowerAI software is optimized for performance by using the IBM Power Systems™ servers that are integrated with NVLink. The AI stack foundation starts with servers with accelerators. graphical processing unit (GPU) accelerators are well-suited for the compute-intensive nature of DL training, and servers with the highest CPU to GPU bandwidth, such as IBM Power Systems servers, enable the high-performance data transfer that is required for larger and more complex DL models.

This publication targets technical readers, including developers, IT specialists, systems architects, brand specialist, sales team, and anyone looking for a guide about how to understand the IBM PowerAI Deep Learning architecture, framework configuration, application and workload configuration, and user infrastructure.

Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Dino Quintero is a Solutions Enablement Project Leader and an IBM Level 3 Certified Senior IT Specialist with IBM Redbooks in Poughkeepsie, New York. Dino shares his technical computing passion and expertise by leading teams developing content in the areas of enterprise continuous availability, enterprise systems management, high-performance computing (HPC), cloud computing, and Analytics Solutions. He also is an Open Group Distinguished Technical Specialist. Dino holds a Master of Computing Information Systems degree and a Bachelor of Science degree in Computer Science from Marist College.

Bing He is a Consultant IT Specialist at IBM China. He joined IBM in 2007. He holds a degree in Computer Science from BeiJing JiaoTong University. His areas of expertise include IBM AIX®, performance tuning, IBM PowerVM®, IBM PowerHA®, IBM Geographically Dispersed Resiliency for IBM Power Systems, SAP HANA on Power, and Linux on Power.

Bruno C. Faria is a Data Engineer at IBM Brazil. He has 9 years of experience in the Data and machine learning (ML) fields. He has worked at IBM for 9 years, and dedicated the last 3 years to big data and how to scale and manage large amounts of data and real-time data, and how to apply ML techniques to those systems. His areas of expertise include several SQL and No-SQL databases, ETL, DL, and others.

Alfonso Jara joined IBM Spain in 2001. He has 15+ years of experience designing and deploying highly available and highly virtualized solutions for AIX and IBM pSeries customers. He holds a degree in Physics from Universidad Complutense de Madrid. Today, he is a Certified IT Architect at IBM Global Services, mainly helping to migrate workloads to IBM Cloud, and designing the underlaying infrastructure for some Infrastructure as a Service (IaaS) solutions.

Chris Parsons is a Machine Learning Engineer working at IBM Lab Services in the UK. He has 3 years of experience in the AI field. Chris gained a First-class honors BSc degree in Computer Science at the University of Swansea. He has several areas of expertise, including Mobile Application Development, Web Development, and DevOps. His current focus is DL and TensorFlow applications on IBM Power Systems.

Shota Tsukamoto is in High Performance Data Analytics (HPDA) technical sales at IBM Japan. Since he joined IBM in 2015, he has been in charge of HPDA for Power Systems, and is one of the founding members of the DL team at IBM Japan. He holds a BS degree from Central Washington University. His areas of expertise include IBM Power Systems, Linux (Ubuntu and Red Hat Enterprise Linux), DL frameworks, and genomics applications running on IBM POWER® processor-based servers that are integrated with GPU technology.

Richard Wale is a Senior IT Specialist working at the IBM Hursley Lab, UK. He holds a B.Sc. (Hons) degree in Computer Science from Portsmouth University, England. He has been supporting production IBM AIX systems since 1998. His areas of expertise include IBM Power Systems, PowerVM, AIX, and IBM i.

Thanks to the following people for their contributions to this project:

Wade Wallace
International Technical Support Organization, Austin Center

Minsik Cho, Jason M Furmanek, Cesar Maciel, Scott Soutter
IBM US

De Gao DG Chu, Lin BJ Dong, Zhi Wen Fu, Qing AG Li, Yu Y Song, Kai Yuan KY Wu, Ze Ming ZM Zhao
IBM China

Junfeng JF Liu, Kelvin Lui, Helena H Krolak, Raymond Zhao
IBM Canada

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  `ibm.com/redbooks`

- Send your comments in an email to:

  `redbooks@us.ibm.com`

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  `http://www.facebook.com/IBMRedbooks`

- Follow us on Twitter:

  `http://twitter.com/ibmredbooks`

- Look for us on LinkedIn:

  `http://www.linkedin.com/groups?home=&gid=2130806`

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  `https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm`

- Stay current on recent Redbooks publications with RSS Feeds:

  `http://www.redbooks.ibm.com/rss.html`

1

Introduction to artificial intelligence and deep learning

This chapter describes essential concepts about deep learning (DL) and neural networks. This chapter also describes the most used and known neural networks architectures, their purposes, and how they started.

This chapter contains the following topics:

- Deep learning
- Neural networks overview
- Deep learning frameworks

1.1 Deep learning

Humans have always searched for a way to describe reality and find solutions for all kinds of problems. From the use of mechanical machines to facilitate and perform the hardest physical tasks to the construction of magnificent mathematical systems to solve problems that initially existed only deep in the mind and can be expressed only through complex mathematical language, humans have seemed to find their way to solve problems across three distinct paths: experimentation, equation solving (analytical solution), and mechanical computation (numerical solution).

Although these paths seem to be distant from each other, every age since the development of mathematics, a few humans have dreamed of finding a way to merge all of these paths, mainly through human-created devices that can develop a certain degree of intelligence.

The 20th century brought the dawn of a new machine. Computers could compute a large number of numbers so that humans could solve complex mathematical systems at an unimaginable scale.

The most promising approach to these problems is artificial intelligence (AI). AI is a scientific concept that is glorified and promoted in literature and films. However, in recent decades, science fiction has steadily moved closer to becoming science fact, and in everyday places in our modern world. Although AI remains a field of science, certain characteristics that were previously labeled as science fiction were defined by subsequent fields and areas of scientific study.

Probably the biggest field in AI is machine learning (ML), which can be described as a scientific representation of decision making, that is, the ability to represent a human decision process in mathematical form so that it can be implemented as a program (or algorithm) that a computer can run against a given data set. To prove and establish the algorithm, a computer must be taught or trained so that learning is based on experience (as is the case in the human context) to enable the system to interpret results and improve the accuracy of results. ML is closely related to statistics, and has grown as large amounts of information are made available through digitalization and computer storage.

For example, imagine a data set of thousands of images from a traffic camera. An algorithm can be written to count the number of vehicles that is captured over time. The algorithm can be improved to count only the number of cars (ignoring trucks and bikes). The results can then be processed again to determine whether a certain car can be considered blue.

Before you consider the detection of color, an initial problem is to define an algorithm to determine whether a certain image was captured and depicted a vehicle (and if so, what type). Although it is obvious to the human eye, the more combinations and permutations that can be handled with more training and algorithms, the more accurate the results become. The size of the vehicle, the time of day of the photograph, weather conditions, and density of objects (think of peak rush hour), and other variables can reduce accuracy. Consider a photograph of a road junction that is taken at peak morning rush hour, in winter, during a snow storm. The initial decision of “is there a vehicle in the photograph?” becomes a more challenging task even for human eyes.

DL is a subset of ML that attempts to model decision making by using neural networks, which mimic the way the human brain itself processes senses (sight, sound, and others).

This book covers several topics about DL and in particular the deployment, use, and optimization of IBM PowerAI, which is a set of tools that enables data scientists and systems administrations to access the power of DL with little effort, and get results in a shorter time.

Figure 1-1 shows how these fields relate to each other.

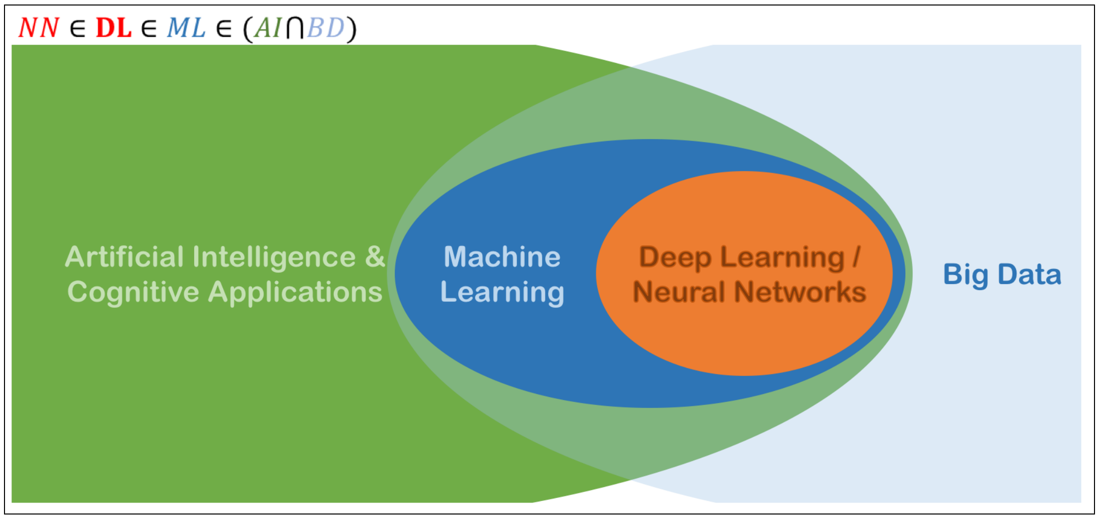

$NN \in \mathbf{DL} \in ML \in (AI \cap BD)$
Artificial Intelligence & Cognitive Applications
Machine Learning
Deep Learning / Neural Networks
Big Data

Figure 1-1 Deep learning in the artificial intelligence landscape

1.1.1 Artificial intelligence milestones and the development of deep learning

Deep Learning[1] suggests that there are three eras in the history of DL:

- 1940s -1960s: Cybernetics
- 1980s -1990s: Connectionism
- 2006 - present: DL

These three eras show that DL is an established discipline with roots in the beginning of computer science that was subject to trends during its history. The history shows that each wave was pushed by certain developments of the era.

The current trends in DL increased because of the availability of large amounts of digital data (closely related to the big data discipline), and the availability of inexpensive computing power in the form of graphical processing units (GPUs) with a high degree of specialization in solving complex algebraic problems.

To put this idea in a wider context, Table 1-1 provides a timeline of some of the most important milestones in this discipline. The boundaries between these disciplines are difficult to perceive clearly, so many of the articles or books can apply to one or more of the fields.

Table 1-1 Milestones for each discipline

Date (CE)	Artificial intelligence	Machine learning	Deep learning
c. 1300	Ramon Llull builds the first mechanical machine that can perform basic calculations.		
1623	Wilhem Schickard builds the first calculating machine.		
c. 1700	Gottfried Leibniz extends the concept of the calculating machine.		
1937	Alan Turing's paper On Computable Numbers, with an Application to the Entscheidungs problem, establishes the foundation of the Theory of Computation.[a]		

1 http://www.deeplearningbook.org

Date (CE)	Artificial intelligence	Machine learning	Deep learning
1943	McCullouch and Pitts' formal design for Turing-complete artificial neurons is the first paper published that covers an AI topic.[b] This text marks the beginning of the development of theories of biological learning.		
1956	The AI research field is born as a workshop at Dartmouth College.		
1958	The perceptron algorithm is invented by Frank Rosenblatt,[c] enabling the training of a single neuron.		
1965			The first general, working learning algorithm for supervised, deep, feed forward, and multilayer perceptrons (MLPs) is published by Ivakhnenko and Lapa.[d]
1980	Commercial use of expert systems grow.		
1986			Rina Dechter introduces the term *deep learning*.[e]
1986			Rumelhart, et al, propose the concept of back-propagation[f] to train neural networks through one or two hidden layers.
1990		A simple ML algorithm can determine whether a Cesarean section is recommended.[g]	
1990s		The ML discipline flourishes because of the increasing availability of digitalized information.	
1997	IBM Deep Blue defeats the reigning Chess World Champion Gary Kasparov.		
2000			Igor Aizemberg, et al., coin the term *Artificial Neural Network*.[h]
2005			Gomez and Schmidhuber publish a paper establishing the concept of reinforced learning.[i]
2006			Hinton, et al, introduce the concept of supervised back propagation in the context of deep neural networks.

Date (CE)	Artificial intelligence	Machine learning	Deep learning
2010		IBM Watson® defeats human contestants in a Jeopardy! quiz exhibition show.[j]	
2015			AlphaGo beats the reigning Go World Champion.[k]

a. https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf
b. https://www.cs.cmu.edu/~epxing/Class/10715/reading/McCulloch.and.Pitts.pdf
c. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf
d. Ivakhnenko, A. G., “Cybernetic Predicting Devices”, CCM Information Corporation, 1973
e. . https://www.aaai.org/Papers/AAAI/1986/AAAI86-029.pdf
f. https://www.iro.umontreal.ca/~vincentp/ift3395/lectures/backprop_old.pdf
g. https://www.ncbi.nlm.nih.gov/pubmed/2342742
h. Aizemberg, et al, “Multi-Valued and Universal Binary Neurons: Theory, Learning, and Applications”. Springer Science & Business Media, 2000
i. http://people.idsia.ch/~tino/papers/gomez.gecco05.pdf
j. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.99.2714&rep=rep1&type=pdf
k. https://www.nature.com/articles/nature16961

1.2 Neural networks overview

This section describes neural networks.

1.2.1 A brief history about neural networks

Artificial neural networks have been around for some time, and they have evolved over time. Today, neural networks play an important role in the AI realm, as described in 1.2.2, “Why neural networks are an important subject” on page 6.

According to Yadav,[2] neural network history can be divided into four greater ages starting in the 1940s when the first article about the subject was published in 1943. It described the fact that neural networks can compute any arithmetic or logical function. Then, researchers looked into brain-like methods of learning as a promising way to create learning algorithms.

The 1950s and 1960s were marked by the first neurocomputer, which was called Snark, and was created by Marvin Minsky in 1951. What characterized these years as the first golden age of the neural networks was the first successful neurocomputer that changed the direction of what we know today. The neurocomputer was created in 1957 - 58 by Frank Rosenblatt, Charles Wightman, and others.

Unfortunately, by the end of 1960s, neural networks studies faltered because most of the researchers were working on experimental studies that were not applicable in real scenarios.

During the 1970s, researches continued experimenting in the areas of adaptive signal processing, biological modeling, and pattern recognition. Many of these scientists who started their work became the ones who were responsible for reviving the neural network field about one decade later.

[2] Yadav, et al, “History of Neural Networks. In: An Introduction to Neural Network Methods for Differential Equations”, Springer Briefs in Applied Sciences and Technology, Springer, Dordrecht, 2015

In the 1980s, after years of research, new proposals started to be submitted that focused on neurocomputers and neural networks applications. During 1983 and 1986, a physicist named John Hopfield became involved with neural networks and published two papers that motivated several experts in the area to become involved with neural networks.

In 1986 when the Parallel Distributed Processing (PDP) books[3] were published, neural networks became a top subject in research areas, and have been constantly researched since then.

1.2.2 Why neural networks are an important subject

Most neural network proposals are around theories and how to improve them, even when researching about their applicability and coming up with successful results. Until recently, there was not enough machine, processing, and storage power to support neural network training.

To train a neural network to recognize and classify different objects, you must use many images of the objects to achieve good accuracy, and it is even more important than for real-life scenarios. For example, in an image recognition field, large quantities of images representing types of dogs are required, although equally important are images that are not dogs to train the neural network in a suitable way.

Figure 1-2 shows how storage capacity grew two times from the 1950s to the1980s and more than 20,000 times from the 1980s to today.

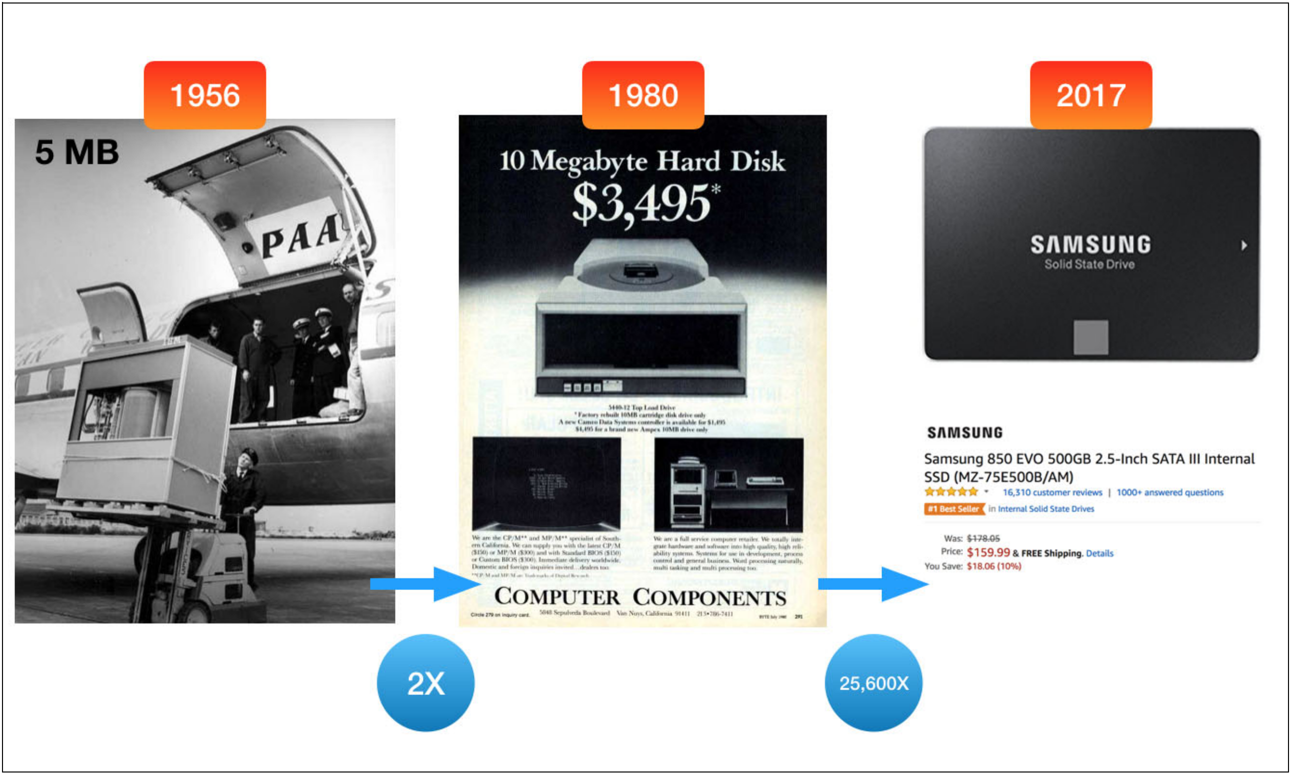

1956
5 MB
PAA
1980
10 Megabyte Hard Disk
$3,495*
COMPUTER COMPONENTS
2017
SAMSUNG
Solid State Drive
SAMSUNG
Samsung 850 EVO 500GB 2.5-Inch SATA III Internal SSD (MZ-75E500B/AM)
16,310 customer reviews | 1000+ answered questions
#1 Best Seller in Internal Solid State Drives
Was: $178.05
Price: $159.99 & FREE Shipping. Details
You Save: $18.06 (10%)
2X
25,600X

Figure 1-2 Storage capacity growth

[3] https://mitpress.mit.edu/books/parallel-distributed-processing

Also, the price dropped exponentially (Figure 1-3). For example, in 1980, the 305 RAMAC was leased for USD 3,200 a month, which is about USD 27,287 in 2016.[4] These changes in storage contributed to the scenario today to where you can start using neural networks and extract their benefits in real-world applications.

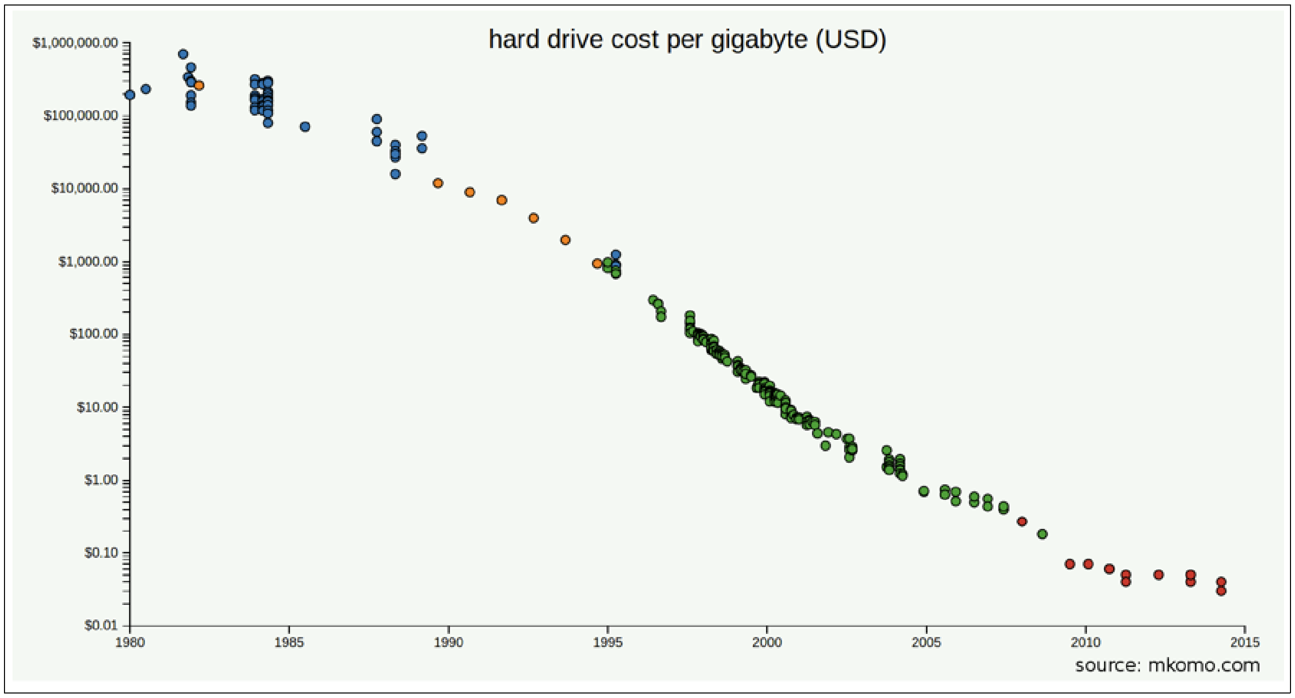


Figure 1-3 Graph showing the decrease of hard disk drive costs

Besides storage, you must have several epochs around the data (loops through the data) to adjust all the neural networks weights and biases so that you can ensure higher accuracy rate. Processing power has become increasingly accessible and cheaper.

4 https://en.wikipedia.org/wiki/IBM_305_RAMAC

Figure 1-4 shows how processing has grown and is about to surpass human brain capacity.

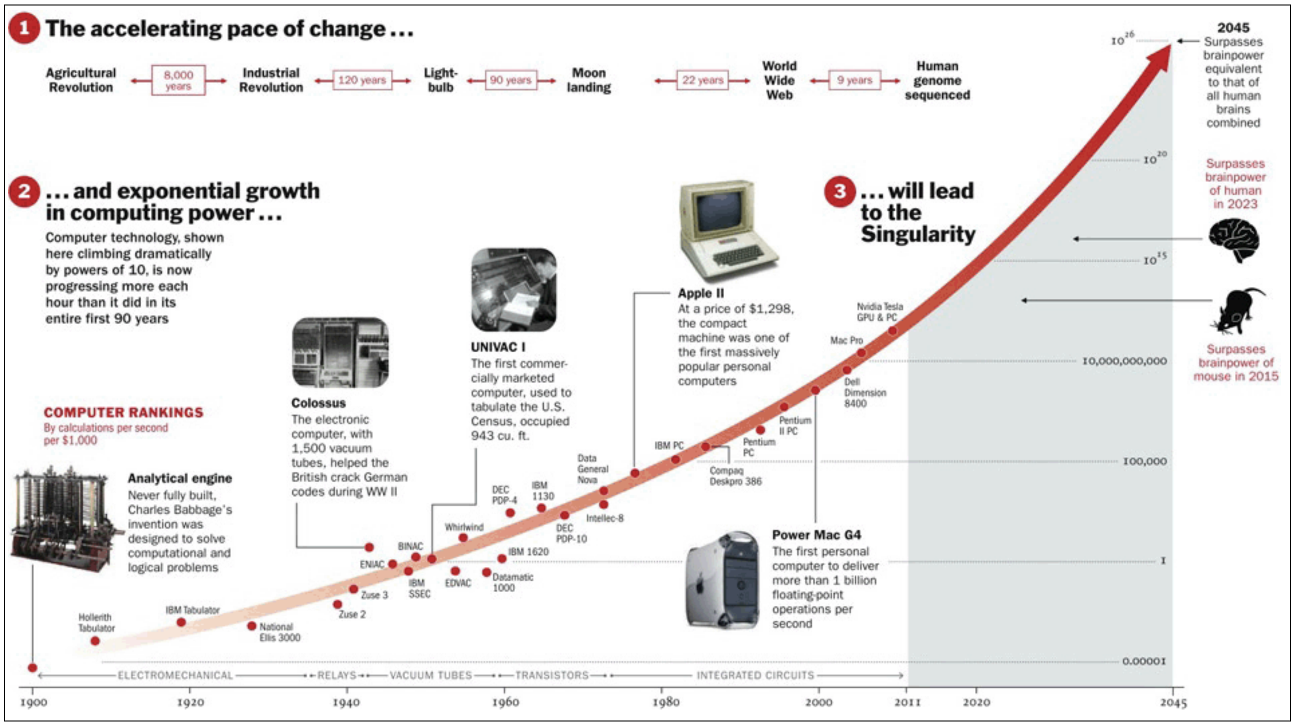


Figure 1-4 Processing capacity growth

In this current context, neural networks use CPU and GPU processing because of distributed architecture, which enables the system to scale horizontally by using on-premises or cloud environments. Storage can also be used in a distributed fashion, which increases the speed of data access by using the principle of locality and memory, reducing disk I/O and network traffic.

1.2.3 Types of neural networks and their usage

There are several different types or architectures of neural networks. This section describes fundamentals and briefly reviews the main architectures.

As the name implies, neural networks were inspired by the human brain, and the idea is to imitate how the brain works. Scientists still have much to learn about the human brain, and the little knowledge that is available so far is enough to produce some interesting theories and implementations for an *artificial neural network*.

In the human brain, there are billions of neurons, and they signal each other through a structure that is called a *synapse*. Every neuron is a single cell that is composed of a cell body (that holds the cell nucleus), an *axon* (a long tail that is insulated from the environment), and many *dendrites* (smaller branches of the cell wall that are around the cell body and at the end of the axon). The synapse can be compared to a chemical bridge between two neurons. The synapse is the gap between two dendrites of two different neurons, and works as a chemical gate that can be opened or closed to allow or prevent the electrical signal from jumping from one neuron to another one.

Neural networks are inspired by the synpase structure and how it passes and processes signals. By using this structure as a base, scientists can implement a new approach to data processing, where different simple elements (neurons) are part of a bigger and more complex structure (neural network) that can perform much more complex tasks.

In this scenario, a neural network is an artificial structure that is composed of artificial neurons that are arranged so that they form layers with a level of specialization. A neuron in a certain layer receives the inputs from a previous layer, processes them (that is, weights all input signals to generate an output signal), and passes the result to the adjacent neuron in the next layer.

Figure 1-5 shows how an artificial neuron receives several inputs and processes them according to a function (f(S)) to produce an output that is propagated to the next layer.

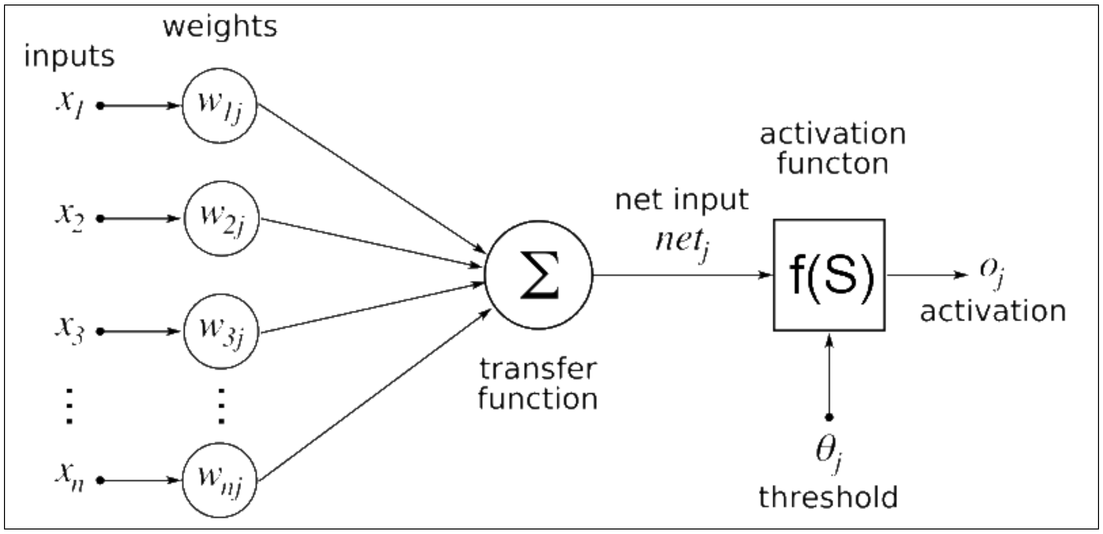


Figure 1-5 Signal processing in a neuron

The network expects some values as an input, so there must be input neurons. All neural networks must have output, so there also must be output neurons.

The set of input neurons is called the input layer, and the set of output neurons is called the output layer. In the brain, there are several neurons communicating to each other and passing information through them; in the artificial neural network, there are neurons between the input and the output layer, which are collectively known as the hidden layer. The following section describes the hidden layers.

Figure 1-6 illustrates a simple neural network with one hidden layer and eight neurons.

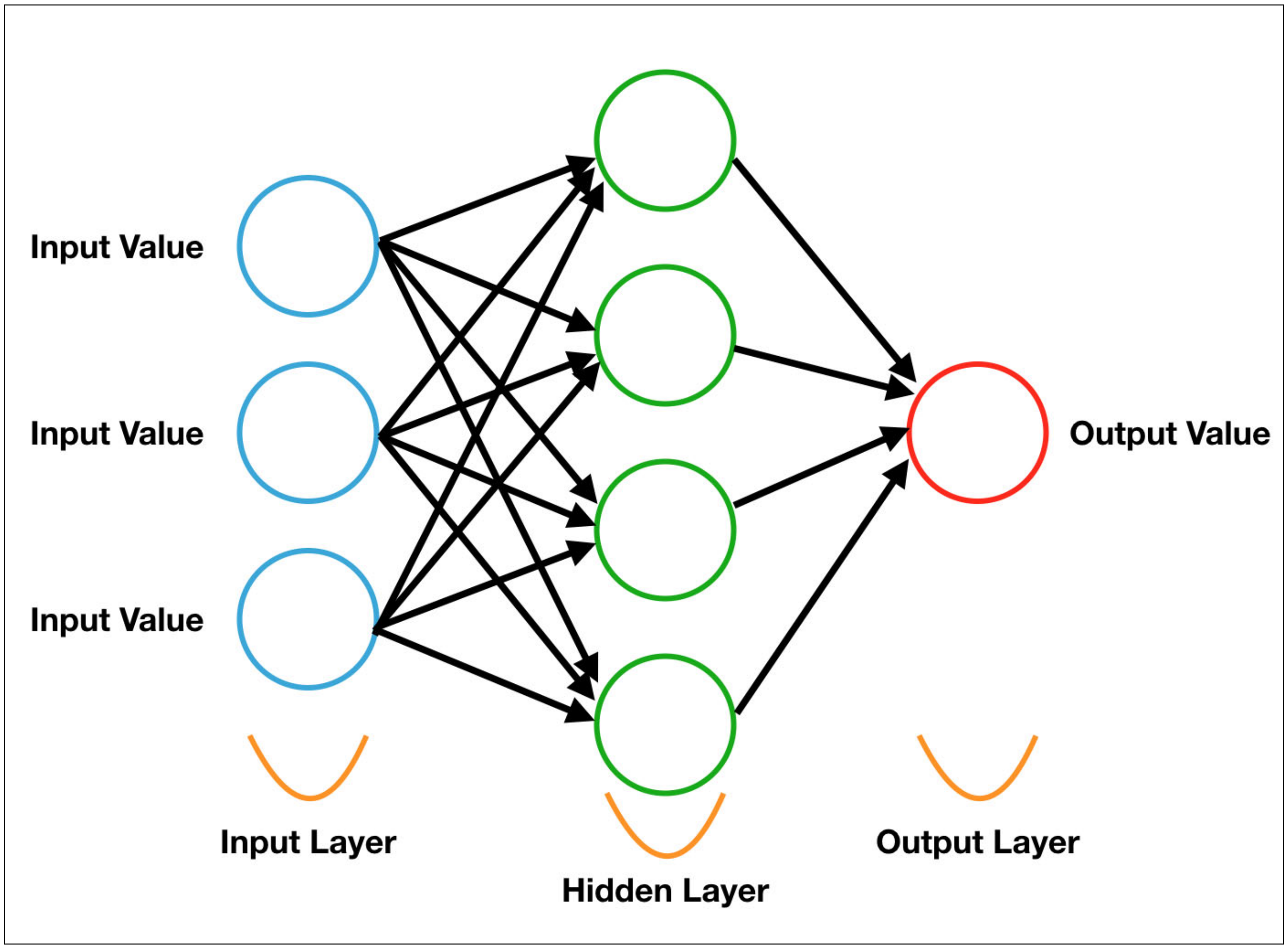


Figure 1-6 Simple feed forward neural network

After you have your neural network structure defined, you adjust each neuron weight after each training step to conform to the provided label. For each type of neural network, there is a specific behavior and way to train and handle its neurons weights and biases. The following section outlines the most common neural network architectures.

1.2.4 Neural network architectures

This section describes some of the most common neural network architectures.

Feed forward neural network

A feed forward neural network (FFNN) is one of the most common neural network architectures. Data comes into the input layer and is propagated forward through the hidden layer until it reaches the output layer with the value it comes up with based on its training.

The FFNN is trained by using a back-propagation algorithm. Basically, for each input, an expected value is provided to the neural network and the output is compared against this value. The difference from the expected value and the one delivered by the neural network (which can be the Mean Square Error or other cost function) is then back-propagated across all the neurons, and an optimization algorithm is used to adjust each neuron's weight aiming to reduce the error that is found.

This process happens for the number of epochs that is defined for the training phase or until the error rate converges, which is decided by the data scientist.

For more information about FFNN, see “The Perceptron: A probabilistic model for information storage and organization in the brain” by Rosenblatt.[5]

Recurrent neural network

Recurrent neural networks (RNNs) use sequential information. In an FFNN, the inputs and outputs are independent of each other. An RNN is different because they need to know their previous input to *understand* the sequence, and make suggestions about that sequence. RNNs are called recurrent because they go through the same processing (layers) for every element in a specific sequence. In a certain way, RNNs can store the data that is calculated so far.

An RNN has two kinds of input: the new input, also known as the present one, and the recent past that happens through a feedback loop returning its previous decision back into the neurons in the hidden layer. These outputs are combined to output the new value.

Figure 1-7 illustrates how an RNN works.

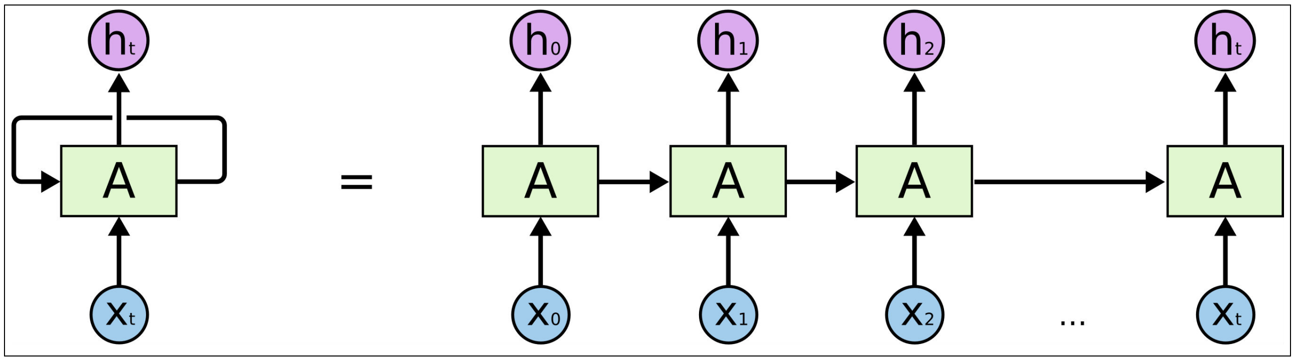


Figure 1-7   How a recurrent neural network works

RNNs are used in many fields, especially the natural language processing (NLP) field because you need to know the sequence of words in a sentence to take actions such as predicting the next word or understanding the sentence’s semantic content. RNN also allows the creation of a language model that can be used to generate text or help with word suggestions and algorithms for language conversion.

One of the limitations of this neural network model is that it can remember only short sequences, and it cannot output values that are based on inputs from the beginning of a large sequence. This limitation was addressed by a variation of the RNN called long / short term memory (LSTM), which is described in “Long and Short Term Memory” on page 12.

For more information, see “Finding structure in time” by Elman.

[5] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf

Long and Short Term Memory

The LSTM network is a specific type of RNN. LSTM is more robust and aims to address the limitations of classical RNNs. LSTM stores long-term memory and can provide relevant outputs about information that is provided far behind in a sequence.

With LSTM, it is possible to preserve error generation across several steps in the loop so that the RNN keeps learning with the input provided that is far behind. This behavior enables the neural network to make relationships and point out connections between remote things.

Using what is called gates, information flowing through the neural network can be stored by creating a *memory* within each cell. Those cells enable the network to perform reads and writes or even erase its content to manage what is useful for the given learning context.

Those gates that are responsible for deciding whether the information is kept, altered, or removed use a sigmoid function, which is similar to a classical neural network, and activates that gate based on its signal strengths that manage the cells' weights. This process is repeated and the weights are adjusted by the RNN learning method.

With this type of neural network, it is possible to understand and infer words or sentences based on a complete book writing style, which leads to several other applications, such as handwriting recognition, music composition, and grammar learning.

For more on LSTM, see "Long short-term memory." by Hochreiter and Schmidhuber.[6]

Convolutional neural networks

Convolutional neural networks (CNNs) have become more prominent because of their effectiveness in areas such image recognition and classification. CNNs can identify different objects within the same image and know where in the image that object is.

In a CNN, there are a few basic operations that are essential to how they work:

- The *convolutional* phase focuses on extracting features from the input object (in most cases, those objects are images) by creating a filter (or kernel) that goes through this object piece by piece, performs a multiplication of the elements, and then sums the result and puts it into a new and smaller generated matrix that is called the *feature map*. In a simple grayscale image (numerical matrix), imagine a filter being a smaller matrix that is applied over the original one at the upper left corner that performs the calculation, moves one pixel to the right, and starts the calculation again until the filter has covered the whole matrix.

[6] https://crl.ucsd.edu/~elman/Papers/fsit.pdf

Figure 1-8 and Figure 1-9 illustrate the process.

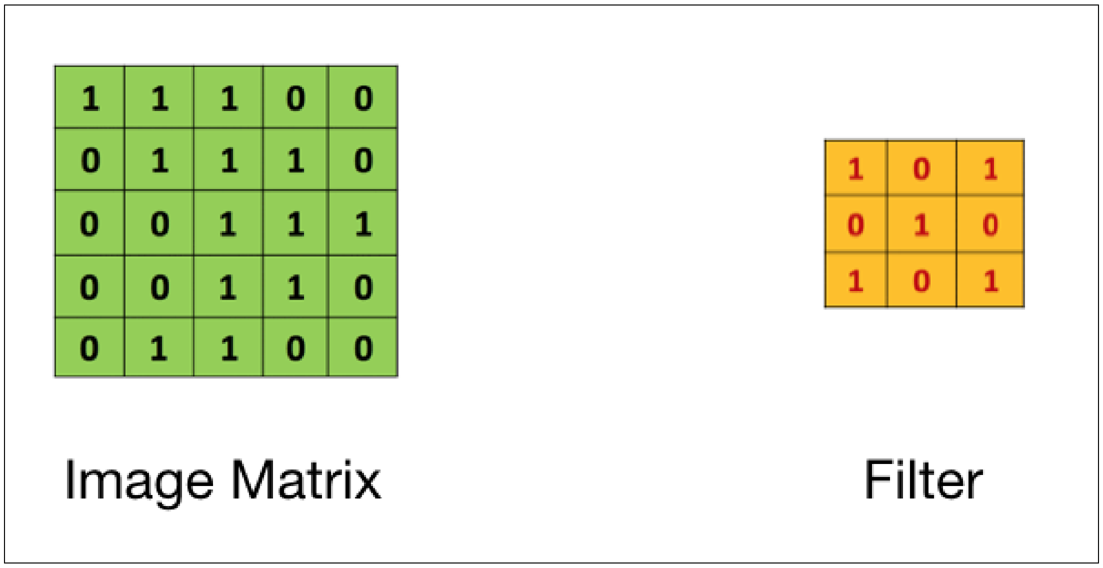


Figure 1-8 Image pixel matrix (simplified) and filter

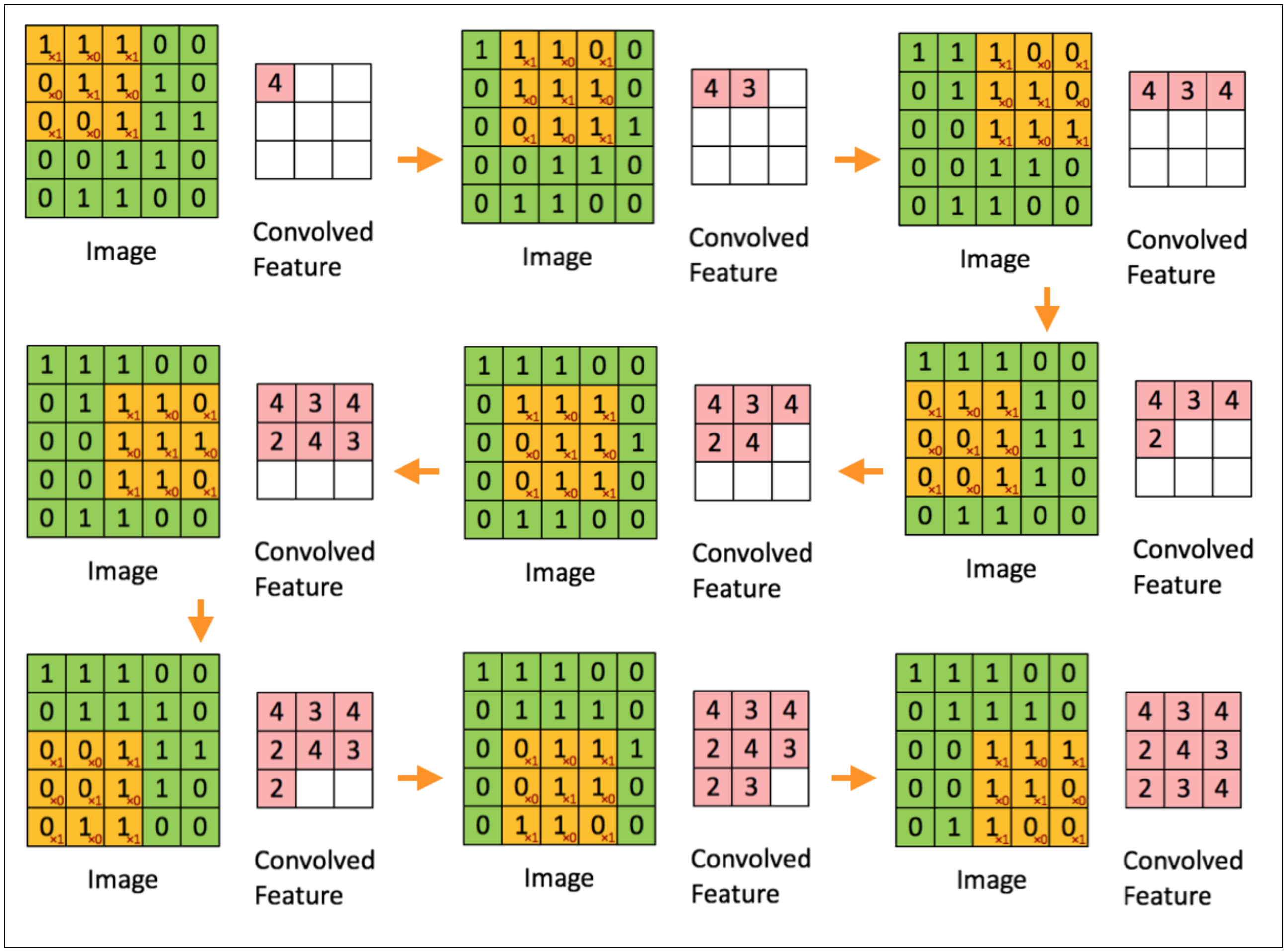


Figure 1-9 Convolutional process example

The filter is initialized randomly before a CNN is trained, and during the training process, it is adjusted and its size is defined by whoever is designing the CNN based on their needs.

After the convolution, it is necessary to add non-linearity to the result because most of the data and applications are non-linear and the convolutional process generates linear results. In Figure 1-10, all the negative pixels are turned to 0 by using an operation called rectified linear unit (ReLU).

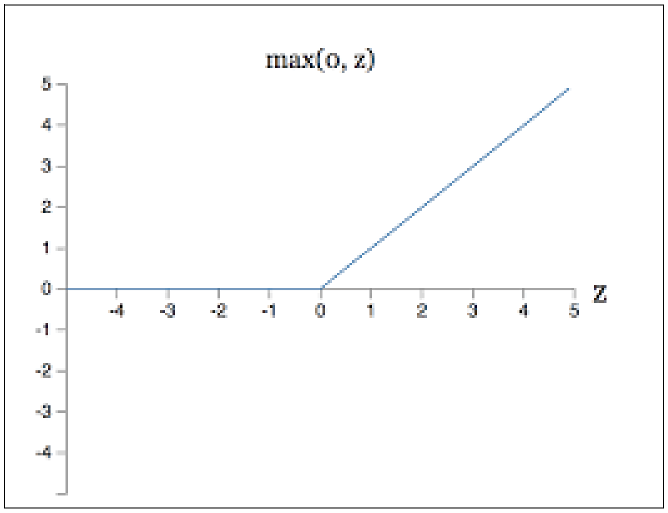


Figure 1-10 Rectified linear unit graph

- The next phase, pooling, reduces the dimensions of the feature map that are generated in the convolutional phase and then rectifies the previous phase. This process is known as *downsampling*. Even though the dimensionality is reduced, the most important information is retained.

There are different methods of pooling, such as extracting the average of a specific group or getting the highest number. This process is similar to the convolutional process where a specific area is defined and then moved across (without overlapping) and the highest number is kept (in the max pooling scenario), as shown in Figure 1-11.

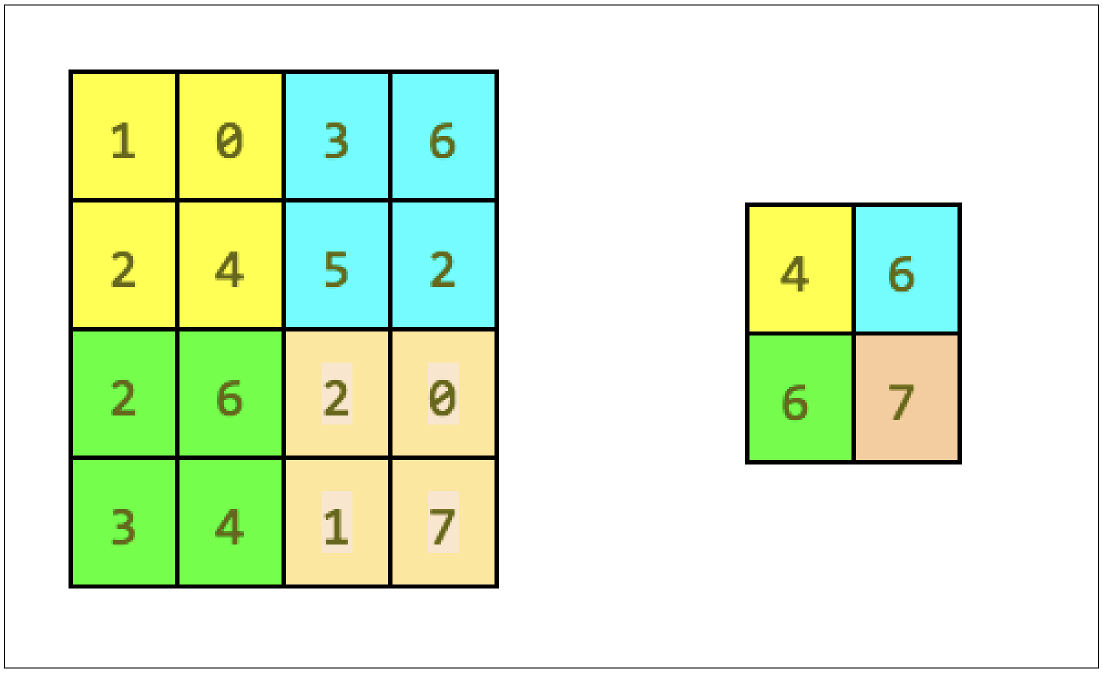


Figure 1-11 Max pooling process example

- Finally comes the last phase, the fully connected layer. This is a classical FFNN with a specific activation function at the output layer (usually a softmax one). The idea is to learn and classify the features that were extracted in the previous phases. It is called fully connected because as in an FFNN, all neurons from a layer are connected to all neurons in the next layer.

  It is worth noting that the convolutional, rectified, and pooling phases can be repeated before reaching the fully connected phase to extract even more features from the object.

  During the training phase, a back-propagation algorithm is used to apply a gradient descent across all the fully connected neurons and into the convolutional filters to adjust their weights and values.

1.2.5 Difference between a classical and deep neural networks

The answer to the question “What is the difference between classical and deep neural networks?” is straightforward. A deep neural network has more than one hidden layer in its architecture, usually even more than two or three.

It is possible to have deep neural networks in many different neural network models, such as a deep FFNN or a deep convolutional neural network, as shown in Figure 1-12.

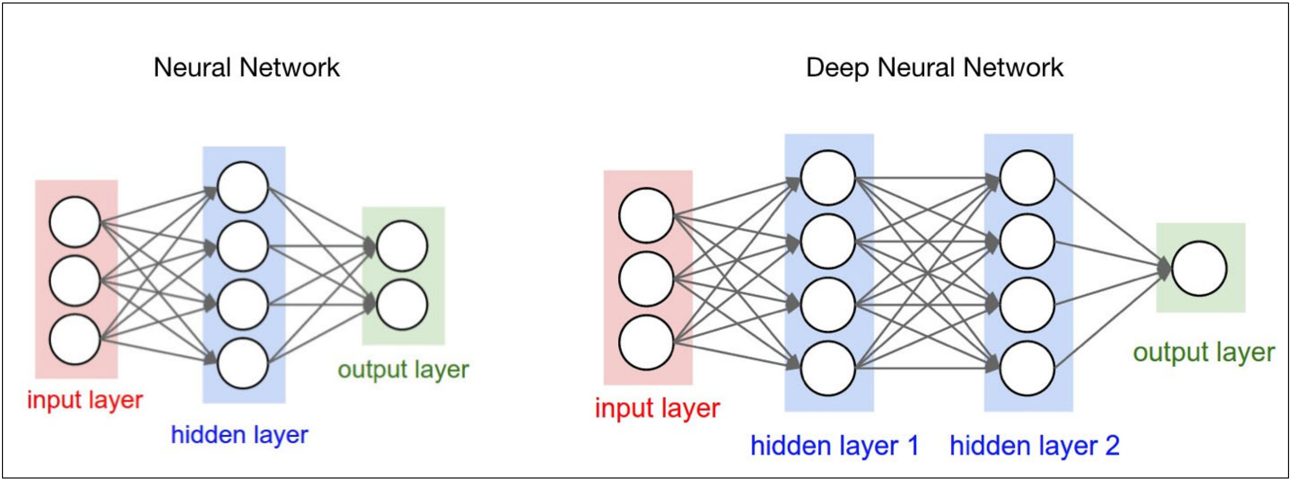


Figure 1-12 Difference between a classical neural network and a deep neural network

So, why use more than one hidden layer, and what are the advantages of doing so?

Using only one hidden layer has proved that it is possible to approximate any function based on the error that is generated.[7] However, the point is that is most approximations in more complex situations, neural networks with one hidden layer do not have accuracies greater than 99%. Even though there can be many reasons for that situation, and there are many studies and research in this area, one thing that has been tried and shown successful results is using extra hidden layers.

Adding more hidden layers to a neural network enables it to better generalize unknown data (not in the training set). This characteristic is essential and helps greatly with the image and object recognition areas; with more hidden layers, it is possible to recognize a set of features in a better way.

However, deep neural networks require much more processing power and powerful hardware that can reach those levels of processing.

[7] https://en.wikipedia.org/wiki/Universal_approximation_theorem

1.2.6 Neural networks versus classical machine learning algorithms

Both neural networks (in addition to DL) and classical ML offer ways to train models and classify data.

If you take pictures of a car and a motorcycle and show them to any human being, you expect them to be able to answer which one is the car and which one is the motorcycle. The reason is that they have seen many cars and motorcycles in their lives so that they know the specific characteristics and shape of each one. A machine tries to learn by studying objects characteristics and behaviors of things and becoming aware the same way humans do. However, machines and humans can fail to identify things sometimes because they are not familiar with them or do not know specific things about it.

For a machine to do a classical ML classification of an image, you must select manually its relevant features to train that specific model (logistic regression or any other). Those features can then be used by the model to identify and classify the image to train this kind of model, and also to perform predictions on it later. The data (images in this case) must be prepared from all the features that were previously extracted to feed the model.

By using a neural network, there is no need to use code to extract those features from the image beforehand because you provide the image directly to the neural network algorithm (the image layers, for example red, green, and blue (RGB) must be sent before they go into the model).

Even though neural networks seem more promising, they require much machine power and large data sets to make the model effective.

When choosing classical ML algorithms, it is possible to train your model in several classifiers and check which one produces the best results, and if needed, work with more than one in conjunction to achieve even more. You also need to know the best features to extract so that the model can perform its best for your data set.

Usually, for more common applications where you already have most of the features, for example, revenue prediction, it might be easier and faster to work with classical ML algorithms. Especially for non-structured data such as sounds, text, images, and others, neural networks can be straightforward even though they require more processing and in many cases, a great deal of data.

It is possible to use neural networks for classical regression and classification because they provide better accuracy[8] when trained with the expected amount of data. In several cases for simple classification or regressions, there is not much data to use in a training set, which makes neural networks performance insufficient.

If you do not have that much data or lack machine power, it is better to stick with classical ML algorithms; otherwise, add neural networks options to your stack and check which one performs better for your specific scenario.

[8] http://bit.ly/2jaQsSx

1.3 Deep learning frameworks

Implementing neural networks algorithms is not an easy task because it usually requires several lines of code to create a simple neural network, and it probably lacks several features. Besides the neural network logic with all its neurons and connections, it is necessary to think about how to distribute the heavy training process across different machines and make them run on CPUs and GPUs.

This is where DL frameworks come into play. They take care of most of the complexities that are involved in running neural network algorithms, and also all the code that is necessary to interact with the GPU library. They also do many other things besides implementing those algorithms, such as providing higher-level APIs to make the DL code easier to implement.

1.3.1 Most popular deep learning frameworks

There are several DL frameworks, and each one has their own specific characteristics, advantages, and disadvantages. This section introduces and describes the most popular learning frameworks.

Caffe

Caffe is one of the first DL frameworks, even though it is not that old. Caffe was created at Berkeley University by Yangqing Jia during his PhD years. Caffe was developed in C++ and it supports Python.

Today, Caffe is stable and suited for production, although it is also used in research. Caffe performs well when the area is computer vision, such as image recognition, but it lacks in other DL areas.

The framework also works with GPUs that use the CUDA Deep Neural Network (cuDNN) library. The code can be found on GitHub.[9] Unfortunately, it still lacks documentation.

Chainer

Chainer is a flexible framework that enables a complex neural network architecture in a simpler way.

Chainer implements a *Define-by-Run* methodology, which means that the network that Chainer creates is defined dynamically during run time. This approach is similar to writing in any language a simple FOR loop. A FOR loop runs the same way within the network as it runs in a normal code.

The framework was developed in Python and there are no other interfaces. As mentioned on the official website.[10] Chainer works by representing a network as an execution path on a computational graph where this computational graph is a series of function applications. Each function is a layer of a neural network, and they are connected by using links where the variables are, and are updated during the training process.

Chainer is successful framework, although other frameworks are capable for applicable models. Chainer has a well documented website[11] and several examples on the web. It is possible to run Chainer on GPUs by using the CuPy library, which can be easily installed.

[9] https://github.com/BVLC/caffe
[10] https://docs.chainer.org/en/stable/tutorial/basic.html
[11] https://chainer.org/

TensorFlow

TensorFlow started in 2011 as a proprietary system that was called DistBelief that was developed by Google Brain. Due to its success across the company's applications, Google focused on improving DistBelief, which then became TensorFlow. The first open source version was released in 2017.

TensorFlow is one of the well-known DL frameworks, and its usage has increased over time because it is sponsored by Google, and it comes with several useful features:

- TensorFlow works in a *Define-and-Run* scheme where the compiler goes through all the code to create a static graph and then runs that graph. An advantage of this approach is that during the graph creation, TensorFlow can optimize all the operations that are defined and the best way to run each one of them.
- TensorFlow was developed in C++ and has an interface for Python. TensorFlow has an interesting feature that is called TensorBoard, which is a GUI that you can use to visualize the graphs that you created and how data is flowing through.
- The framework supports GPUs and runs in a distributed mode. Running in Define-and-Run mode introduces a lack of flexibility because the graph is static, but this mode also enables a better way to distribute the code across different nodes, ensures that the graph can be shipped, and runs equally in all of the GPUs.

Theano

Theano is a numerical computation library like TensorFlow. Its development began at the University of Montreal in 2007, so it is one of the older frameworks. It is developed in Python, which is its only supported interface.

Similar to TensorFlow, Theano also works in a Define-and-Run scheme where the graph is built first during compile time, and then all the processes flow through the graph during run time. Theano supports working with GPUs and can run in a distributed mode.

Although the framework is stable and ready for use in production, in September 2017, the development team announced that Theano's development was stopping after release 1.0.[12] At the time of writing, the future of Theano is unknown. Theano is open source, so it is possible that its development might continue under new management.

Torch and PyTorch

Even though these two frameworks look alike and have the same C/C++ engine, they are not the same.

Torch came first, and it its core engine was developed in C/C++ with interfaces for Lua and LuaJIT as its scripting language. Torch was developed by research teams with Facebook, Twitter, and Google.

PyTorch was developed by Facebook and released in 2017. The main purpose of PyTorch is to make Torch available with a Python interface because Lua is a language that is not often used among DL researchers and developers. Even though that was the main purpose, PyTorch was designed in a way that Python tightly integrates with the Torch core engine that is developed in C, improving memory management and optimization.

PyTorch works in a Defined-by-Run scheme similar to Chainer and focus on speed processing.

Torch and PyTorch both have a modular way of working, they are easy to combine with each other, not that difficult to run by using GPUs, and have several pre-trained models.

[12] https://groups.google.com/forum/#!msg/theano-users/7Poq8BZutbY/rNCIfvAEAwAJ

1.3.2 A final word on deep learning frameworks

There is not a *best* framework. The choice of framework depends on your need and context. At a high level, PyTorch and TensorFlow have been growing and becoming *fit-all* frameworks. Caffe has been around for some time, is stable, and good for production, but specializes in computer vision. Theano is stable, one of the oldest, might be going away, and unless you already have Theano implementations, you should analyze it carefully before starting projects with this framework. Chainer has a great deal of documentation, has been the inspiration for PyTorch, and looks promising, but is not growing as fast as TensorFlow.

The frameworks that are mentioned in this chapter are all supported by IBM PowerAI and are optimized to run on IBM Power Systems machines.

2

Introduction and overview of IBM PowerAI

This chapter provides introductory information about IBM PowerAI.

This chapter contains the following topics:

- What is IBM PowerAI
- Why IBM PowerAI simplifies adoption of deep learning
- IBM unique capabilities
- Extra integrations that are available for IBM PowerAI

2.1 What is IBM PowerAI

IBM PowerAI is a package of software distributions for many of the major deep learning (DL) software frameworks for model training, such as TensorFlow, Caffe, Chainer, Torch, and Theano, and their associated libraries, such as CUDA Deep Neural Network (cuDNN), and nvCaffe. They are extensions that take advantage of accelerators, for example, nvCaffe is NVIDIA extension to Caffe so that it can work on graphical processing units (GPU). As with nvCaffe, IBM has an own extension to Caffe, which is called IBM Caffe. Furthermore, the IBM PowerAI solution is optimized for performance by using the NVLink-based IBM POWER8® server, the IBM Power S822LC for High Performance Computing server, and its successor, the IBM Power System AC922 for High Performance Computing server. The stack also comes with supporting libraries, such as Deep Learning GPU Training System (DIGITS), OpenBLAS, Bazel, and NVIDIA Collective Communications Library (NCCL).

2.1.1 Contents of IBM PowerAI (IBM PowerAI Release 4)

The IBM PowerAI package is released with all of the upgraded versions of each framework and libraries. However, under special circumstances, such as with an increase in demand, targets for the next release vary. Table 2-1 shows the IBM PowerAI software stack.

Table 2-1 IBM PowerAI software stack

IBM PowerAI software stack (basic)		
Deep learning frameworks		
Core	TensorFlow	IBM Caffe
Other frameworks	BLVC Caffe	nvCaffe
	Chainer	Torch
	Theano	N/A
Libraries and other components		
Major components	DIGITS	NCCL
	OpenBLAS	Bazel
Distributed DL	IBM Distributed Deep Learning (DDL) Library for IBM Caffe	
	TensorFlow Operator for IBM DDL Library	

2.1.2 Minimum hardware requirement for IBM PowerAI

IBM PowerAI can be run on POWER8 and IBM POWER9™ servers. However, to run the DL platform, IBM recommends the following configuration in those servers:

- Two POWER8 CPUs
- 128 GB of memory
- NVIDIA Tesla P100 or V100 with NVLink GPUs
- NVIDIA NVLink interface to Tesla GPUs

You can download at no cost the current version of IBM PowerAI from Index of /software/server/POWER/Linux/mldl/ubuntun.

Note: The instruction guide for IBM PowerAI can be found in the readme file.

Additionally, if you want to download specific versions of frameworks from previous releases, you can download them from Index of software/server/POWER/Linux/mldl/ubuntu/archive. This website provides a readme file and direct download links for the following releases:

- Release 3.0
- Release 3.1
- Release 3.2
- Release 3.3
- Release 3.4
- Release 4.0

Tip: You can confirm the contents of each release to specify the versions of the frameworks within each release package in 3.2, “IBM PowerAI compatibility matrix” on page 58.

2.2 Why IBM PowerAI simplifies adoption of deep learning

Today, organizations are using DL to develop powerful new analytic capabilities spanning multiple usage patterns, from computer vision and object detection, to improved human computer interaction through natural language processing (NLP), to sophisticated anomaly detection capabilities. At the heart of any use case that is associated with DL are sophisticated pattern recognition and classification capabilities that serve as the birthplace for revolutionary applications and insights of the future. However, in situations where organizations try to expand their area for DL or to start working on the development of DL, there are enormous difficulties, especially the performance issues that are caused by hardware limitations, and time-consuming process in each framework regarding setup, tuning, upgrades, and others.

In the fourth quarter of 2016, IBM announced a significant revamp of IBM PowerAI, seeking to address some of the bigger challenges facing developers and data scientists. The goals were to cut down the time that is required for artificial intelligence (AI) system training, making and running a snap with an enterprise-ready software distribution for DL and AI, and simplifying the development experience. The idea behind IBM PowerAI is to embrace and extend the open source community, embrace and extend the capability and creativity that is happening there, and add IBM unique capabilities. This situation manifests itself in a number of value differentiators for AI applications that IBM Power Systems brings to the table.

IBM PowerAI provides the following benefits:

- Fast time to deploy a DL environment so that clients can get to work immediately:
  - Simplified installation in usually less than 1 hour
  - Precompiled DL libraries, including all required files
- Optimized performance so users can capture value sooner:
  - Built for IBM Power Systems servers with NVLink CPUs and NVIDIA GPUs, delivering performance unattainable elsewhere
  - Distributed DL, taking advantage of parallel processing
- Designed for enterprise deployments:
  - Multitenancy supporting multiple users and lines of business (LOBs)
  - Centralized management and monitoring by integrations with other software
- IBM service and support for the entire solution, including the open source DL frameworks

2.3 IBM unique capabilities

This section highlights the unique capabilities of IBM products and solutions that you can take advantage with IBM PowerAI, which includes the following items:

- NVLink and NVLink 2.0
- Distributed Deep Learning (DDL)
- Large Model Support (LMS)

2.3.1 NVLink and NVLink 2.0

Announced in 2014, NVLink is the NVIDIA proprietary high-speed, low-latency physical interconnect. NVLink improves on the traditional communication bottleneck between multiple GPUs, enabling efficient and performance communication between GPUs and system memory. Additionally, NVLink provides more performant and efficient communication between system CPUs and GPUs.

The initial generation of NVLink was implemented with the NVIDIA GPU based on their Pascal-microarchitecture. It provides higher bandwidth than the third generation of the industry-standard Peripheral Component Interconnect Express (PCIe) interface. NVLink was first made commercially available in September 2016 with the launch of the IBM Power System S822LC for High Performance Computing server. This model is available in configurations with either 2 or 4 NVIDIA Tesla P100 GPUs.

The implementation of NVLink in this IBM POWER8 system provides a bidirectional path between GPUs of up to 180 GBps and a bidirectional path between the GPUs and CPUs of up to 80 GBps.

In systems that are configured with more than one GPU, the benefits that are provided by NVLink increase and scale. Each extra GPU adds a separate interconnect between itself and neighboring GPUs and between itself and the nearest CPU.

In 2017, NVIDIA announced the second generation of NVLink with the introduction of its Volta-microarchitecture. It provides improved data rates of up to 300 GBps. The POWER9 processor-based IBM Power System AC922 for High Performance Computing server launched in December 2017 with the second generation of NVLink, supporting configurations of up to six NVIDIA GPUs.

Note: For more information about the implementation and architecture of NVLink, including performance metrics, see the NVDIA NVLink website.

2.3.2 Power AI Distributed Deep Learning

IBM PowerAI DDL is a software/hardware co-optimized distributed DL system that can achieve near-linear scaling up to 256 GPUs. This algorithm is encapsulated as a communication library that is called DDL that provides communication APIs for implementing DL communication patterns across multiple DL frameworks.

The inherent challenge with distributed DL systems is that as the number of learners increases, the amount of computation decreases at the same time the amount of communication remains constant. This intuitively results in unwanted communication ratios.

To mitigate the impact of this scaling problem, IBM PowerAI DDL uses an innovative multi-ring communication algorithm that balances the communication latency and communication impact. This algorithm adapts to the hierarchy of communication bandwidths, including intranode, internode, and interrack, within any system. This implementation allows for adaptation, which enables IBM PowerAI DDL to deliver to the *optimal* distributed DL solution for an environment.

The current implementation of IBM PowerAI DDL is based on IBM Spectrum MPI because IBM Spectrum MPI provides many of the required functions, such as scheduling processes and communication primitives in a portable, efficient, and mature software infrastructure. IBM Spectrum MPI specifically provides functions and optimizations to IBM Power Systems and InfiniBand network technology.

IBM PowerAI DDL **`objects()`** are the foundation of IBM PowerAI DDL. Each **`object()`** instance combines both the data and relevant metadata. The data structure component itself can contain various payloads, such as a tensor of gradients. The metadata provides a definition of the host, type of device (GPU along others), device identifier, and the type of memory where the data structure is. All of the functions within IBM PowerAI DDL operate on these objects, which are in effect synonymous with variables.

In a 256 GPU environment, about 90 GB of data must be transmitted to perform a simple reduction operation, and the same amount of data must be transmitted to copy the result to all the GPUs. Even a fast network connection with a 10 GBps transfer rate bringing this data to a single Parameter Server (PS) can take 9 seconds. IBM PowerAI DDL performs the entire reduction and distribution of 350 MB in less than 100 ms by using a communication scheme that is optimized for the specific network topology.

A key differentiator for IBM PowerAI DDL is its ability to optimize the communication scheme based on the bandwidth of each available link, topology, and latency for each segment. In a heterogeneous environment, with multiple link speeds, IBM PowerAI DDL adjusts its dimensions to use the fastest available link, thus preventing the slowest link from dominating the run time.

Using IBM PowerAI DDL on cluster of 64 IBM S822LC for High Performance Computing servers, we demonstrated a validation accuracy of 33.8% for Resnet-101 on Imagenet 22k in ~7 hours (Figure 2-1). For comparison, Microsoft ADAM[1] and Google DistBelief[2] were not able to reach 30% validation accuracy for this data set.

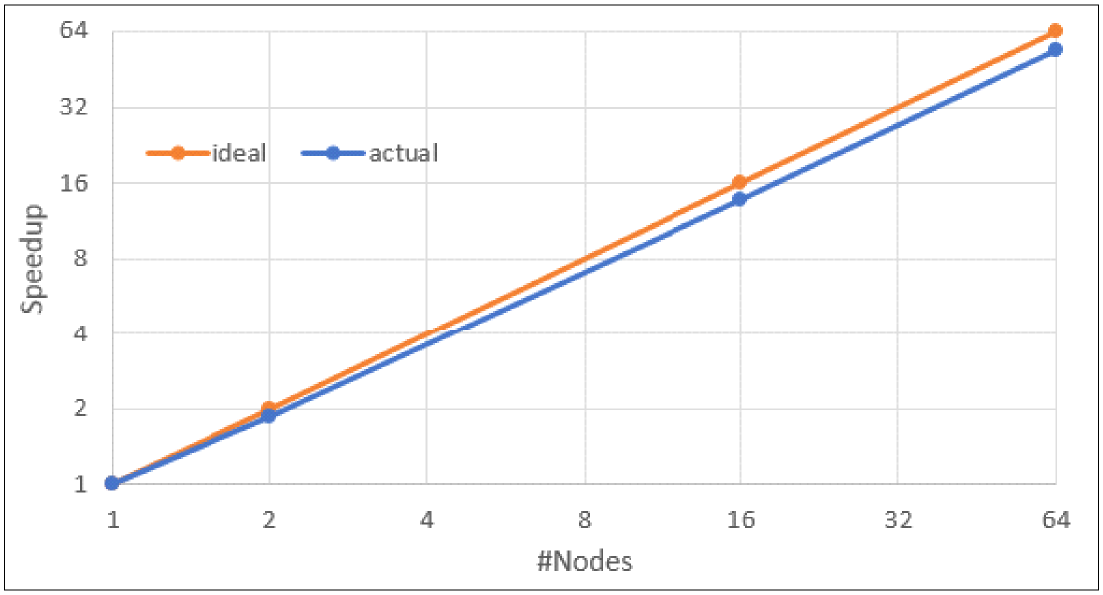


Figure 2-1 Resnet-101 for 22k classes using up to 256 GPUs

2.3.3 Large Model Support

When defining and training large neural networks that are prevalent in the DL space for tasks such as high definition image recognition, large batch size tasks, and highly deep neural networks, the amount of GPU memory that is available can be a significant limitation. To overcome this challenge in IBM PowerAI V1.4, IBM provides LMS, which uses main system memory with the GPU memory to train these large-scale neural networks.

LMS enables you to process and train larger models that do not fit within the GPU memory space that is available currently. LMS uses the GPU memory and a fast interconnect (NVLink) to accommodate such large models and fetch only the working set in each instance to ensure the fastest computation possible while operating with a larger batch size outside the GPU memory address space.

To accommodate such models that occupy both system and GPU memory, LMS performs many CPU to GPU copy operations. There is regular transfer of data from system to GPU memory during the training phase. To perform this operation without inducing delays that impact computation time, the system requires a high-bandwidth interconnect between the CPU and the GPU. On IBM Power Systems servers, LMS can take advantage of the higher bandwidth interconnect that is provided by NVLink, which is described in 2.3.1, “NVLink and NVLink 2.0” on page 24.

[1] Trishul Chilimbi, et al. “Project Adam: building an efficient and scalable Deep Learning training system”, OSDI’14 Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation.

[2] Dean, et al. “Large-Scale Distributed Deep Networks. Advances in Neural Information Processing Systems (NIPS)”, Neural Information Processing Systems Foundation, Inc., 2012.

Figure 2-2 shows the connections between the layers of a deep neural network and demonstrates the challenge and inherent complexity that is derived when trying to train a system at this scale, which LMS seeks to alleviate.

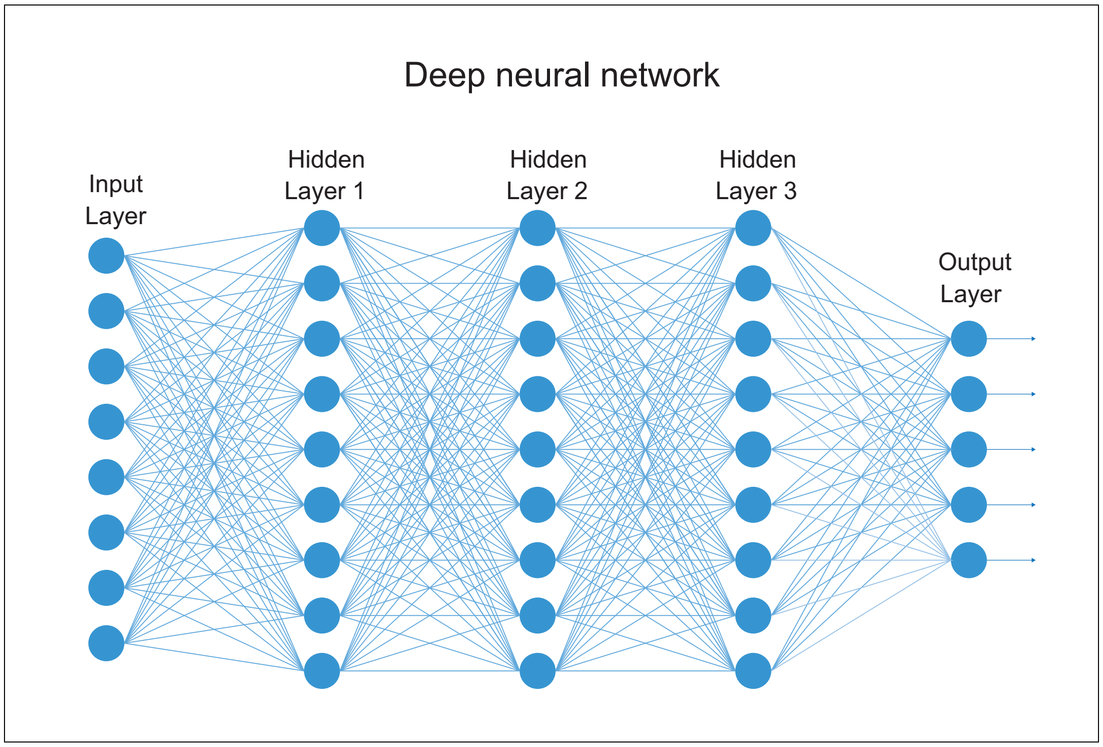


Figure 2-2 Neural networks diagram

To use LMS within IBM Caffe, add the following flag during the run time:

```
-lms <size in KB>
```

This flag sets the threshold size for determining which memory allocations must happen on the CPU memory, and which must be on the GPU memory. For example:

```
-lms 1028
```

When you use this parameter, any memory chunk larger than 1028 KB is processed on the main system memory address space by the CPU, and only fetched to GPU memory when required for computation. So, smaller values result in a more aggressive use of LMS and a greater amount of main system memory is used. However, large values, such as 100,000,000, essentially disable the LMS feature and ensure that all chunks are allocated to GPU memory.

This parameter can be used to control the performance tradeoff because more data being brought in from the CPU during the run time results in a runtime impact.

An alternative way to use LMS within your environment is by using the **`lms_frac`** flag:

```
-lms_frac <x> [0 < x < 1.0]
```

In this instance, LMS does not use system memory for allocating chunks of data until a certain threshold of GPU memory is allocated. For example:

```
-lms_frac 0.75
```

Until more than 75% of the GPU memory that is available is allocated, the system continues to assign chunks of any size to the GPU. The advantage to this method is that you can essentially disable the LMS function for faster training of smaller networks, or have more efficient utilization of GPU memory for larger networks.

2.4 Extra integrations that are available for IBM PowerAI

This section highlights the availability of extra integrations with other IBM products, which gives you more benefits for using IBM PowerAI. This section includes the following products:

- IBM Data Science Experience (IBM DSX)
- IBM PowerAI Vision (technology preview)
- Spectrum Conductor Deep Learning Impact (DLI)

2.4.1 IBM Data Science Experience

IBM DSX is a infrastructure of open source-based frameworks, libraries, and tools for scientists to develop algorithms and validate, deploy, and collaborate with communities of scientists and developers. IBM DSX enables data scientists to develop algorithms in their most preferred language, integrated developer environment (IDE), and libraries.

In September 2017, IBM announced that IBM DSX can be used on IBM Power Systems servers, which enables data scientists to take advantage of IBM PowerAI to deliver performance advantages for DL workloads, as shown in Figure 2-3 on page 29. IBM PowerAI is also integrated with Jupyter notebook, which supports R, Python, and Scala Spark APIs, and no separate installation or configuration is required to use libraries in IBM PowerAI.

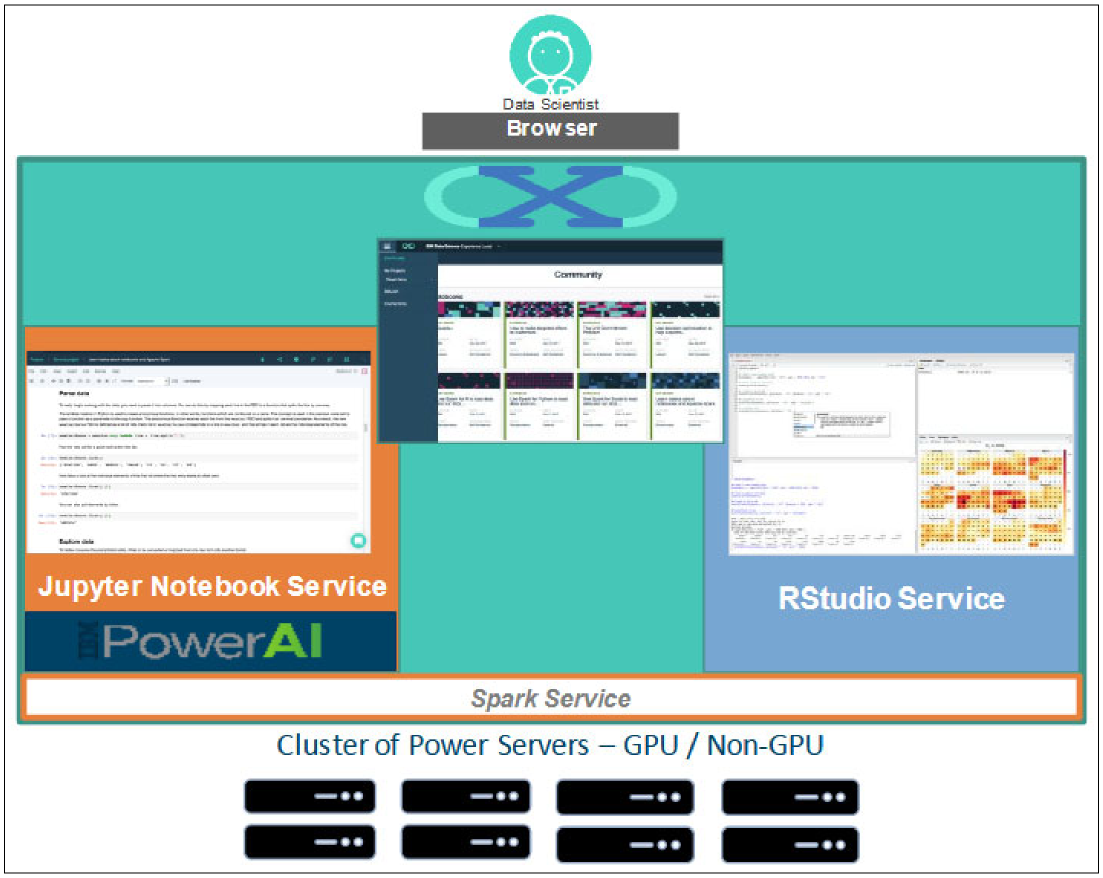


Figure 2-3 IBM Data Science Experience on Power Systems layers

Even though IBM DSX is a software product that can be integrated with IBM PowerAI, the installation method is different from the other two software products that are introduced in this section. In fact, IBM DSX is a package software that contains IBM PowerAI. Therefore, after IBM DSX is installed in the environment, IBM PowerAI is also already in the system and ready to use.

System requirements

There are two default installation options that an IBM DSX local customer can choose to install: a three-server cluster or a nine-server cluster. The two common requirements regardless of the number of servers are the following ones:

- IBM DSX local requires dedicated servers (virtual machines (VMs) or baremetal).
- IBM DSX local runs on RHEL7.3.[3]

Three-node cluster hardware recommendations

Each server runs control, storage, and compute operations. A minimum of three servers is required, each with a minimum of 20 cores, 64 GB memory, and 500 GB of attached storage, plus an extra 2 TB of auxiliary storage.

[3] As of December 2017

Nine-node cluster hardware recommendations

Here are the hardware recommendations for the nine node cluster:

- A minimum of three servers are required to run control operations.
- Each must have a minimum of 8 cores, 16 GB memory, and 500 GB of attached storage.
- A minimum of three servers are required to run storage operations.
- Each must have a minimum of 16 cores, 32 GB memory, and 500 GB of attached storage.
- A minimum of three servers are required to run compute operations.
- Each must have a minimum of 20 cores, 64 GB memory, and 500 GB of attached storage.

The three- and nine-node cluster installations are starting points. After they are installed, you can add and move nodes as required.

Installation details are documented at Install Data Science Experience Local.

2.4.2 IBM PowerAI Vision (technology preview)

IBM PowerAI Vision provides a complete platform for image analytics to create data sets, label them, train a model, and validate and deploy them. The tool makes the journey from raw data to a deployed model faster, and does not impose a need for data scientists. IBM PowerAI Vision realizes rapid data labeling from images, creating labeled data sets from videos, and semi-auto labeling of video streams for enhancing data sets that are required for training, as shown in Figure 2-4.

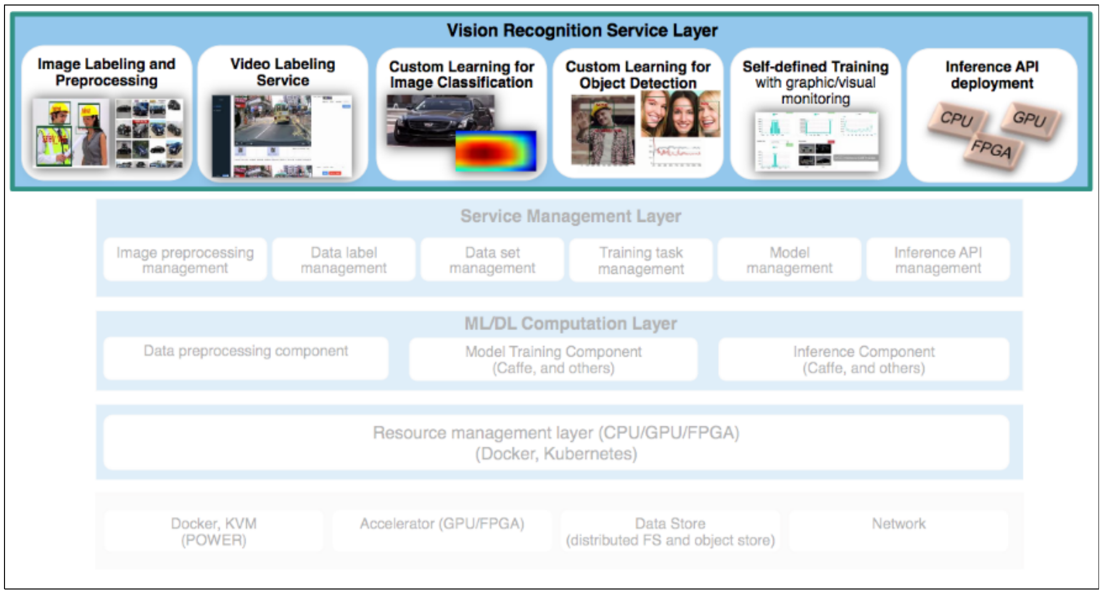

Vision Recognition Service Layer
Image Labeling and Preprocessing
Video Labeling Service
Custom Learning for Image Classification
Custom Learning for Object Detection
Self-defined Training with graphic/visual monitoring
Inference API deployment
CPU
GPU
FPGA
Service Management Layer
Image preprocessing management
Data label management
Data set management
Training task management
Model management
Inference API management
ML/DL Computation Layer
Data preprocessing component
Model Training Component (Caffe, and others)
Inference Component (Caffe, and others)
Resource management layer (CPU/GPU/FPGA) (Docker, Kubernetes)
Docker, KVM (POWER)
Accelerator (GPU/FPGA)
Data Store (distributed FS and object store)
Network

Figure 2-4 IBM PowerAI Vision layer

Hardware prerequisites

Here are the minimum hardware prerequisites for IBM PowerAI Vision:

- CPU: POWER8 or OpenPOWER infrastructure with a minimum of eight hardware cores
- Memory: 128 GB or larger

- GPU:
  - At least one NVIDIA Tesla GPU
  - GPU memory of one GPU core greater than 4 GB
- Disk: Greater than 20 GB available disk space
- Network: At least one Ethernet interface

Software prerequisites

Here are the software prerequisites for IBM PowerAI Vision:

- Operating system:
  - Ubuntu 16.04 LTS for IBM PowerAI V1.4
  - Red Hat Enterprise Linux 7.4 for V1.5 (architecture ppc64le)
- Docker.io 1.12.0 or newer.
- NVIDIA driver version 384.59 or higher is required.
- NVIDIA CUDA version 8.0.44 or higher is required.
- NVIDIA cuDNN 6.0 or higher is required.

For more information about IBM PowerAI Vision, see Deep Learning and PowerAI Development.

2.4.3 IBM Spectrum Conductor Deep Learning Impact

IBM Spectrum Conductor™ DLI is an add-on to IBM Spectrum Conductor with Spark that provides DL capabilities. DLI supports both IBM Power Systems servers and x86 servers.

Figure 2-5 shows the relationship between DLI and IBM PowerAI on IBM Power Systems servers.

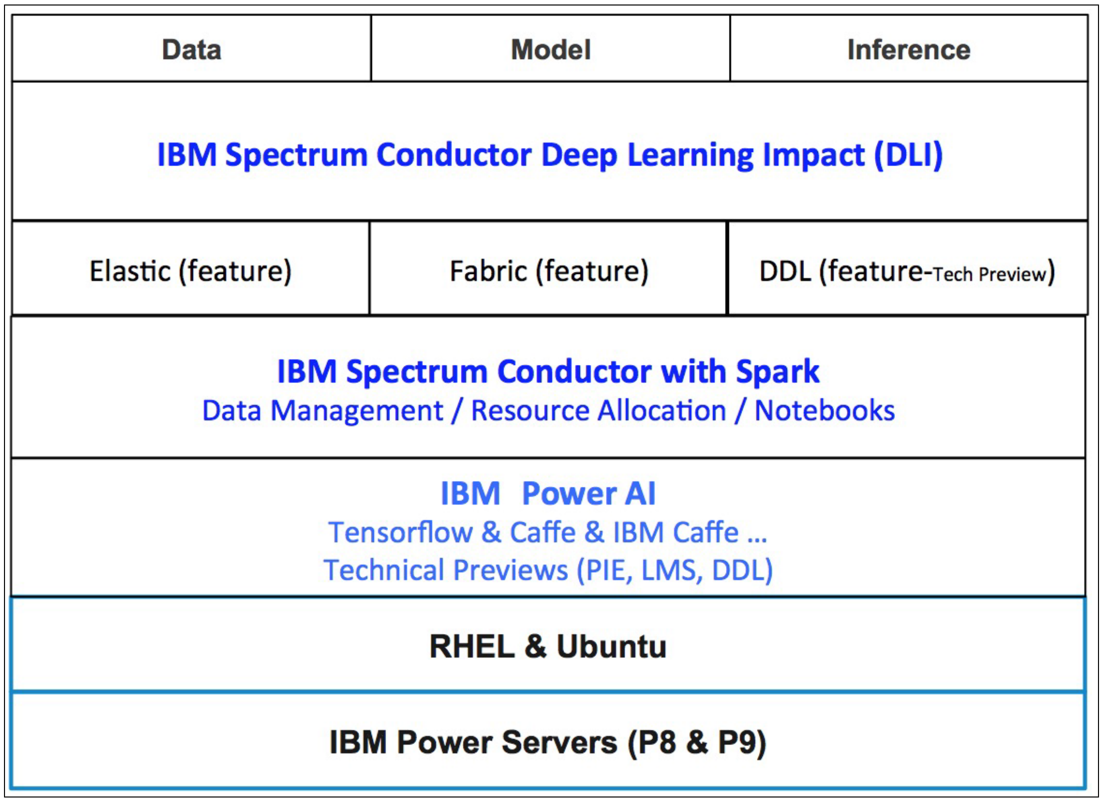

Data
Model
Inference
IBM Spectrum Conductor Deep Learning Impact (DLI)
Elastic (feature)
Fabric (feature)
DDL (feature-Tech Preview)
IBM Spectrum Conductor with Spark
Data Management / Resource Allocation / Notebooks
IBM Power AI
Tensorflow & Caffe & IBM Caffe …
Technical Previews (PIE, LMS, DDL)
RHEL & Ubuntu
IBM Power Servers (P8 & P9)

Figure 2-5 Relationship between DLI and IBM PowerAI on the IBM Power platform

DLI provides a GUI to take advantage of frameworks of IBM PowerAI to perform DL activities, such as data set importing, model training, model validation, and tuning, and supports transforming a trained model to inference mode and providing service for customer to-do prediction. In addition, DLI uses IBM Spectrum Conductor with Spark to provide a multitenant solution and manage GPU resources flexibly in the DL cluster environment. DLI is an end-to-end DL lifecycle management product.

There are some environment requirements for DLI Version 1.1.0 to run in an IBM Power platform:

- IBM PowerAI V1.5
- IBM Conductor with Spark V2.2.1
- NVIDIA GPU device in the compute node

For more information about the hardware and software requirements for IBM Spectrum Conductor DLI V1.1.0, see IBM Knowledge Center.

Chapter 6, “Introduction to IBM Spectrum Conductor Deep Learning Impact” on page 143 provides more content about DLI, including DLI benefits, deployment, how to use it, and case scenarios.

3

IBM PowerAI components

This chapter describes IBM PowerAI.

This chapter contains the following topics:

- IBM PowerAI components
- IBM PowerAI compatibility matrix

3.1 IBM PowerAI components

This section covers the components of IBM PowerAI since the first public release (V1.3) to the current release (V1.5.0) as of December 2017.[1]

IBM PowerAI is an IBM Cognitive Systems offering for the rapidly growing and quickly evolving artificial intelligence (AI) tier of deep learning (DL). IBM PowerAI provides a suite of capabilities from the open source community, and combines them into a single enterprise distribution of software that incorporates complete lifecycle management of installation and configuration, data ingestion and preparation, building, optimizing, and training the model, inference, testing, and moving the model into production. IBM PowerAI takes advantage of a distributed architecture to help enable your teams to quickly iterate through the training cycle with more data to help continuously improve the model over time.

IBM PowerAI is designed for enterprise scale, with software that is optimized for both single-server and cluster DL training.

IBM PowerAI provides an end-to-end DL platform for data scientists. It offers many optimizations that can ease installation and management, and can accelerate performance:

- Ready-to-use DL frameworks (TensorFlow and IBM Caffe).
- Distributed as easy-to-install binary files.
- Includes all dependencies and libraries.
- Easy updates: Code updates arrive from a repository.
- Validated DL platform with each release.
- Integrated ingest interfaces, with optional parallel transformation capability to unlock larger DL data sets.
- Dedicated support teams for DL.
- Designed for enterprise scale with multisystem cluster performance and large memory support.

IBM PowerAI software distribution consists of a meta-package that includes all libraries, DL frameworks, and software customizations to enable a fast, reliable, and optimized way to deploy a DL solution on IBM Power Systems.

IBM PowerAI is distributed through electronic download from My Entitled System Support, or it can be preinstalled on the hardware. The following items are included:

- IBM international license agreement for non-warranted programs
- Required installation files

Note: There is no physical media available. IBM PowerAI does not include the operating system or the NVIDIA graphical processing units (GPUs) drivers and libraries due to license limitations, or extra software such as Python, IBM Spectrum Conductor Deep Learning Impact (DLI), or IBM PowerAI Vision.

The following section also describes in greater detail all IBM PowerAI components and extra requirements (whether they are part of IBM PowerAI), including the versions that are supported in each release. For those freely available and not included in the IBM PowerAI meta-package, the following section provides a URL with more information and downloadable content.

[1] As of December 2017, IBM PowerAI V1.5 for POWER9 systems was released as part of an Early Ship Program. More frameworks are expected to be part of the final release of this version for POWER9 servers.

For those components that are part of IBM PowerAI, at the time of writing, we document a tabulation of dependencies between the component and the specific IBM PowerAI release.

3.1.1 IBM PowerAI support and extra services from IBM

IBM offers optional Level 3 enterprise support for IBM PowerAI, and also services for the deployment and optimization of IBM PowerAI to its customers. The following section describes the services in more detail.

IBM Global Technology Services

The list of selected services that are available in your region, either as standard or customized offerings for the efficient installation, implementation, and integration of this program, can be obtained from your respective IBM Global Technology Services® Regional Offering Executive.

Sales support

Technical sales support is offered globally and includes professionals from IBM Systems Group and IBM Sales and Delivery.

Resources include Field Technical Support Specialists (FTSSs), who provide onsite support for sales opportunities through pilots, prototypes, and demonstrations. Extra technical resources include:

- Advanced Technical Skills (ATS): Complex solution design, solutions assurance, proof of concept, early product introduction, and performance and benchmarking support.
- Techline: Presales services include System Design and Configurations, Design Assessments, Middleware and ISV Sizings, Services Support, Intellectual Capital, Competitive Assessments, and Business Partner Enablement.

For IBM presales support, see Global Technical Sales.

IBM Systems Lab Services

IBM Systems Lab Services offers a wide array of services that are available for your enterprise. The team brings expertise on the current technologies from the IBM development community and can help with your most difficult technical challenges.

IBM Systems Lab Services exists to help you successfully implement emerging technologies that can accelerate your return on investment and improve your satisfaction with your IBM systems and solutions. Services examples include initial implementation, integration, migration, and skills transfer on IBM systems solution capabilities and preferred practices.

For more information about available services, contact your IBM representative or see IBM Systems Lab Services.

IBM Power Systems PowerAI basic startup services

IBM Systems Lab Services offers IBM Power Systems PowerAI basic startup services that include pre-implementation design workshop and provide onsite assistance with installation, configuration, and implementation for software and hardware infrastructure for the customer's IBM PowerAI solution.

Note: Contact your IBM representative for availability in your country.

3.1.2 IBM Power Systems for deep learning

As the main part of the solution, IBM PowerAI relies on both Power Systems servers and NVIDIA GPUs to deliver maximum performance and a reliable solution for DL workloads. This is a software offering, so hardware is not included with IBM PowerAI, but it is required for the solution to run.

IBM Power System S822LC for High Performance Computing (8335-GTB) server

The IBM Power System S822LC for High Performance Computing server is the IBM chosen platform for running DL workloads. This server benefits from POWER8 CPUs plus dedicated P100 GPUs from NVIDIA to support demanding workloads, such as the typical DL workload during the training phase.

The IBM Power System S822LC for High Performance Computing server was designed and built in collaboration with the OpenPOWER Foundation partners NVIDIA, Mellanox, and Wistron to tackle high-performance and technical computing workloads. This is the first server to incorporate NVIDIA NVLink technology into the processor technology. For more information, see the NVIDA NVlink website.

This system is designed to provide the highest performance and greatest efficiency for workloads that use GPU acceleration, including CFD and molecular modeling applications. GPU acceleration is also used extensively in machine learning (ML), DL, and cognitive workloads.

This system supports up to four NVIDIA Tesla P100 GPUs, which are connected through the NVIDIA NVLink. Each node can include up to 1 TB of DDR4 memory, and supports Mellanox 100 Gbps (Enhanced Data Rate (EDR)) InfiniBand adapters for interconnect. Large clusters accommodate high-speed interconnect between nodes.

The system also includes the following items:

- Choice of air-cooled or water-cooled models for greater thermal efficiency
- Option of NVMe-based storage devices for greater performance
- 100 Gbps InfiniBand adapters that use Mellanox ConnectX-4 technology

For computationally rich high-performance or technical computing workloads that do not benefit from GPU acceleration, see *IBM Power System S822LC Technical Overview and Introduction*, REDP-5283.

IBM POWER9 servers: IBM Power System AC922 server

The IBM Power System AC922 server is the next generation of the IBM POWER processor-based systems, and is designed for DL and AI, high-performance analytics, and high-performance computing (HPC).

The IBM Power System AC922 (8335-GTW and 8335-GTG) is an OpenPOWER Linux scale-out server. In 2U form factors, these servers provide two POWER9 single chip module (SCM) processors, with up to 40 processor cores, coherently sharing 16 directly attached DDR4 dual inline memory module (DIMM) slots. The supported memory DIMMs are 8 - 128 GB.

These servers support the NVIDIA SXM2 form factor graphical processor units (GPUs) with an NVLink2 interface. The system also contains four directly attached, Coherent Accelerator Processor Interface (CAPI)-enabled Gen4 Peripheral Component Interconnect Express (PCIe) slots.

Four to six GV 100 GPUs can be installed on the system backplane.

Table 3-1 shows the NVIDIA GV 100 GPU features.

Table 3-1 NVIDIA GV 100 GPU features

GPU	Peak double precision floating point performance	Memory bandwidth	GPU memory size
Tesla V100	7.8 teraflops base[a]	1.2 TBps	16 GB or 32 GB

a. This is a projection from NVIDIA as of November 2017.

The 6-GPU configuration is water-cooled. The 4-GPU configuration is air-cooled as a standard solution, and water-cooled as an optional solution. For the air-cooled 2-GPU configuration, a feature upgrade is required to upgrade the system to a 4-GPU configuration. For a water-cooled system, different system backplanes are required for 4-GPU and 6-GPU configurations.

The system includes several features to improve performance:

- POWER9 processors:
  - Each POWER9 processor module has either 16 or 20 cores, and is based a 64-bit architecture:
    - Clock speeds for a 16-core chip of 2.6 GHz (3.09 GHz turbo)
    - Clock speeds for a 20-core chip of 2.0 GHz (2.87 GHz turbo)
  - 512 KB of L2 cache per core, and up to 120 MB of L3 cache per chip.
  - Up to four threads per core.
  - 120 GBps memory bandwidth per chip.
  - 64 GBps SMP interconnect between POWER9 chips.
- DDR4 memory:
  - Sixteen DIMM memory slots.
  - Maximum of 1024 GB DDR4 system memory.
  - Improved clock 1333 - 2666 MHz for reduced latency.
- NVIDIA Tesla V100 GPUs:
  - Up to six NVIDIA Tesla V100 GPUs, based on the NVIDIA SXM2 form factor connectors.
  - 7.8 TFLOPs per GPU for double precision.
  - 15.7 TFLOPs per GPU for single precision.
  - 125 TFLOPs per GPU for DL. New 640 Tensor Cores per GPU, designed for DL.
  - 16 GB HBM2 internal memory with 900 GBps bandwidth, 1.5x the bandwidth compared to Pascal P100.
  - Liquid cooling for six GPUs configurations to improve compute density.
- NVLink 2.0:
  - Twice the throughput, compared to the previous generation of NVLink.
  - Up to 200 GBps of bidirectional bandwidth between GPUs.
  - Up to 300 GBps of bidirectional bandwidth per POWER9 chip and GPUs, compared to 32 GBps of traditional PCIe Gen3.

- IBM OpenCAPI™ 3.0:
  - Open protocol bus to allow for connections between the processor system bus in a high speed and cache coherent manner with OpenCAPI compatible devices like accelerators, network controllers, storage controllers, and advanced memory technologies.
  - Up to 100 GBps of bidirectional bandwidth between CPUs and OpenCAPI devices.
- PCIe Gen4 slots. Four PCIe Gen4 slots up to 64 GBps bandwidth per slot, twice the throughput from PCIe Gen3. Three CAPI 2.0 capable slots.

Table 3-2 provides a summary of the IBM Power System AC922 server available models.

Table 3-2 Summary of the IBM Power System AC922 server available models

Server model	POWER9 chips	Max. memory	Max. GPU cards	Cooling
8335-GTG	2	1 TB	4	Air-cooled
8335-GTW			6	Water-cooled

Note: For more information, see *IBM Power System AC922 Introduction and Technical Overview*, REDP-5472.

IBM Power System AC922 server model 8335-GTG

This summary describes the standard features of the IBM Power System AC922 model 8355-GTG:

- 19-inch rack-mount (2U) chassis
- Two POWER9 processor modules:
  - 16-core 2.6 GHz processor module
  - 20-core 2.0 GHz processor module
  - Up to 1024 GB of 2666 MHz DDR4 error correction code (ECC) memory
- Two small form factor (SFF) bays for hard disk drives (HDDs) or solid-state drives (SSDs) that support:
  - Two 1 TB 7200 RPM NL SATA disk drives (#ELD0)
  - Two 2 TB 7200 RPM NL SATA disk drives (#ES6A)
  - Two 960 GB SATA SSDs (#ELS6)
  - Two 1.92 TB SATA SSDs (#ELSZ)
  - Two 3.84 TB SATA SSDs (#ELU0)
- Integrated SATA controller
- Four PCIe Gen4 slots:
  - Two PCIe x16 Gen4 Low Profile slots, CAPI-enabled
  - One PCIe x8 Gen4 Low Profile slot, CAPI-enabled
  - One PCIe x4 Gen4 Low Profile slot
- Two or four NVIDIA Tesla V100 GPUs (#EC4J), based on the NVIDIA SXM2 form factor connectors air-cooled

- Integrated features:
  - IBM EnergyScale™ technology
  - Hot-swap and redundant cooling
  - One front USB 3.0 port for general use
  - One rear USB 3.0 port for general use
  - One system port with RJ45 connector
- Two power supplies (both are required)

IBM Power System AC922 server model 8335-GTW

This summary describes the standard features of the Power AC922 model 8335-GTW:

- 19-inch rack-mount (2U) chassis
- Two POWER9 processor modules:
  - 16-core 2.6 GHz processor module
  - 20-core 2.0 GHz processor module
  - Up to 1024 GB of 2666 MHz DDR4 ECC memory
- Two SFF bays for HDDs or SSDs that support:
  - Two 1 TB 7200 RPM NL SATA disk drives (#ELD0)
  - Two 2 TB 7200 RPM NL SATA disk drives (#ES6A)
  - Two 960 GB SATA SSDs (#ELS6)
  - Two 1.92 TB SATA SSDs (#ELSZ)
  - Two 3.84 TB SATA SSDs (#ELU0)
- Integrated SATA controller
- Four PCIe Gen4 slots:
  - Two PCIe x16 Gen4 Low Profile slots, CAPI-enabled
  - One PCIe x8 Gen4 Low Profile slot, CAPI-enabled
  - One PCIe x4 Gen4 Low Profile slot
- Four or six NVIDIA Tesla V100 GPUs (#EC4J), based on the NVIDIA SXM2 form factor connectors water-cooled
- Integrated features:
  - IBM EnergyScale technology
  - Hot-swap and redundant cooling
  - One rear USB 3.0 port for general use
  - One system port with RJ45 connector
- Two power supplies (both are required)

Table 3-3 shows a list of IBM POWER servers that support NVIDIA GPUs and NVLink or NVLink2 buses. IBM PowerAI is supported by IBM Power Systems servers with NVLink CPUs and NVIDIA GPUs.

Table 3-3 IBM Power Systems servers that support NVIDIA GPUs

Name	Type-model	CPU	Bus	NVIDIA GPUs
IBM Power System S822LC for High Performance Computing	8335-GTB	POWER8	NVLink	Standard configuration: 2 or 4 Tesla P100 GPUs
IBM Power System AC922	8335-GTW	POWER9	NVLink2	Standard configurations: 6 Tesla V100 GPUs (water-cooled) Optional configurations: 4 Tesla V100 GPUs (water-cooled)
	8335-GTG			Standard configurations: 4 Tesla V100 GPUs (air-cooled) Optional configurations: 2 Tesla V100 GPUs (air-cooled)

3.1.3 Linux on Power for deep learning

IBM PowerAI is built on Linux on Power. IBM PowerAI benefits from both the Power platform performance and reliability, and the Linux open source model. At the time of writing, two Linux distributions are supported for IBM PowerAI in its different releases, as shown in Table 3-4 (although the operating system is not part of IBM PowerAI).

Table 3-4 IBM PowerAI supported operating systems

PowerAI	Linux	Extra requirements
V1.3	Ubuntu 16.04 on Power	► Linux kernel: 4.4 ► Extra packages: >= libc6 2.23-0ubuntu5[a] ► Extra repository: Updates
V1.3.1		
V1.3.2		
V1.3.3		
V13.4		
V1.4.0		
V1.5.0	RHEL 7.4 for POWER8 / RHEL 7.4 for POWER9	Extra repositories: ► Optional ► Extra ► EFEL

a. After installing Ubuntu 16.04, update the libc6 package to version 2.23-0ubuntu5 or higher. That version fixes problems with Torch and TensorFlow. You might need to enable the updates repository to install this update.

Ubuntu 16.04 LTS for IBM POWER8 (ppc64le)[2]

Ubuntu for POWER8 brings the Ubuntu server and Ubuntu infrastructure to IBM POWER8. Ubuntu 16.04 continues to enable rapid innovation for POWER8, including support for new POWER8 models, memory and PCI Hotplug, Docker, and many performance and availability enhancements.

For more information, see Ubuntu Server Guide.

Red Hat Enterprise Linux for IBM POWER

Starting with Red Hat Enterprise Linux 7.1, Red Hat provides separate builds and licenses for big endian and little endian versions for IBM Power Systems servers.

Red Hat Enterprise Linux 7.4 for IBM Power LE (POWER9) introduces the Red Hat Enterprise Linux 7.4 user space with an updated kernel.

The Red Hat Enterprise Linux 7.4 kernel depends on the CPU POWER architecture, so it is different for POWER8 and POWER9 processors.

For more information about Red Hat Enterprise Linux for IBM POWER, see Red Hat Enterprise Linux 7.4 for IBM Power LE (POWER9) - Release Notes and Power System S822LC for High Performance Computing.

3.1.4 NVIDIA GPUs

GPUs are fundamental when it comes to calculations that are related to DL. Unlike CPUs, GPUs are designed and optimized to perform mathematical operations over matrixes and vectors, which are the core of DL, in particular, during training tasks (the most intensive).

Table 3-5 shows the main differences between CPUs and GPUs.

Table 3-5 GPUs versus CPUs

GPUs	CPUs
Hundreds or thousands of simpler cores	A few complex cores
Thousands of concurrent hardware threads	Single or low number of threads performance optimization
Maximize floating-point throughput	Not applicable
Most die surface for integer and floating-point units	Transistor space that is dedicated to ILP

IBM PowerAI relies on the IBM POWER architecture, which is known for its resilience and SMT capabilities, plus the NVIDIA GPUs and NVLink technologies, as shown in Table 3-6.

Table 3-6 NVIDIA GPUs specifications

Component	Tesla V100	Tesla P100
GPU	GV100	GP100
Architecture	Volta	Pascal
CUDA cores	5376	3840

[2] ppc64le is a pure little endian mode that was introduced with POWER8 as the prime target for technologies that are provided by the OpenPOWER Foundation, aiming at enabling the porting of the x86 Linux-based software with minimal effort.

Component	Tesla V100	Tesla P100
Tensor cores	640	N/A
Core clock	Not disclosed	1328 MHz
Boost clocks	1455 MHz	1480 MHz
SMs[a]	84	60
Memory	HBM2	
Memory bus width	4096-bit	
Memory bandwidth	900 GBps	600 GBps
Shared memory	128 KB, configurable	24 KB L1, 64 KB shared
L2 cache	6 MB	4 MB
Half precision	30 teraflops	21.2 teraflops
Single precision	15.7 teraflops	10.6 teraflops
Double precision	7.8 teraflops (1/2 rate)	5.3 teraflops (1/2 rate)
For DL	125 teraflops	N/A
Die size	815 mm^2	610 mm^2
Transistor count	21.1×10^9	15.3×10^9
TDP	300 W	
Manufacturing process	TSMC 12 nm FFN	TSMC 16 nm FinFET

a. Streaming multiprocessor

3.1.5 NVIDIA components

For proper operation, IBM PowerAI requires low-level access to all functions that are provided by NVIDIA GPU and NVLink technologies. To achieve this access, NVIDIA drivers, CUDA, and CUDA Deep Neural Network (cuDNN) software are needed, but they are not provided within IBM PowerAI due to license constraints by NVIDIA. These components can be downloaded from the vendor webpage (as shown in 3.1.5, “NVIDIA components” on page 42 and 3.1.6, “NVIDIA drivers” on page 44).

CUDA

CUDA is a parallel computing platform and programming model that enables the use of the GPU to accelerate computing performance.

CUDA broadly follows the data-parallel model of computation. Typically, each thread runs the same operation on different elements of the data in parallel.

The data is divided into a 1D, 2D, or 3D grid of blocks. Each block can be 1D, 2D, or 3D in shape, and can consist of over 512 threads on current hardware. Threads within a thread block can cooperate by way of shared memory. Thread blocks are run as smaller groups of threads that are known as *warps*.

NVIDIA CUDA Toolkit

The NVIDIA CUDA Toolkit provides a development environment for creating high-performance, GPU-accelerated applications. GPU-accelerated CUDA libraries enable drop-in acceleration across multiple domains, such as linear algebra, image and video processing, DL, and graph analytics. For developing custom algorithms, you can use available integrations with commonly used languages and numerical packages and well-published development APIs. Table 3-7 shows the supported NVIDIA CUDA Toolkit versions.

Table 3-7 Supported NVIDIA CUDA Toolkit versions

IBM PowerAI	NVIDIA CUDA Toolkit	
	Minimal	**Recommended**
V1.3.0	8.0	>= 8.0.44
V1.3.1	>= 8.0.44	
V1.3.2		
V1.3.3		8.0.61
V1.3.4		
V1.4.0		
V1.5.0 for POWER8	9.0	9.0.176
V1.5.0 for POWER9	9.1.85	

For more information, see the CUDA Toolkit.

NVIDIA cuDNN

The NVIDIA cuDNN library is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines, such as forward and backward convolutional, pooling, normalization, and activation layers. Table 3-8 shows the supported NVIDIA cuDNNs.

Table 3-8 Supported NVIDIA cuDNNs

IBM PowerAI	NVIDIA cuDNN		
	Minimal	**Recommended**	**Runtime libraries**
V1.3.0	Version 5.1	N/A	► cuDNN v5.1 Runtime Library for Ubuntu16.04 POWER8 (Deb) ► cuDNN v5.1 Developer Library for Ubuntu16.04 POWER8 (Deb) ► cuDNN v5.1 Code Samples and User Guide POWER8 (Deb)
V1.3.1			
V1.3.2			
V1.3.3			
V1.3.4		5.1.10	
V1.4.0	Version 6.0	6.0.20	► cuDNN v6.0 Runtime Library for Ubuntu16.04 POWER8 (Deb) ► cuDNN v6.0 Developer Library for Ubuntu16.04 POWER8 (Deb) ► cuDNN v6.0 Code Samples and User Guide POWER8 (Deb)

IBM PowerAI	NVIDIA cuDNN		
	Minimal	Recommended	Runtime libraries
V1.5.0 for POWER8	Version 7.0	7.0.5	▶ cuDNN v7.0.5 Runtime Library for Ubuntu16.04 POWER8 (Deb) ▶ cuDNN v7.0.5 Developer Library for Ubuntu16.04 POWER8 (Deb) ▶ cuDNN v7.0.5 Code Samples and User ▶ Guide POWER8 (Deb)
V1.5.0 for POWER9	7.0.5		▶ cuDNN v7.0.5 Runtime Library for Ubuntu16.04 POWER8/POWER9 (Deb) ▶ cuDNN v7.0.5 Developer Library for Ubuntu16.04 POWER8/POWER9 (Deb) ▶ cuDNN v7.0.5 Code Samples and User ▶ Guide POWER8/POWER9 (Deb)

cuDNN accelerates widely used DL frameworks, including Caffe2, TensorFlow, and Theano.

Here are some cuDNN key features:

- Forward and backward paths for many common layer types, such as pooling, LRN, LCN, batch normalization, dropout, CTC, rectified linear unit (ReLU), Sigmoid, softmax and Tanh
- Forward and backward convolutional routines, including cross-correlation, which is designed for convolutional neural nets
- Long and short term memory (LSTM) and GRU recurrent neural networks (RNNs) and persistent RNNs
- Arbitrary dimension ordering, striding, and subregions for 4D tensors means easy integration into any neural net implementation
- Tensor transformation functions
- Context-based API allows for easy multithreading

For more information, see NVIDIA cuDNN.

3.1.6 NVIDIA drivers

NVIDIA drivers are software that helps the operating system access all the capabilities of the NVIDIA GPUs. These drives are provided by NVIDIA and are not part of IBM PowerAI distribution because of NVIDIA proprietary software policies.

NVIDIA drivers download

To download the appropriate drivers (Table 3-9 on page 45), complete the following steps:

1. Go to NVIDIA.
2. Select **DRIVERS**.
3. From the drop-down menu, select **ALL NVIDIA DRIVERS**.
4. Select **Manually find drivers for my NVIDIA products**:
   a. Product Type: Tesla
   b. Product Series: pSeries
   c. Product: Tesla P100

   d. Operating system: Select **Show all operating systems** and then, depending on your operating system, select **Linux POWER8 Ubuntu 16.04** or **Linux POWER8 RHEL7**.

5. Choose the CUDA version that is recommended for your IBM PowerAI release and **English** as the language, and click **Search**. Then, you have access to a web page where you can download the drivers and read some recommendations and release notes.

Table 3-9 Supported NVIDIA drivers

IBM PowerAI	**NVIDIA drivers**	
	Minimal	**Recommended**
V1.3.0	>=361.93.03	
V1.3.1		
V1.3.2		
V1.3.3	>=361.93.03	361.119
V1.3.4		361.121
V1.4.0	384.66	
V1.5.0 for POWER8	384.81	
V1.5.0 for POWER9	387.86	

3.1.7 IBM PowerAI deep learning package

IBM provides a set of libraries, frameworks, and several customizations to get the best performance from IBM Power Systems servers in a convenient meta-package. This is the core of IBM PowerAI and facilitates a fast and an easy deployment of the product.

Table 3-10 shows the IBM PowerAI DL packages.

Table 3-10 IBM PowerAI deep learning packages

PowerAI	**IBM PowerAI DL packages**
V1.3.0	`mldl-repo-local_1-3ibm2_ppc64el.deb`
V1.3.1	`mldl-repo-local_1-3ibm5_ppc64el.deb`
V1.3.2	`mldl-repo-local_1-3ibm7_ppc64el.deb`
V1.3.3	`mldl-repo-local_3.3.0_ppc64el.deb`
V1.3.4	`mldl-repo-local_3.4.1_ppc64el.deb` `mldl-repo-local_3.4.2_ppc64el.deb`
V1.4.0	`mldl-repo-local_4.0.0_ppc64el.deb`
V1.5.0	`mldl-repo-local-cuda9.0-5.0.0-*.ppc64le.rpm`

The main repository for IBM PowerAI DL packages is in an Ubuntu archive.

3.1.8 Libraries

A few libraries are included within the IBM PowerAI meta-package, which are part of the solution that is shown in Table 3-11. Many of these libraries are not used in all scenarios, although they are a prerequisite for the proper operation of IBM PowerAI. Every IBM PowerAI release contains the libraries and versions that are required for the frameworks of the release.

Table 3-11 Extra libraries included

IBM PowerAI	OpenBLAS	NVIDIA Collective Communications Library	Bazel	OpenMPI
V1.3.0	N/A	N/A	N/A	N/A
V1.3.1	Part of IBM PowerAI distributed packages	Version 1		
V1.3.2			Part of IBM PowerAI distributed packages	
V1.3.3				
V1.3.4				
V1.4.0				Part of IBM PowerAI[a]
V1.5.0				Part of IBM PowerAI[b]

a. The caffe-ibm and ddl-tensorflow packages require the IBM PowerAI OpenMPI package, which is built with NVIDIA CUDA support. That OpenMPI package conflicts with Ubuntu non-CUDA-enabled OpenMPI packages.
b. IBM PowerAI V1.5 includes IBM Spectrum MPI as a prerequisite.

OpenBLAS

OpenBLAS is an open source implementation of the Basic Linear Algebra Subprograms (BLAS) API with many hand-crafted optimizations for specific processor types. OpenBLAS is developed at the Lab of Parallel Software and Computational Science, ISCAS.

OpenBLAS adds optimized implementations of linear algebra kernels for several processor architectures. OpenBLAS is a fork of GotoBLAS2, which was created by Kazushige Goto at the Texas Advanced Computing Center.

For more information, see OpenBLAS.

NVIDIA Collective Communications Library

The NVIDIA Collective Communications Library (NCCL) implements multi-GPU and multi-node collective communication primitives that are performance-optimized for NVIDIA GPUs. NCCL provides routines such as all-gather, all-reduce, broadcast, reduce, and reduce-scatter, that are optimized to achieve high bandwidth over PCIe and NVLink high-speed interconnect.

Note: This library is usually pronounced as the word “nickel” [nik-uhl].

For more information, see NVIDIA Collective Communications Library.

Bazel

Bazel is a build tool that builds code quickly and reliably. Supported build tasks include running compilers and linkers to produce executable programs and libraries, and assembling deployable packages.

Bazel is a type of the tool that Google uses to build its server software internally, but expanded to build other software as well. For more information, see Bazel at GithHub and Bazel build.

OpenMPI

OpenMPI is a Message Passing Interface (MPI)[3] library project combining technologies and resources from several other projects: FT-MPI, LA-MPI, LAM/MPI, and PACX-MPI.

Open MPI represents the merger between three well-known MPI implementations:

- FT-MPI from the University of Tennessee
- LA-MPI from Los Alamos National Laboratory
- LAM/MPI from Indiana University
- With contributions from the PACX-MPI team at the University of Stuttgart

These four institutions are the founding members of the OpenMPI development team.

The OpenMPI code has three major code modules:

- OMPI: The MPI API and supporting logic
- ORTE: The Open Run-Time Environment (support for different back-end runtime systems)
- OPAL: The Open Portable Access Layer (utility and *glue* code that is used by OMPI and ORTE)

For more information, see OpenMPI.

Note: Uninstall any openmpi or libopenmpi packages before installing IBM Caffe or the Distributed Deep Learning (DDL) custom operator for TensorFlow. Purge any configuration files to avoid interference by running the following commands:

```
$ dpkg -l | grep openmpi
$ sudo apt-get purge ..
```

3.1.9 Frameworks

IBM PowerAI provides in a single package, easy to deploy and set up, some of the most used frameworks for DL and their dependencies, along with the required libraries (described in 3.1.8, “Libraries” on page 46). The following section is a brief description of every one of the frameworks that are included in any of the IBM PowerAI releases post-R3.0.

Berkeley Vision and Learning Center upstream Caffe[4]

Caffe is a DL framework that has expression, speed, and modularity. It is developed by Berkeley AI Research (BAIR)[5]/Berkeley Vision and Learning Center (BVLC)[6] and community contributors.

[3] MPI is a standardized and portable message-passing standard that is designed by a group of researchers from academia and industry to function on a wide variety of parallel computing architectures. The standard defines the syntax and semantics of a core of library routines that are useful to a wide range of users writing portable message-passing programs in C, C++, and Fortran. There are several well-tested and efficient implementations of MPI, many of which are open source or in the public domain.

[4] Y. Jia, et al., “Caffe: Convolutional Architecture for Fast Feature Embedding”, arXiv preprint arXiv:1408.5093, 2014.

[5] The BAIR Lab brings together University of California Berkeley researchers across the areas of computer vision, ML, natural language processing (NLP), planning, and robotics. For more information, see BAIR.

[6] https://github.com/BVLC

Yangqing Jia[7] created the project during his PhD at the University of California Berkeley. Caffe is released under the BSD 2-Clause license. Table 3-12 shows the Caffe BVLC version for IBM PowerAI.

Table 3-12 Caffe Berkeley Vision and Learning Center versions

IBM PowerAI	Berkeley Vision and Learning Center upstream Caffe
V1.3.0	V1.0.0 rc3
V1.3.1	
V1.3.2	
V1.3.3	
V1.3.4	V1.0.0 rc5
V1.4.0	V1.0.0
V1.5.0 for POWER8	
V1.5.0 for POWER9	Not included in IBM PowerAI V1.5 for P9 ESP

Here are the main benefits:

- Large user community with academic research projects and industrial applications in vision, speech, and multimedia
- Extensible code fosters active development
- Expressive architecture encourages application and innovation
- Models and optimization are defined by configuration without hardcoding
- Model Zoo[8] of pre-trained networks

For more information, see Caffe Berkeley Vision.

IBM optimized version of Berkeley Vision and Learning Center Caffe

IBM Caffe is a variant of BVLC/Caffe and it is optimized for NVLink-enabled IBM Power Systems servers. Table 3-13 shows the IBM Caffe versions that are used with IBM PowerAI.

Table 3-13 IBM Caffe versions

IBM PowerAI version	IBM Optimized version of Berkeley Vision and Learning Center Caffe
V1.3.0	V1.0.0 rc3
V1.3.1	
V1.3.2	
V1.3.3	
V1.3.4	
V1.4.0	V1.0.0
V1.5.0 for POWER8	
V1.5.0 for POWER9	Not included in IBM PowerAI V1.5 for P9 ESP

[7] `http://daggerfs.com/`

[8] A standard format for packaging Caffe model information, plus tools to upload/download model information to/from GitHub Gists, and to download trained `.caffemodel` binary files.

Aside from all the features and capabilities that are provided by BVLC Caffe, the IBM optimized version of BVLC Caffe offers extra advantages:

- This a compilation that is optimized by IBM Power systems engineers and designed to run on Power Systems servers, so it is highly optimized for POWER processors.
- As part of the collaboration with NVIDIA, this compilation is also optimized for NVIDIA GPUs (including most of the optimizations NVCaffe includes).
- Includes support for Large Model Support (LMS).

For more information, see GitHub - open-power-ref-design/deep-learning.

NVIDIA fork of Caffe

NVIDIA Caffe is an NVIDIA-maintained fork of BVLC Caffe that is tuned for NVIDIA GPUs, particularly in multi-GPU configurations. Table 3-14 shows the NVIDIA Caffe version for IBM PowerAI.

Table 3-14 NVIDIA Caffe versions

IBM PowerAI version	**NVIDIA fork of Caffe**	
	Alternative	**Default**
V1.3.0	V0.14.15	
V1.3.1	V0.14.15	V0.15.13
V1.3.2		
V1.3.3		
V1.3.4		V0.15.14
V1.4.0	V0.15.14	
V1.5.0	Deprecated, no longer part of IBM PowerAI	

Here are the major features:

- 16-bit (half) floating point train and inference support.
- Mixed-precision support to store and compute data in either 64-, 32-, or 16-bit formats. Precision can be defined for every layer (forward and backward passes might be different too), or it can be set for the whole net.
- Integration with cuDNN v6.
- Automatic selection of the best cuDNN convolutional algorithm.
- Integration with Version 1.3.4 of NCCL library for improved multi-GPU scaling.
- Optimized GPU memory management for data and parameters storage, I/O buffers, and workspace for convolutional layers.
- Parallel data parser and transformer for improved I/O performance.
- Parallel back propagation and gradient reduction on multi-GPU systems.
- Fast solvers implementation with fused CUDA kernels for weights and history update.
- Multi-GPU test phase for even memory load across multiple GPUs.
- Compatibility with earlier versions of BVLC Caffe and NVCaffe 0.15.
- Extended set of optimized models (including 16-bit floating point examples).

For more information, see GitHub - NVIDIA/Caffe: Caffe: A fast open framework for deep learning.

Theano

Theano is a Python library that you can use to define, optimize, and evaluate efficiently mathematical expressions involving multi-dimensional arrays. Computations are expressed by using a NumPy-esque syntax, and compiled to run efficiently on either CPU or GPU architectures.

Theano is an open source project that is primarily developed by an ML group at the Université de Montréal. On September 28, 2017, Pascal Lamblin announced that major development is ceasing after the 1.0 release, due before the end of 2017 because of competing offerings by strong industrial players.

Table 3-15 shows the Theano versions that are supported by IBM PowerAI.

Table 3-15 Theano versions

IBM PowerAI version	Theano
V1.3.0	V 0.8.2
V1.3.1	
V1.3.2	
V1.3.3	
V1.3.4	V 0.9.0
V1.4.0	
V1.5.0	Deprecated, no longer part of IBM PowerAI

For more information, see GitHub - Theano and Theano at Deep Learning Theano at Deep Learning.

Torch

Torch is an open source ML library, a scientific computing framework, and a script language based on the Lua[9] programming language. Torch provides a wide range of algorithms for deep ML, uses the scripting language LuaJIT, and an underlying C implementation.
Table 3-16 shows the Torch versions that are supported by IBM PowerAI.

Table 3-16 Torch versions

IBM PowerAI version	Torch
V1.3.0	Version 7
V1.3.1	
V1.3.2	
V1.3.3	
V1.3.4	
V1.4.0	
V1.5.0	Deprecated, no longer part of IBM PowerAI

9 https://www.lua.org/

Torch is the main package in Torch V7 where data structures for multi-dimensional tensors and mathematical operations over them are defined.

Here are the core features of Torch:

- A powerful N-dimensional array
- Many routines for indexing, slicing, and transposing
- Interface to C through LuaJIT
- Linear algebra routines
- Neural network, and energy-based models
- Numeric optimization routines
- Fast and efficient GPU support

For more information, see Torch.

Deep Learning GPU Training System

Deep Learning GPU Training System (DIGITS) provides an interface for training and classification that can be used to train DNNs with a few clicks. DIGITS runs as a web application that is accessed through a web browser. Table 3-17 shows the DIGITS versions that are used with IBM PowerAI.

Table 3-17 Deep Learning GPU Training System versions

IBM PowerAI version	DIGITS
V1.3.0	N/A
V1.3.1	Version 5.0.0-rc.1[a]
V1.3.2	
V1.3.3	
V1.3.4	Version 5.0.0[b]
V1.4.0	
V1.5.0	Deprecated, no longer part of IBM PowerAI

a. The digits and python-socketio-server packages conflict with the Ubuntu python-socketio package. Uninstall the python-socketio package before installing DIGITS.
b. The digits and python-socketio-server packages conflict with the Ubuntu python-socketio package. Uninstall the python-socketio package before installing DIGITS.

DIGITS makes it easy to visualize networks and quickly compare their accuracies. When you select a model, DIGITS shows the status of the training exercise and its accuracy, and provides the option to load and classify images during the time the network is training or after training completes.

Both Caffe and Torch are used by DIGITS for image classification. Table 3-18 shows the prerequisites for Torch + DIGITS.

Table 3-18 Prerequisites for Torch + Deep Learning GPU Training System

IBM PowerAI version	Prerequisites for Torch + DIGITS
V1.3.0	N/A
V1.3.1	libhdf5-serial-dev liblmdb-dev (+ extra luarocks)[a]
V1.3.2	
V1.3.3	

IBM PowerAI version	Prerequisites for Torch + DIGITS
V1.3.4	N/A
V1.4.0	
V1.5.0	Deprecated, no longer part of IBM PowerAI

a. Using Torch with DIGITS requires extra packages that are not part of the IBM PowerAI release 3.1 distribution. See steps 1 - 3 on page 52 to install the packages.

To make Torch work with DIGITS, complete the following steps:

1. Install IBM PowerAI Torch and DIGITS packages:

```
$ sudo apt-get install digits torch
```

2. Install prerequisite packages from Ubuntu:

```
$ sudo apt-get install libhdf5-serial-dev liblmdb-dev
```

3. Install extra luarocks that are needed for DIGITS Torch support:

```
$ source /opt/DL/torch/bin/torch-activate
$ luarocks install --local --dep-mode=order tds
$ luarocks install --local --dep-mode=order totem
$ luarocks install --local
--dep-mode=order"https://raw.github.com/deepmind/torch-hdf5/master/hdf5-0-0.roc
kspec"
$ luarocks install --local
--dep-mode=order"https://raw.github.com/Neopallium/lua-pb/master/lua-pb-scm-0.r
ockspec"
$ luarocks install --local --dep-mode=order lightningmdb 0.9.18.1-1
LMDB_INCDIR=/usr/include LMDB_LIBDIR=/usr/lib/powerpc64le-linux-gnu
$ luarocks install --local --dep-mode=order
"https://raw.githubusercontent.com/ngimel/nccl.torch/master/nccl-scm-1.rockspec
"
```

For more information, see DIGITS at NVIDIA.

Google TensorFlow

TensorFlow is an open source software library for numerical computation that uses data flow graphs. It is provided by Google.

Although new to the open source landscape, the Google TensorFlow DL framework has been in development for years as proprietary software. It was developed originally by the Google Brain Team for conducting research in ML and deep neural networks. The framework's name is derived from the fact that it uses data flow graphs, where nodes represent a computation and edges represent the flow of information, in Tensor form, from one node to another.

TensorFlow offers a good amount of documentation for installation, learning materials and tutorials that are aimed at helping beginners understand some of the theoretical aspects of neural networks, and getting TensorFlow set up and running relatively painlessly. Table 3-19 on page 53 shows the TensorFlow versions for IBM PowerAI.

Table 3-19 TensorFlow versions

IBM PowerAI version	Google TensorFlow		
	TensorFlow		ddl-tensorflow
	Alternative	Recommended	
V1.3.0	N/A		N/A
V1.3.1	Version 0.9.0		
V1.3.2	Version 0.12.0		
V1.3.3	Version 0.12.0	Version 1.0.0	
V1.3.4		Version 1.0.1	
V1.4.0	Version 1.1.0		Technology preview[a]
V1.5.0	Version 1.4.0		0.4.0

a. DDL custom operator for TensorFlow.

This release of IBM PowerAI includes a technology preview of the IBM PowerAI DDL custom operator for TensorFlow. The DDL custom operator uses CUDA-aware OpenMPI and NCCL to provide high-speed communications for distributed TensorFlow.

The DDL custom operator can be found in the ddl-tensorflow package. For more information about DDL and about the TensorFlow operator, see the following files:

- `/opt/DL/ddl/doc/README.md`
- `/opt/DL/ddl-tensorflow/doc/README.md`
- `/opt/DL/ddl-tensorflow/doc/README-API.md`

The DDL TensorFlow operator makes it easy to enable Slim-style models for distribution. The package includes examples of Slim models that are enabled with DDL, which you can access by running the following commands:

```
$ source /opt/DL/ddl-tensorflow/bin/ddl-tensorflow-activate
$ ddl-tensorflow-install-samples <somedir>
```

Those examples are based on a specific commit of the TensorFlow models repository with a small adjustment. If you prefer to work from an upstream clone, rather than the packaged examples, run the following commands:

```
$ git clone https://github.com/tensorflow/models.git
$ cd models
$ git checkout 11883ec6461afe961def44221486053a59f90a1b
$ git revert fc7342bf047ec5fc7a707202adaf108661bd373d
$ cp /opt/DL/ddl-tensorflow/examples/slim/train_image_classifier.py slim/
```

Unlike any other framework, TensorFlow can do partial subgraph computation, which involves taking a subsample of the total neural network and then training it, apart from the rest of the network. This is also called *model parallelization*, and allows for distributed training.

For more information, see TensorFlow and GitHub - TensorFlow.

Distributed Deep Learning TensorFlow

DDL TensorFlow is a library that enables TensorFlow to spread the workload over several nodes. As training models grow, they can reach a point where the data sets cannot fit into a one or multiple GPUs in a server. In such cases, the distributed TensorFlow architecture offers a great advantage, enabling several servers to support this workload, resulting in radically reduced processing time.

For more information, see Distributed Deep Learning TensorFlow.

Chainer

Chainer is a DL framework, primarily sponsored by Preferred Networks that focuses on the flexibility to write complex architectures simply and intuitively.

Chainer adopts a Define-by-Run scheme, for example, the network is defined dynamically through actual forward computation. More precisely, Chainer stores the history of computation instead of programming logic. This strategy enables it to fully take advantage of the power of programming logic in Python. The Define-by-Run scheme is the core concept of Chainer.

Table 3-20 shows the Chainer versions for IBM PowerAI.

Table 3-20 Chainer versions

IBM PowerAI version	Chainer
V1.3.0	N/A
V1.3.1	
V1.3.2	Version 1.18.0
V1.3.3	
V1.3.4	Version 1.20.0.1
V1.4.0	Version 1.23.0
V1.5.0	Deprecated, no longer part of IBM PowerAI

For more information, see Chainer and Chainer documentation.

3.1.10 Other software and functions

This section presents other software and their functions.

Python

Python is one of the most extended programming languages and the most used in the DL field. Most of the frameworks that are included in IBM PowerAI support Python (and some additional programming languages, such as C and Java).

At the time of writing, the only supported Python version in IBM PowerAI is Version 2. IBM intends to include support for Python V3 in the next releases of the product.

For more information, see Python.

Anaconda

Starting in IBM PowerAI V1.5 with Red Hat Enterprise Linux as the required operating system, Anaconda is required to distribute, update, and install software.

Anaconda is the installation program that is used by Fedora, Red Hat Enterprise Linux, and some other distributions. During installation, a target computer's hardware is identified and configured, and the appropriate file systems for the system's architecture are created. Anaconda enables the user to install the operating system software on the target computer. Anaconda can also upgrade existing installations of earlier versions of the same distribution. After the installation is complete, you can restart your installed system and continue doing customization by using the initial setup.

Anaconda is a fairly sophisticated installer. It supports installation from local and remote sources such as CDs and DVDs, images that are stored on an HDD, NFS, HTTP, and FTP. Installation can be scripted with kickstart to provide a fully unattended installation that can be duplicated on scores of machines. Anaconda can also be run over VNC on machines without an interface. Various advanced storage devices, including LVM, RAID, iSCSI, and multipath, are supported from the partitioning program. Anaconda provides advanced debugging features such as remote logging, access to the Python interactive debugger, and remote saving of exception dumps.

Starting from IBM PowerAI V1.5, Anaconda is a requirement when using TensorFlow and Caffe frameworks.

Table 3-21 shows the Anaconda versions for IBM PowerAI.

Table 3-21 Anaconda versions

IBM PowerAI version	Anaconda 2
V1.3.0	N/A
V1.3.1	
V1.3.2	
V1.3.3	
V1.3.4	
V1.4.0	
V1.5.0	5.0.0[a]

a. Anaconda is required if TensorFlow or Caffe frameworks are used.

For more information, see Anaconda at the Fedora Project.

IBM PowerAI Distributed Deep Learning

Many times, the amount of data that is required to be processed exceeds the capacity processing of a single server. In other cases, the training time can potentially be reduced by dividing the tasks among a cluster of servers.

To accelerate the time that is dedicated to training a model, the IBM PowerAI stack uses new technologies to deliver exceptional training performance by distributing a single training job across a cluster of servers.

IBM PowerAI DDL provides intelligence about the structure and layout of the underlying cluster (topology), which includes information about the location of the cluster's different compute resources, such as GPUs and CPUs and the data on each node.

IBM PowerAI is unique in that this capability is incorporated into the DL frameworks as an integrated binary file, reducing complexity for clients as they bring in high-performance cluster capability.

As a result of this capability, IBM PowerAI with DDL can scale jobs across many cluster resources with little loss to communication impact.

For an in-depth discussion on IBM PowerAI and IBM PowerAI DDL scalability, read *IBM PowerAI DDL (2017)*. It was written by a team of IBM scientists who describe and prove that DL workload performance when using IBM PowerAI DDL scales close to linearly with the number of nodes.

Note: At the time of writing, DDL is available as a technology preview with IBM PowerAI V1.4 and V1.5.0, and is compatible with bundled TensorFlow and IBM Caffe frameworks.

For more information, see Deep Learning and PowerAI Development.

Large Model Support

Because the models are becoming more complex and the data sets are getting larger, DL workloads are becoming bigger. In many cases, the size of the data set overgrows the GPUs memory space, keeping these workloads from been analyzed in these systems.

Figure 3-1 shows data fragmentation in a traditional approach that divides the problem into 16 GB chunks without LMS.

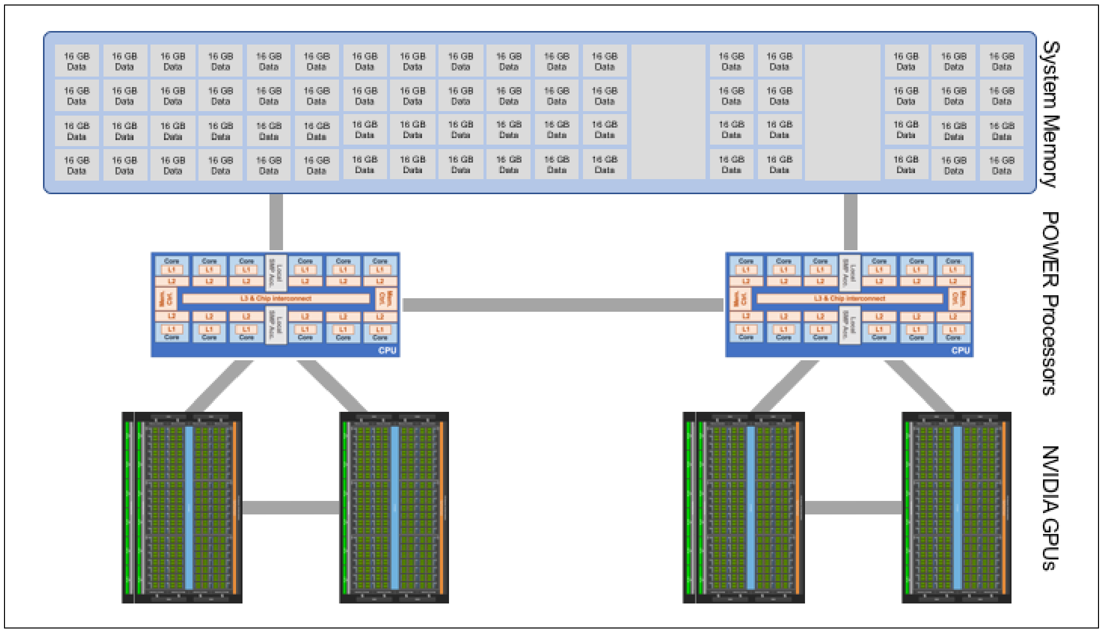


Figure 3-1 Large Model Support disabled

LMS uses the system memory with GPU memory to overcome GPU memory limitations in DL training.

Figure 3-2 shows the use of system memory with GPU memory in a system with LMS enabled.

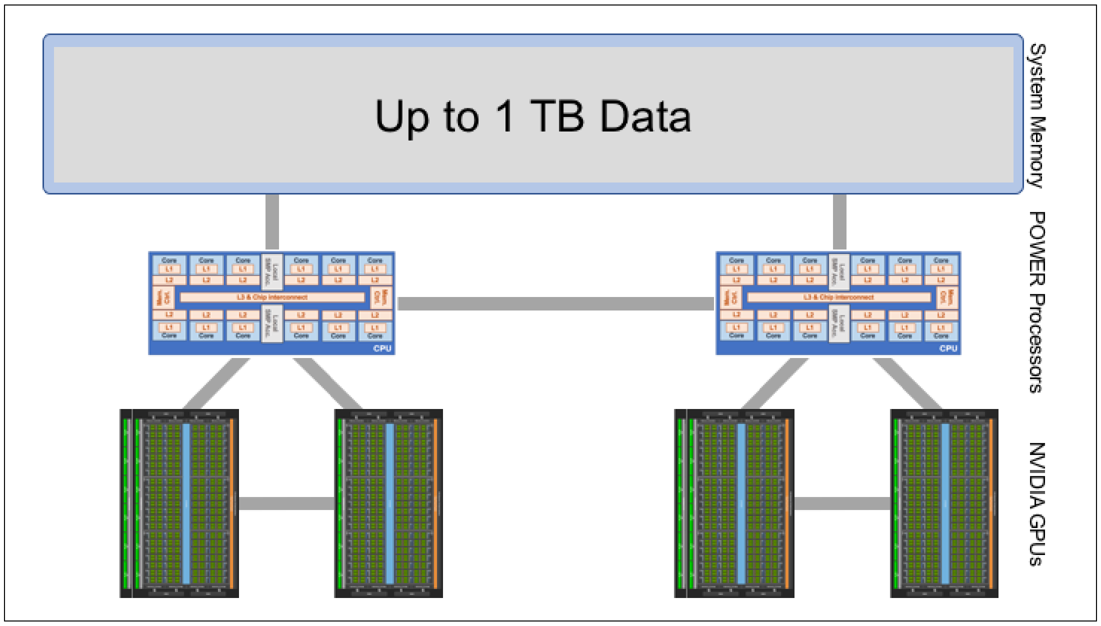


Figure 3-2 Large Model Support enabled

IBM PowerAI from V1R4.0 includes support for LMS in IBM Caffe as a technology preview.

IBM Spectrum MPI

IBM Spectrum MPI is a high-performance, production-quality implementation of MPI that accelerates application performance in distributed computing environments. Based on OpenMPI, IBM Spectrum MPI provides a familiar interface that is easily portable. IBM Spectrum MPI incorporates advanced CPU affinity features, dynamic selection of interface libraries, superior workload manager integrations and improved performance, and improved RDMA networking, which supports NVIDIA GPUs. IBM Spectrum MPI supports a broad range of industry-standard platforms, interconnects, and operating systems, ensuring that parallel applications can run on a multitude of platforms.

IBM Spectrum MPI delivers a number of features:

- Improved RDMA networking, supporting NVIDIA GPUs based on the IBM PAMI back end
- Reduce time to results:
  - Improved point-to-point performance by way of proprietary PAMI back end (for specific MOFED)
  - Enhanced collective library (blocking and non-blocking algorithms)
- Ease of use for installations, starts, and debugging
- Cluster test options, improving startup services
- Debug and instrumentation libraries
- IBM High Performance Computing Toolkit for analyzing performance of applications
- Single MPI with support by IBM for IBM POWER8 and x86

For more information, see IBM Spectrum MPI.

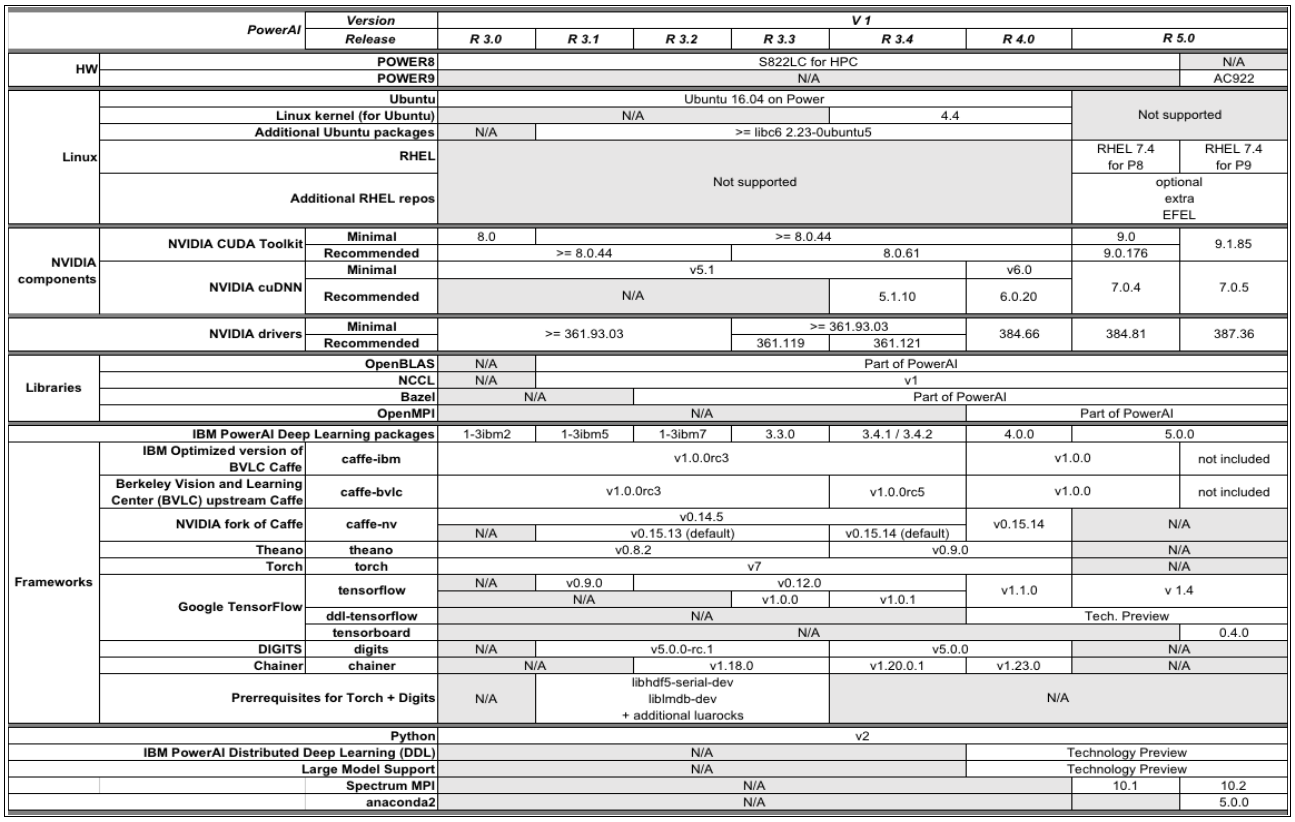
Table 3-22 recaps all additional software and functions that are described in this section.

Table 3-22 Extra software

IBM PowerAI version	Python	IBM PowerAI DDL	LMS	IBM Spectrum MPI
V1.3.0	Version 2	N/A	N/A	N/A
V1.3.1				
V1.3.2				
V1.3.3				
V1.3.4				
V1.4.0		Technology preview	Technology preview	
V1.5.0 for POWER8				10.1
V1.5.0 for POWER9				10.2

3.2 IBM PowerAI compatibility matrix

Figure 3-3 shows the requirements and elements that are included in every version of IBM PowerAI as of December 2017.



	PowerAI	Version	V 1							
		Release	R 3.0	R 3.1	R 3.2	R 3.3	R 3.4	R 4.0	R 5.0	
HW	POWER8		S822LC for HPC							N/A
	POWER9		N/A							AC922
Linux	Ubuntu		Ubuntu 16.04 on Power						Not supported	
	Linux kernel (for Ubuntu)		N/A				4.4			
	Additional Ubuntu packages		N/A	>= libc6 2.23-0ubuntu5						
	RHEL		Not supported						RHEL 7.4 for P8	RHEL 7.4 for P9
	Additional RHEL repos								optional extra EFEL	
NVIDIA components	NVIDIA CUDA Toolkit	Minimal	8.0	>= 8.0.44					9.0	9.1.85
		Recommended	>= 8.0.44			8.0.61			9.0.176	
	NVIDIA cuDNN	Minimal	v5.1					v6.0	7.0.4	7.0.5
		Recommended	N/A				5.1.10	6.0.20		
	NVIDIA drivers	Minimal	>= 361.93.03			>= 361.93.03		384.66	384.81	387.36
		Recommended				361.119	361.121			
Libraries	OpenBLAS		N/A	Part of PowerAI						
	NCCL		N/A	v1						
	Bazel		N/A		Part of PowerAI					
	OpenMPI		N/A					Part of PowerAI		
	IBM PowerAI Deep Learning packages		1-3ibm2	1-3ibm5	1-3ibm7	3.3.0	3.4.1 / 3.4.2	4.0.0	5.0.0	
Frameworks	IBM Optimized version of BVLC Caffe	caffe-ibm	v1.0.0rc3					v1.0.0		not included
	Berkeley Vision and Learning Center (BVLC) upstream Caffe	caffe-bvlc	v1.0.0rc3				v1.0.0rc5	v1.0.0		not included
	NVIDIA fork of Caffe	caffe-nv	v0.14.5					v0.15.14	N/A	
			N/A	v0.15.13 (default)			v0.15.14 (default)			
	Theano	theano	v0.8.2				v0.9.0		N/A	
	Torch	torch	v7						N/A	
	Google TensorFlow	tensorflow	N/A	v0.9.0	v0.12.0			v1.1.0	v 1.4	
			N/A			v1.0.0	v1.0.1			
		ddl-tensorflow	N/A					Tech. Preview		
		tensorboard	N/A							0.4.0
	DIGITS	digits	N/A	v5.0.0-rc.1			v5.0.0		N/A	
	Chainer	chainer	N/A		v1.18.0		v1.20.0.1	v1.23.0	N/A	
	Prerrequisites for Torch + Digits		N/A	libhdf5-serial-dev liblmdb-dev + additional luarocks			N/A			
	Python		v2							
	IBM PowerAI Distributed Deep Learning (DDL)		N/A					Technology Preview		
	Large Model Support		N/A					Technology Preview		
	Spectrum MPI		N/A						10.1	10.2
	anaconda2		N/A							5.0.0

Figure 3-3 Compatibility matrix

4

Deploying IBM PowerAI

This chapter includes installation guides and instructions to set up and test different frameworks inside IBM PowerAI. This chapter focuses mostly on details for system administrators and teams that are responsible for deploying and supporting IBM PowerAI.

This chapter contains the following topics:

- IBM PowerAI V1.4 setup guide
- About this chapter
- Testing IBM PowerAI V1.4
- Setting up IBM PowerAI V1.5.0 on a POWER S822LC for High Performance Computing server

4.1 IBM PowerAI V1.4 setup guide

This chapter contains information about how to set up IBM PowerAI V1.4.

4.1.1 About this chapter

This chapter shows the procedure for installing Ubuntu 16.04 on a bare metal IBM Power System S822LC for High Performance Computing (M/T 8335-GTB) server. This POWER8 server is equipped with a NVIDIA Tesla P100 graphical processing unit (GPU).

After the installation of Ubuntu, this chapter describes the procedure for installing IBM PowerAI V1.4.

This chapter is a comprehensive and a visual guide to IBM PowerAI installation and setup by using the most common installation scenario. For more information for all IBM PowerAI releases, see the IBM PowerAI Release Notes.

Note: To avoid conflicts with untested or unsupported versions of certain system libraries, disable automatic updates.

4.1.2 Preparing to install IBM PowerAI V1.4

Because the IBM Power System S822LC for High Performance Computing model does not have a DVD drive, you must install the operating system in one of the following ways:

1. Mounting virtual media with the Remote Console function of the baseboard management controller (BMC)
2. USB device
3. Network boot installation

During the use of any of these methods, you must have a terminal to connect to the system through Intelligent Platform Management Interface (IPMI).

For more information about installing Linux on IBM Power Systems LC servers, see IBM Knowledge Center.

Preparation steps

Before starting the configuration, prepare the connection environment from the target server to the internet. When installing the package, the download is performed from the internet to update any dependency relationship.

Downloads

This section describes the installation packages to be downloaded.

Ubuntu installation media

The Ubuntu installation media files are the following ones:

- Ubuntu 16.04.x ISO image file (`ubuntu - 16.04.3 - server - ppc64el.iso`)
- Ubuntu 16.04.3 LTS (Xenial Xerus)

To download the files, see Ubuntu 16.04.3 LTS (Xenials Xerus).

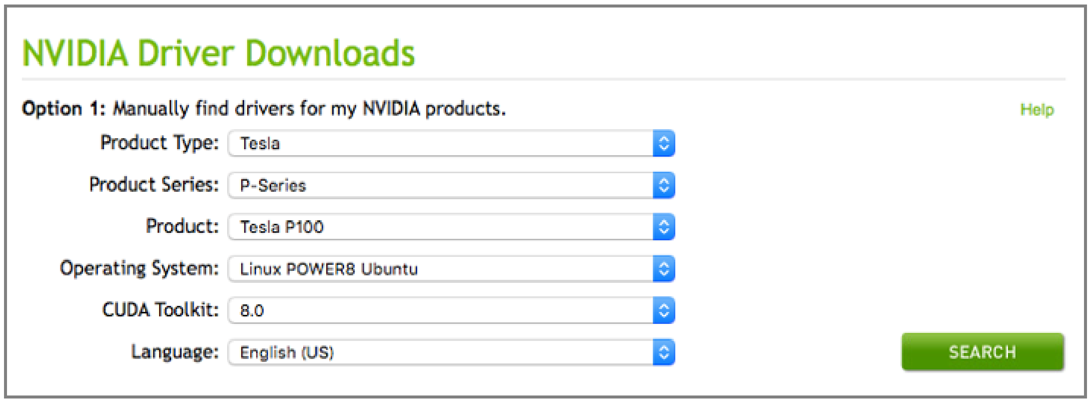
Fixes for IBM Power System S822LC for High Performance Computing server

As a preferred practice, check for firmware updates for the IBM Power System S822LC for High Performance Computing server at IBM Fix Central.

NVIDIA device driver

The NVIDIA GPU TESLA P100 device driver can be downloaded from NVIDIA.

When downloading the device driver, select the appropriate version of the drivers, as described in "Installing NVIDIA drivers and components" on page 94 and shown in Figure 4-1.



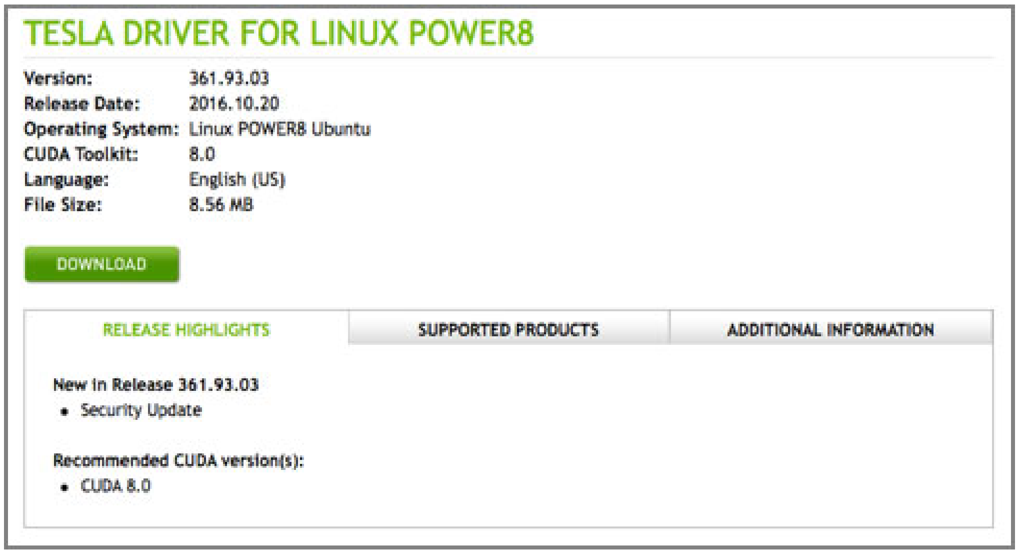
NVIDIA Driver Downloads

Option 1: Manually find drivers for my NVIDIA products.

Product Type: Tesla
Product Series: P-Series
Product: Tesla P100
Operating System: Linux POWER8 Ubuntu
CUDA Toolkit: 8.0
Language: English (US)

Figure 4-1   NVIDIA drivers download options

Figure 4-2 shows the TESLA drive for Linux on POWER8 download window.



TESLA DRIVER FOR LINUX POWER8

Version:	361.93.03
Release Date:	2016.10.20
Operating System:	Linux POWER8 Ubuntu
CUDA Toolkit:	8.0
Language:	English (US)
File Size:	8.56 MB

New in Release 361.93.03
- Security Update

Recommended CUDA version(s):
- CUDA 8.0

Figure 4-2   NVIDIA drivers download webpage

IBM PowerAI

IBM PowerAI deep learning (DL) packages can be downloaded from Index of /software/server/POWER/LInux/mldl/ubuntu.

NVIDIA components

This section shows the additional NVIDIA components to be downloaded:

- NVIDIA CUDA 8.0 toolkit[1]
- NVIDIA CUDA Deep Neural Network (cuDNN)

Download the following files for POWER8, checking the appropriate version, as shown in Figure 4-2 on page 61:

- cuDNN vX.0 Runtime Library for Ubuntu16.04 POWER8 (Deb)
- cuDNN vX.0 Developer Library for Ubuntu16.04 POWER8 (Deb)
- cuDNN vX.0 Code Samples and User Guide POWER8 (Deb)

4.1.3 IBM Power System S822LC for High Performance Computing initial setup

This section describes the initial setup for the IBM Power System S822LC for High Performance Computing server.

Configuring the management processor IP

The section describes the steps to configure the processor IP:

1. Connect the VGA port on the back of the IBM Power System S822LC for High Performance Computing server to the display and the USB keyboard, as shown in Figure 4-3.

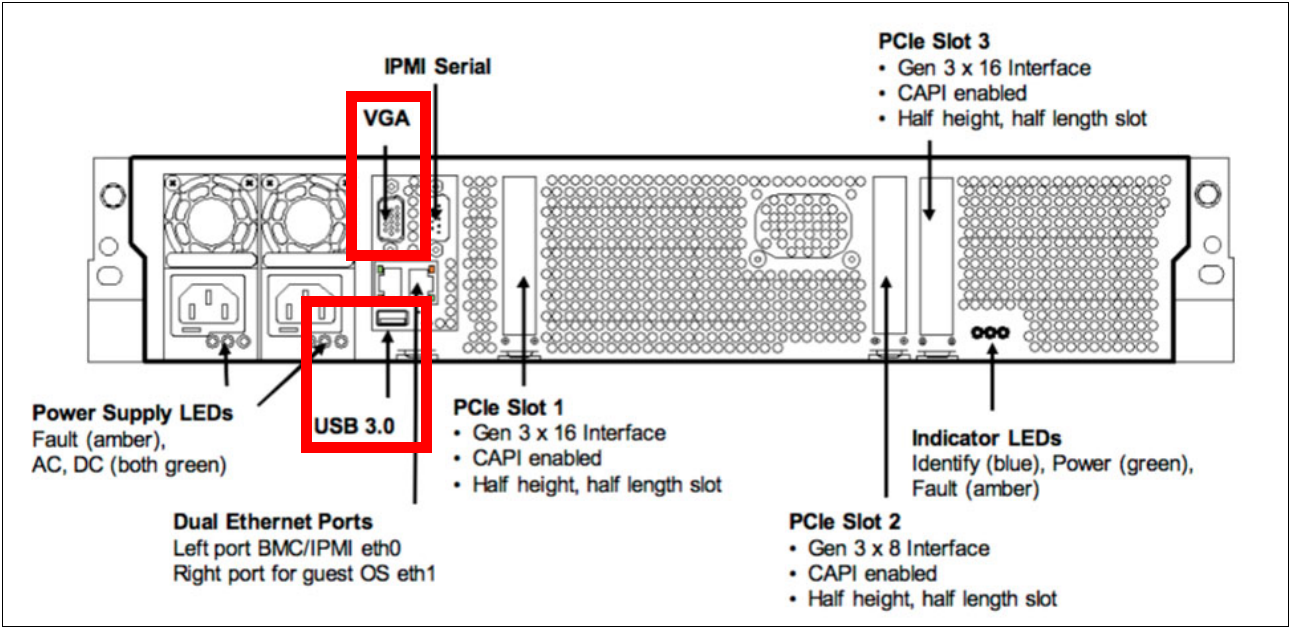
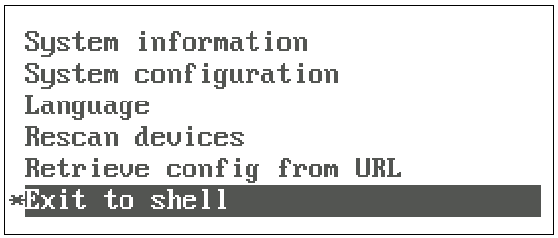


Figure 4-3 VGA and USB connection ports for terminal (IBM Power System S822LC for High Performance Computing rear panel)

[1] No-charge registration to the NVIDIA Accelerated Developer Program might be required.

Tip: The management processor's IP address is set to DHCP by default. Therefore, if there is a DHCP server that is prepared in the network, the IP address is automatically assigned to the management processor. For checking the IP address that is assigned by DHCP or assigning a static IP address, you must enter the shell from Petitboot (boot loader) and check and change the setting.

2. Power on the IBM Power System S822LC for High Performance Computing server.
3. Select `Exit to Shell` from the Petitboot window, as shown in Figure 4-4.



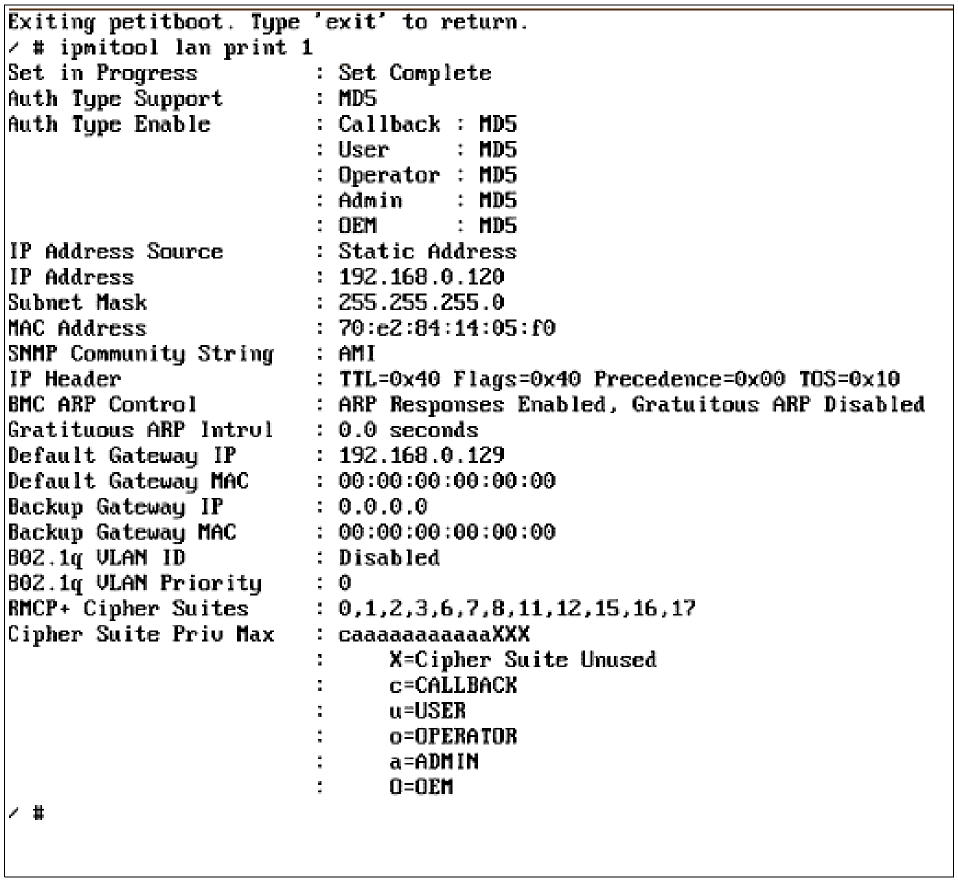
```
 System information
 System configuration
 Language
 Rescan devices
 Retrieve config from URL
*Exit to shell
```

Figure 4-4 Petitboot initial window

4. Use the **`ipmitool`** command to check or change the network settings that are assigned to the management processor:

   ```
   # ipmitool lan print 1
   ```

   If DHCP is set, check the IP address, as shown in Figure 4-5.



```
Exiting petitboot. Type 'exit' to return.
/ # ipmitool lan print 1
Set in Progress          : Set Complete
Auth Type Support        : MD5
Auth Type Enable         : Callback : MD5
                         : User     : MD5
                         : Operator : MD5
                         : Admin    : MD5
                         : OEM      : MD5
IP Address Source        : Static Address
IP Address               : 192.168.0.120
Subnet Mask              : 255.255.255.0
MAC Address              : 70:e2:84:14:05:f0
SNMP Community String    : AMI
IP Header                : TTL=0x40 Flags=0x40 Precedence=0x00 TOS=0x10
BMC ARP Control          : ARP Responses Enabled, Gratuitous ARP Disabled
Gratituous ARP Intrvl    : 0.0 seconds
Default Gateway IP       : 192.168.0.129
Default Gateway MAC      : 00:00:00:00:00:00
Backup Gateway IP        : 0.0.0.0
Backup Gateway MAC       : 00:00:00:00:00:00
802.1q VLAN ID           : Disabled
802.1q VLAN Priority     : 0
RMCP+ Cipher Suites      : 0,1,2,3,6,7,8,11,12,15,16,17
Cipher Suite Priv Max    : caaaaaaaaaaaXXX
                         :     X=Cipher Suite Unused
                         :     c=CALLBACK
                         :     u=USER
                         :     o=OPERATOR
                         :     a=ADMIN
                         :     O=OEM
/ #
```

Figure 4-5 The ipmitool lan print 1 command sample output

5. Change the IP address to static:

```
# ipmitool lan set 1 ipsrc static
```

6. Set up the IP, gateway, and network mask addresses:

```
# ipmitool lan set 1 ipaddr IP_ADDRESS
# ipmitool lan set 1 defgw ipaddr GATEWAY_ADDRESS
# ipmitool lan set 1 netmask NETMASK_ADDRESS
```

7. Confirm the settings:

```
# ipmitool lan print 1
```

8. Stop the system to restart the management processor:

```
# ipmitool power off
```

9. After the system stops, disconnect both power cords, and reconnect the power cords after 30 seconds.
10. Set the system power to ON.
11. Using the set IP address, confirm that you can access the BMC by using your browser[2] to connect to `https://IP_ADDRESS`.

Figure 4-6 shows the BMC/IPMI Ethernet ports for the IBM Power System S822LC for High Performance Computing server.

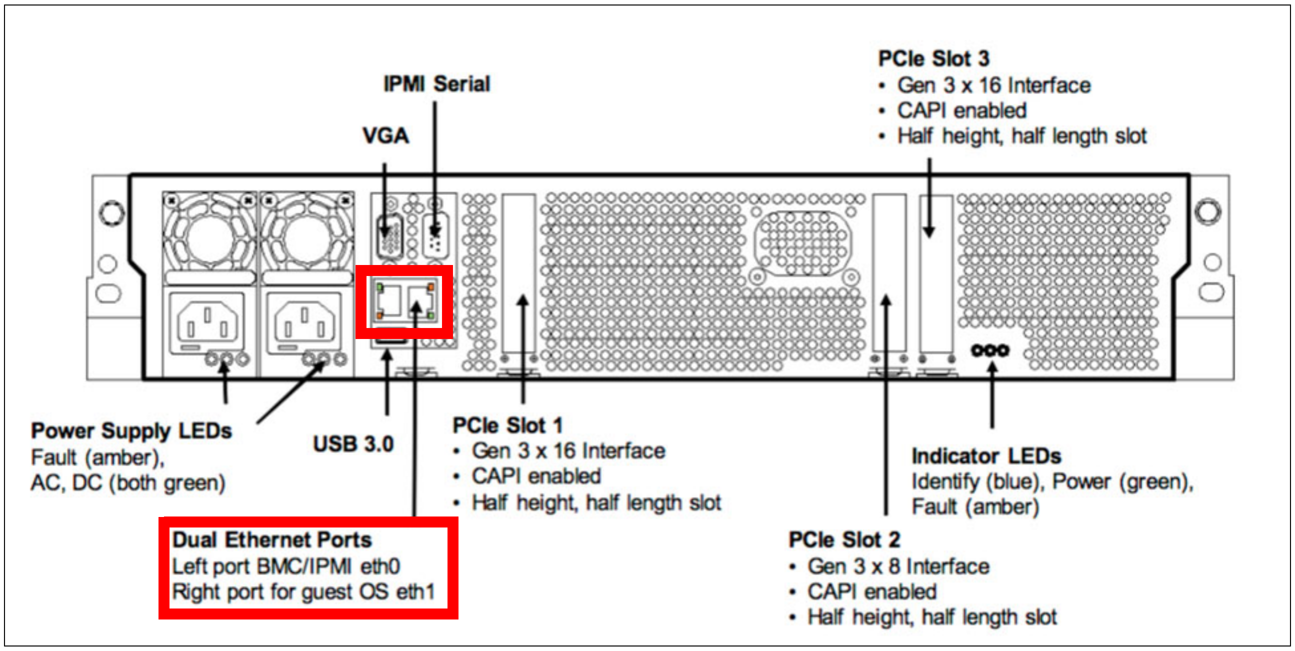


Figure 4-6 BMC/IPMI Ethernet port (IBM Power System S822LC for High Performance Computing rear panel)

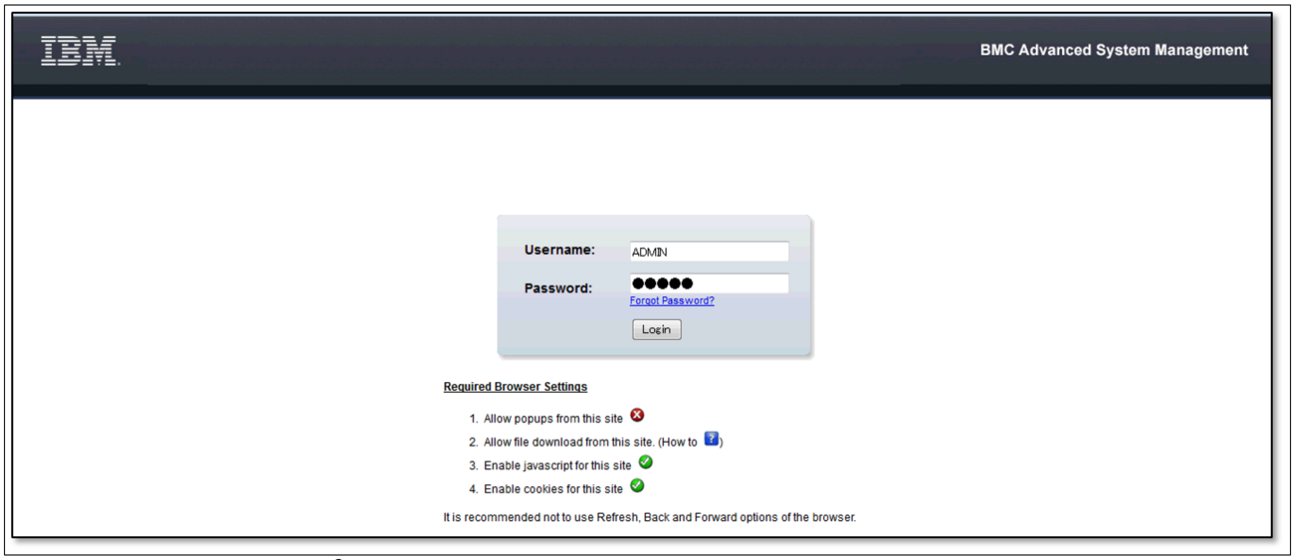
How to access the baseboard management controller

To access the BMC, complete the following steps:

1. Use your browser to connect to `https://IP_ADDRESS`.

   Figure 4-7 on page 65 shows the BMC login window.

[2] To access the BMC by using the browser, it is necessary to connect this port and the working terminal to the same network.



Username: ADMIN

Password: ●●●●●

Forgot Password?

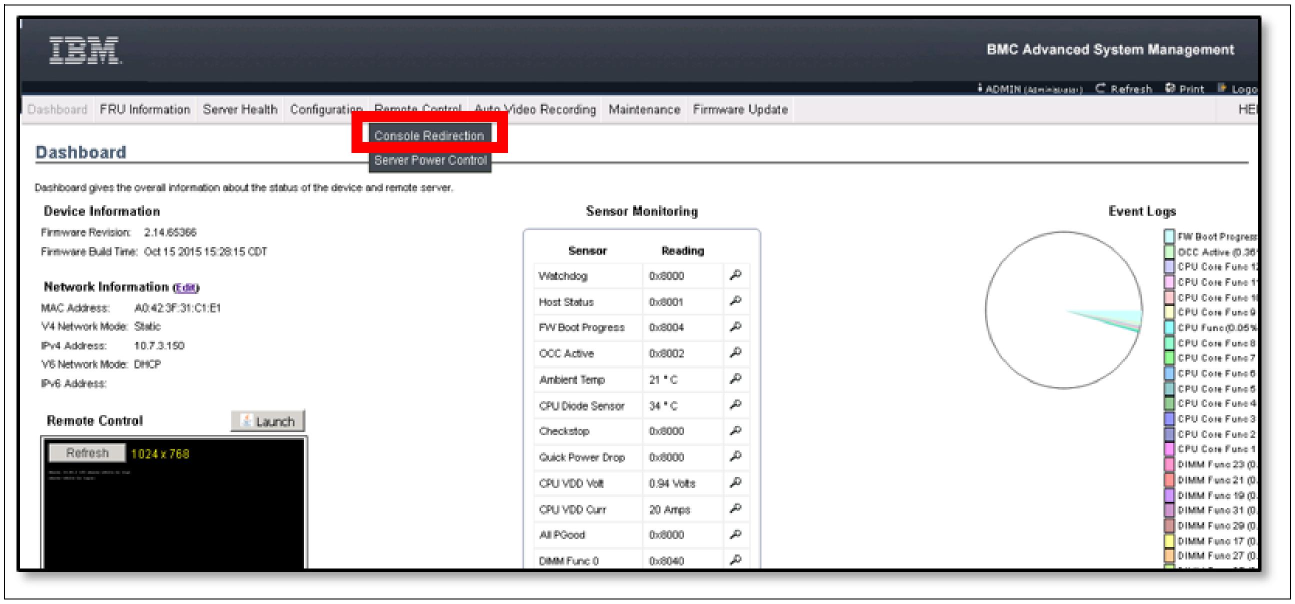
Required Browser Settings

1. Allow popups from this site
2. Allow file download from this site. (How to)
3. Enable javascript for this site
4. Enable cookies for this site

It is recommended not to use Refresh, Back and Forward options of the browser.

Figure 4-7 BMC login window[3]

2. Log in to the system with the following values (these are the default values):
   – User name: `ADMIN`
   – Password: `admin`
3. Access the remote Java-based console:
   a. Select **Remote Control**, and then **Console Redirection**, as shown in Figure 4-8.



Dashboard

Dashboard gives the overall information about the status of the device and remote server.

Device Information

Firmware Revision: 2.14.65366

Firmware Build Time: Oct 15 2015 15:28:15 CDT

Network Information (Edit)

MAC Address: A0:42:3F:31:C1:E1

V4 Network Mode: Static

IPv4 Address: 10.7.3.150

V6 Network Mode: DHCP

IPv6 Address:

Remote Control

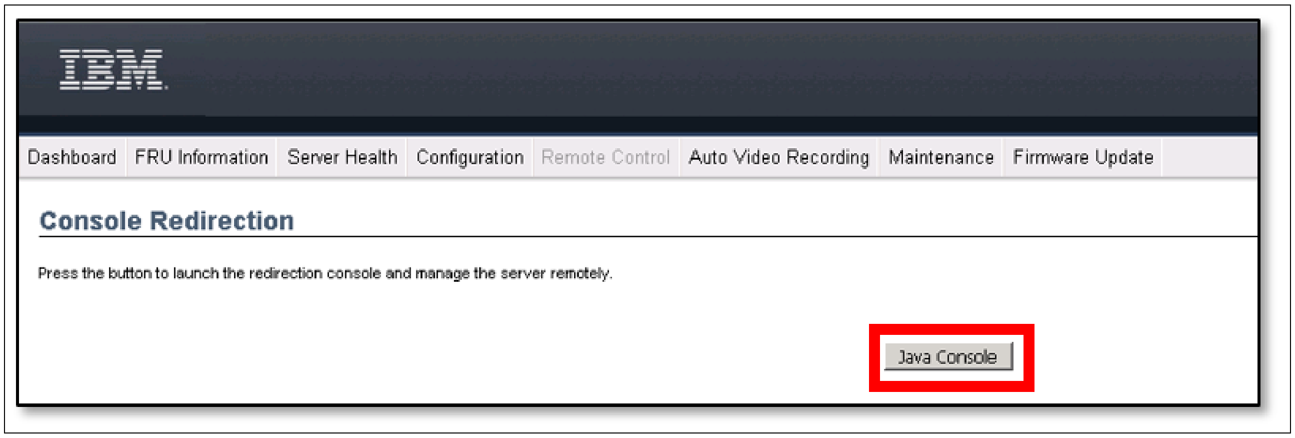
Sensor Monitoring

Sensor	Reading
Watchdog	0x8000
Host Status	0x8001
FW Boot Progress	0x8004
OCC Active	0x8002
Ambient Temp	21 °C
CPU Diode Sensor	34 °C
Checkstop	0x8000
Quick Power Drop	0x8000
CPU VDD Volt	0.94 Volts
CPU VDD Curr	20 Amps
All PGood	0x8000
DIMM Func 0	0x8040

Figure 4-8 Enabling console redirection

[3] In this screen capture, pop-up windows are not permitted in the browser.

b. Select **Java Console**, as shown in Figure 4-9.



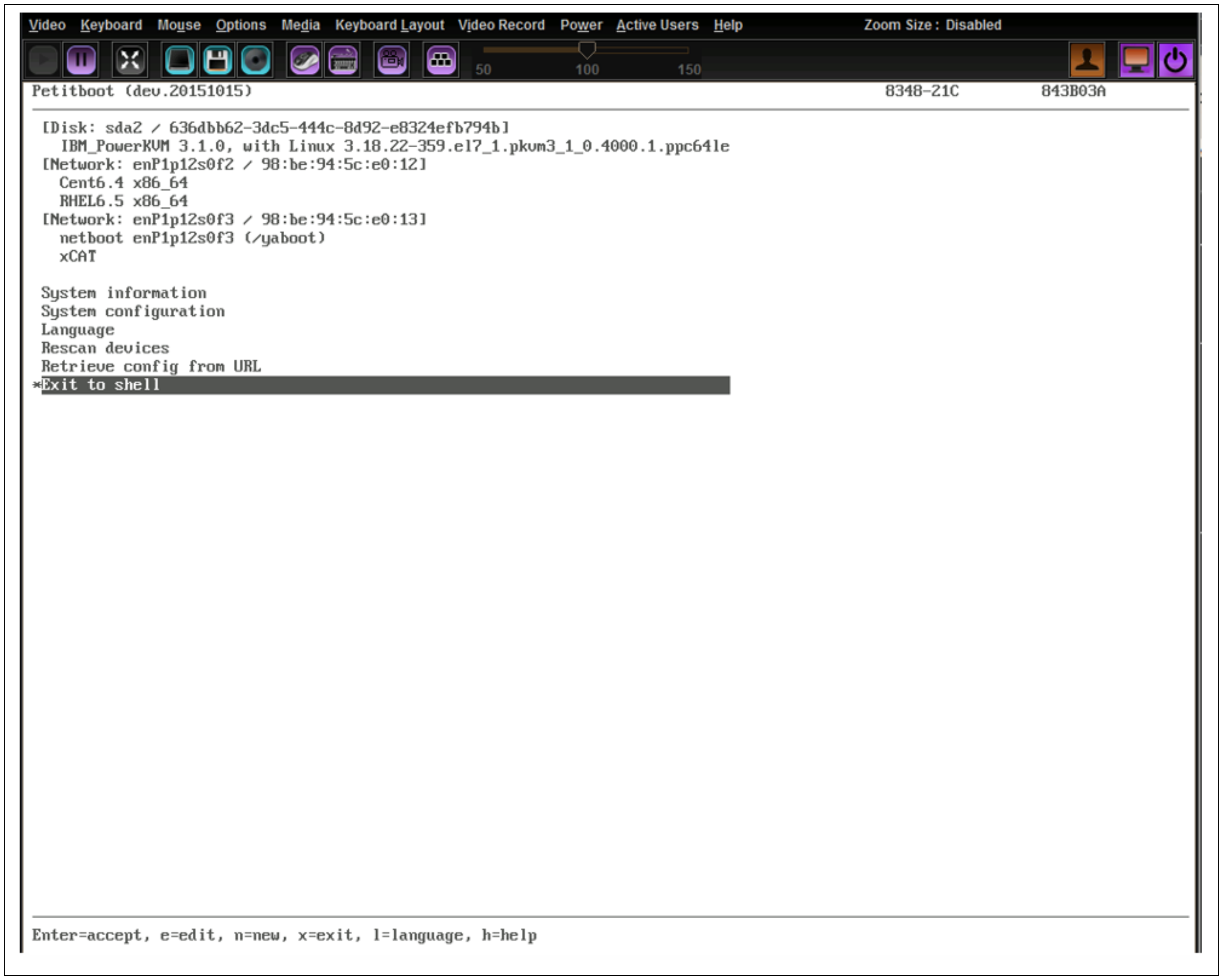
Console Redirection

Press the button to launch the redirection console and manage the server remotely.

Figure 4-9 Opening Java Console

c. Accept the Java jviewer.jnpl dialog.

The Java application starts and the console window, opens, as shown in Figure 4-10.



```
Petitboot (dev.20151015)                                                  8348-21C        843B03A

 [Disk: sda2 / 636dbb62-3dc5-444c-8d92-e8324efb794b]
   IBM_PowerKVM 3.1.0, with Linux 3.18.22-359.el7_1.pkvm3_1_0.4000.1.ppc64le
 [Network: enP1p12s0f2 / 98:be:94:5c:e0:12]
   Cent6.4 x86_64
   RHEL6.5 x86_64
 [Network: enP1p12s0f3 / 98:be:94:5c:e0:13]
   netboot enP1p12s0f3 (/yaboot)
   xCAT

 System information
 System configuration
 Language
 Rescan devices
 Retrieve config from URL
*Exit to shell

Enter=accept, e=edit, n=new, x=exit, l=language, h=help
```

Figure 4-10 Sample Java Console

4.1.4 Installing Ubuntu 16.04.x

This section provides details about an Ubuntu 16.04.x installation.

Attaching a virtual media (a .iso file as a CD/DVD)

This section describes how to attach virtual media. Complete the following steps:

1. In the Java Console application menu, select **Media →Virtual Media Wizard**, as shown in Figure 4-11.

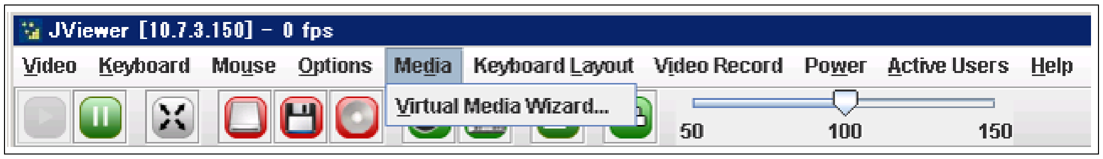


Figure 4-11 Opening the Virtual Media Wizard

2. On the computer accessing the BMC, select the `.iso` file previously downloaded (“Downloads” on page 60) and assign as a CD/DVD:

   a. Click **Browse** for CD/DVD Media: 1, as shown in Figure 4-12.

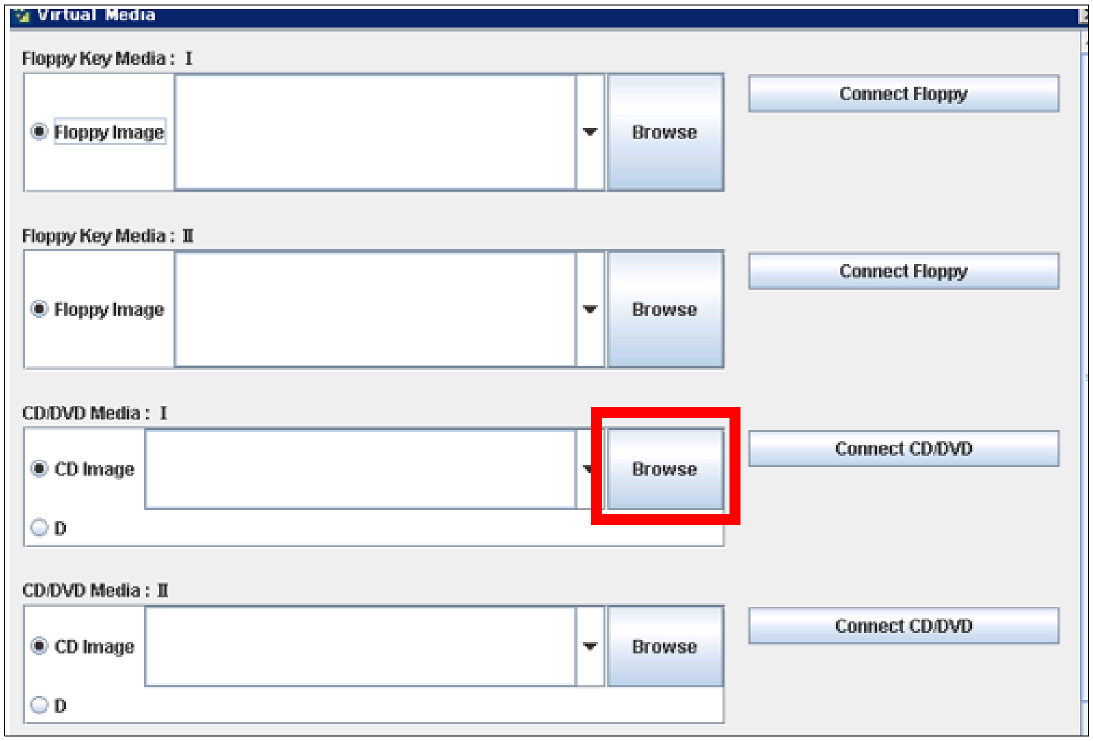


Figure 4-12 Connecting a CD/DVD

   b. Click **Open** and select a file.

3. Select **Connect CD/DVD**, as shown in Figure 4-13.

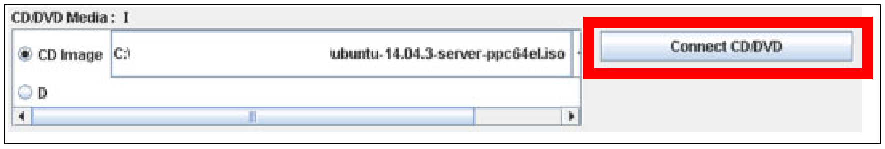


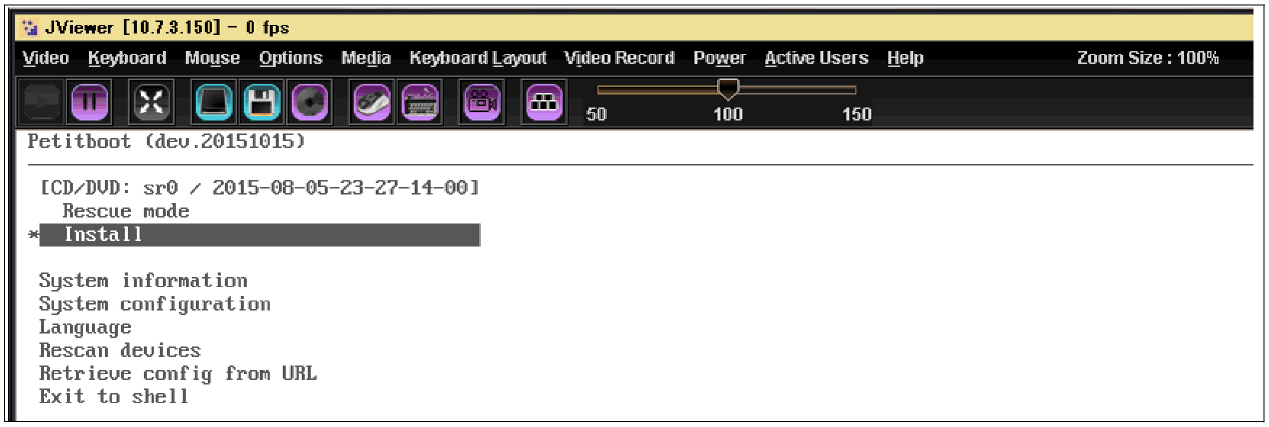
Figure 4-13 Clicking Connect CD/DVD

4. Click **Close** to complete the attachment.

Installing Ubuntu 16.04[4]

This section describes the steps for installing Ubuntu 16.04:

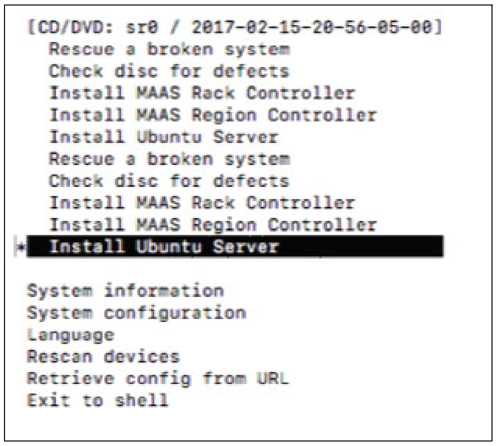
1. Power on the server by using the red icon at the upper right corner of the console window or under the **Power** option menu (**Power On Server** option).
2. The system starts and the Petitboot boot menu is available. Select the Ubuntu installation media, as shown in Figure 4-14.


JViewer [10.7.3.150] - 0 fps
Video Keyboard Mouse Options Media Keyboard Layout Video Record Power Active Users Help
Zoom Size : 100%
50 100 150

```
Petitboot (dev.20151015)

 [CD/DVD: sr0 / 2015-08-05-23-27-14-00]
   Rescue mode
*  Install

 System information
 System configuration
 Language
 Rescan devices
 Retrieve config from URL
 Exit to shell
```

Figure 4-14 Petitboot boot window

3. Click `Install Ubuntu Server`, as shown Figure 4-15.

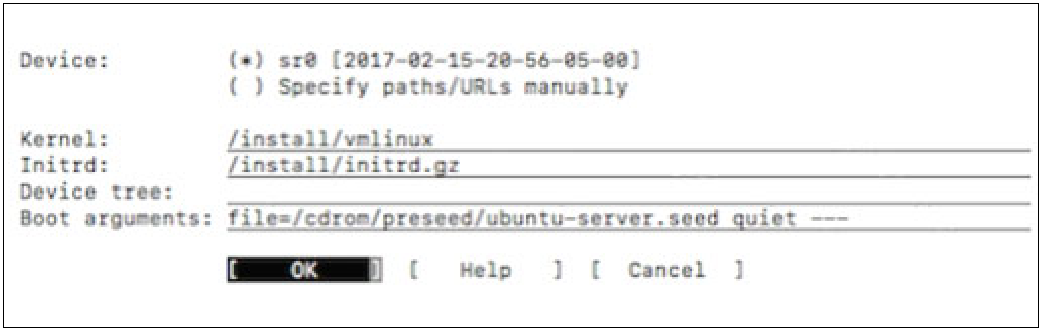


```
[CD/DVD: sr0 / 2017-02-15-20-56-05-00]
  Rescue a broken system
  Check disc for defects
  Install MAAS Rack Controller
  Install MAAS Region Controller
  Install Ubuntu Server
  Rescue a broken system
  Check disc for defects
  Install MAAS Rack Controller
  Install MAAS Region Controller
* Install Ubuntu Server

System information
System configuration
Language
Rescan devices
Retrieve config from URL
Exit to shell
```

Figure 4-15 Install Ubuntu Server

4 This procedure was checked for Ubuntu 16.04.01 and 16.04.02.

4. Click `E` and select kernel,[5] as shown in Figure 4-16.



```
Device:         (*) sr0 [2017-02-15-20-56-05-00]
                ( ) Specify paths/URLs manually

Kernel:         /install/vmlinux
Initrd:         /install/initrd.gz
Device tree:
Boot arguments: file=/cdrom/preseed/ubuntu-server.seed quiet ---

                [    OK    ]  [   Help   ]  [  Cancel   ]
```

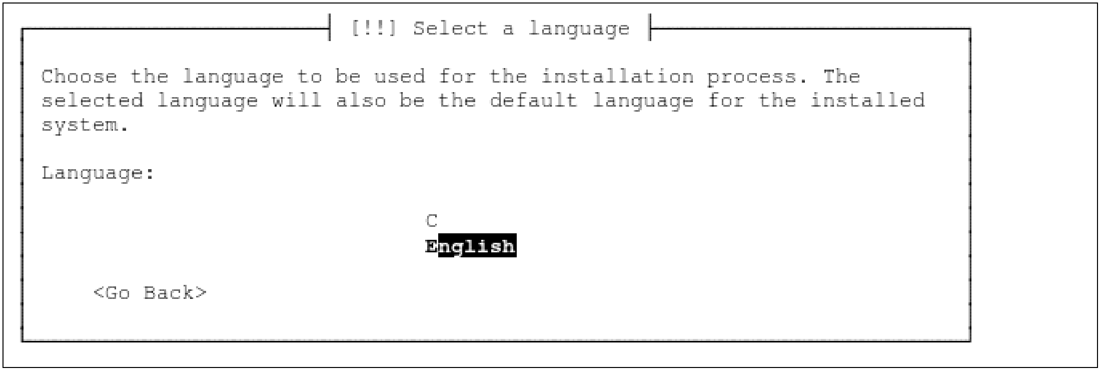
Figure 4-16 Select kernel

Tip: When the installer starts, the window session of the remote control can be disconnected. In that case, switch from the working terminal to **`ipmitool`** on Serial over LAN with the following command:

```
# ipmitool -I lanplus -H BMC_IPADDRESS -U USER -P PASSWORD sol activate
```

Note: Ubuntu LTS installation media can also be installed by using Hardware Enablement (HWE) in addition to the normal kernel.

5. In the language selection, click `English`, as shown in Figure 4-17.



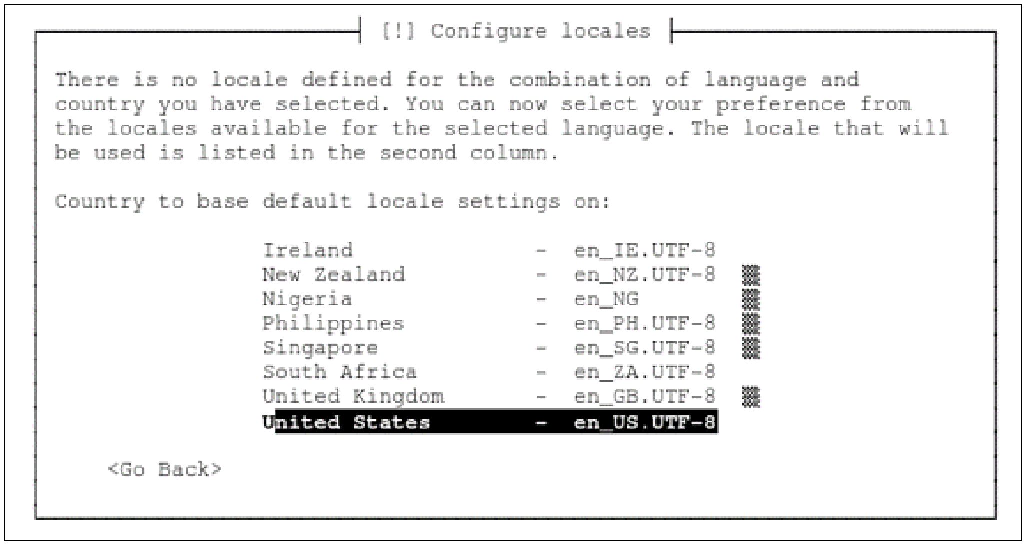
```
┌───────────────────────┤ [!!] Select a language ├───────────────────────┐
│                                                                        │
│ Choose the language to be used for the installation process. The       │
│ selected language will also be the default language for the installed  │
│ system.                                                                │
│                                                                        │
│ Language:                                                              │
│                                                                        │
│                               C                                        │
│                               English                                  │
│                                                                        │
│     <Go Back>                                                          │
│                                                                        │
└────────────────────────────────────────────────────────────────────────┘
```

Figure 4-17 Selecting a language

6. Select the location of your preference.

[5] Select normal kernel, not hwe-vmlinux.

7. Configure the system locale, as shown in Figure 4-18.



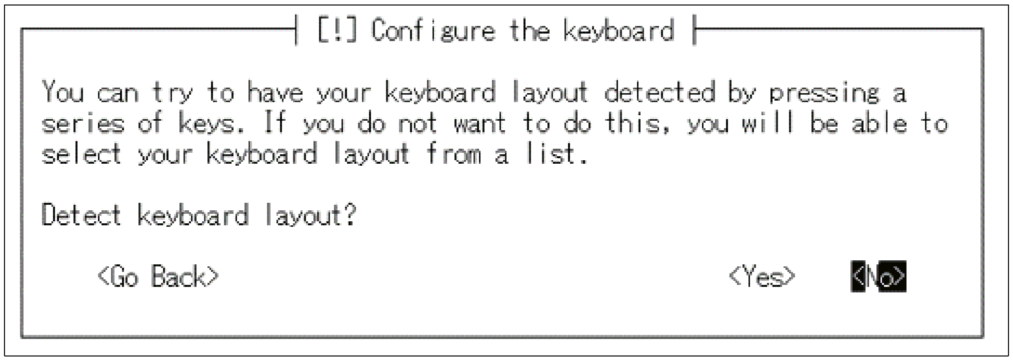
```
[!] Configure locales

There is no locale defined for the combination of language and
country you have selected. You can now select your preference from
the locales available for the selected language. The locale that will
be used is listed in the second column.

Country to base default locale settings on:

                Ireland              -  en_IE.UTF-8
                New Zealand          -  en_NZ.UTF-8
                Nigeria              -  en_NG
                Philippines          -  en_PH.UTF-8
                Singapore            -  en_SG.UTF-8
                South Africa         -  en_ZA.UTF-8
                United Kingdom       -  en_GB.UTF-8
                United States        -  en_US.UTF-8

    <Go Back>
```

Figure 4-18 Selecting a locale

Note: To avoid possible conflicts with the locale settings of some of the frameworks that are deployed by IBM PowerAI, choose `United States - en_US_UTF-8` or any other UTF-8 locale as the default locale.

8. Set up the keyboard:

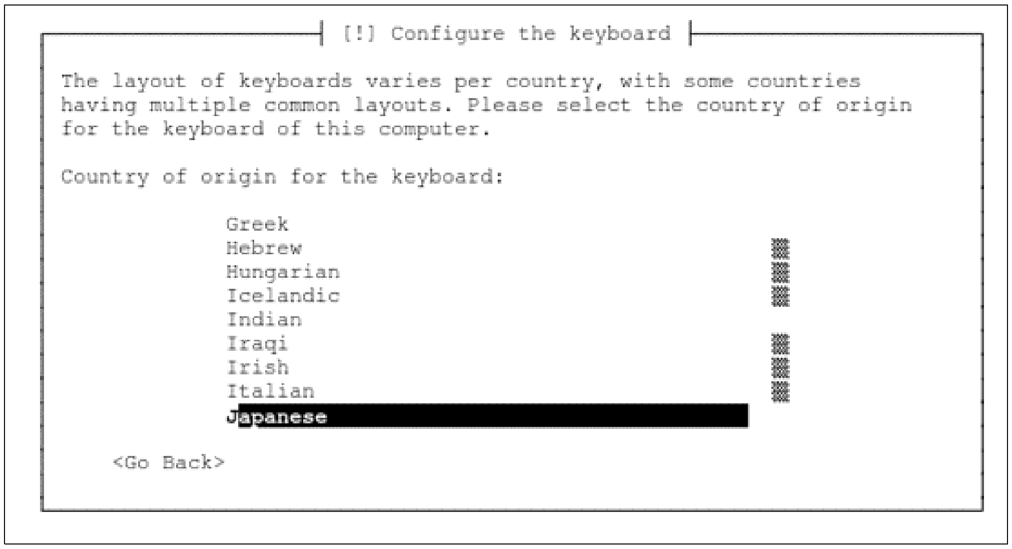
   a. When asked `Detect Keyboard layout?`, click `No`, as shown in Figure 4-19.



```
[!] Configure the keyboard

You can try to have your keyboard layout detected by pressing a
series of keys. If you do not want to do this, you will be able to
select your keyboard layout from a list.

Detect keyboard layout?

    <Go Back>                                      <Yes>    <No>
```

Figure 4-19 Automatic keyboard selection

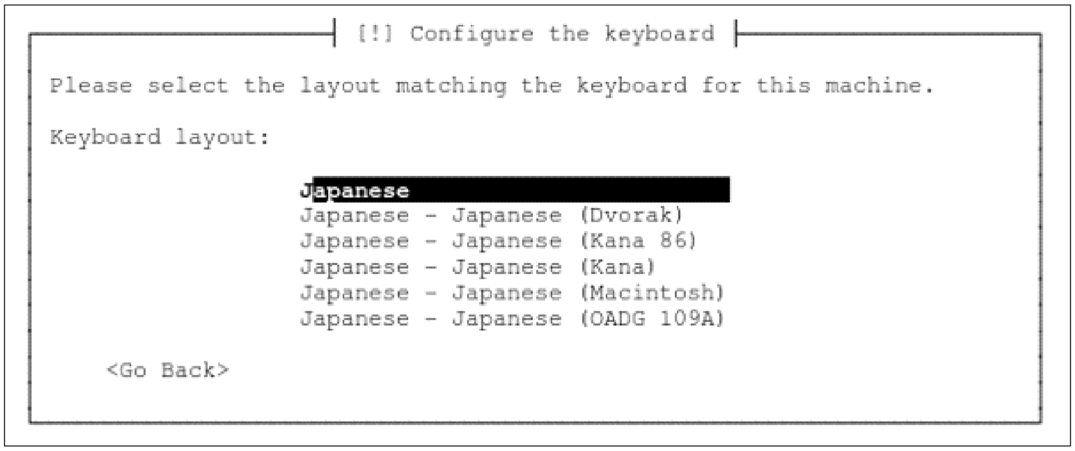
   b. In Keyboard layout, select the keyboard of your preference, as shown in Figure 4-20 on page 71.



```
[!] Configure the keyboard

The layout of keyboards varies per country, with some countries
having multiple common layouts. Please select the country of origin
for the keyboard of this computer.

Country of origin for the keyboard:

             Greek
             Hebrew
             Hungarian
             Icelandic
             Indian
             Iraqi
             Irish
             Italian
             Japanese

    <Go Back>
```

Figure 4-20   Selecting the keyboard layout

Figure 4-21 shows the keyboard selection.



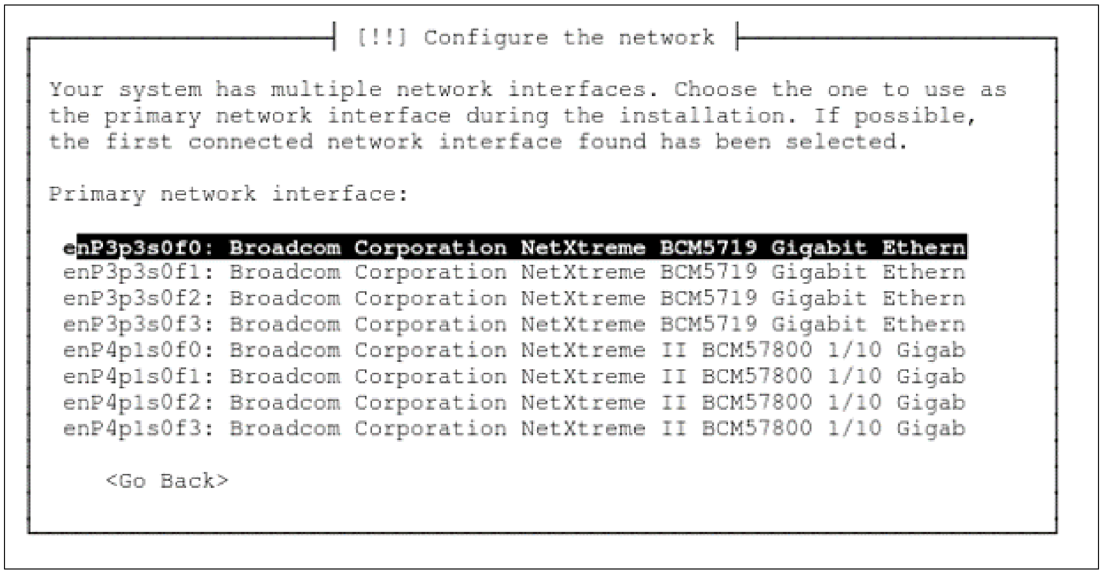
```
[!] Configure the keyboard

Please select the layout matching the keyboard for this machine.

Keyboard layout:

                 Japanese
                 Japanese - Japanese (Dvorak)
                 Japanese - Japanese (Kana 86)
                 Japanese - Japanese (Kana)
                 Japanese - Japanese (Macintosh)
                 Japanese - Japanese (OADG 109A)

    <Go Back>
```

Figure 4-21   Selecting the keyboard layout

Network setup

This section describes the steps to configure the network setup:

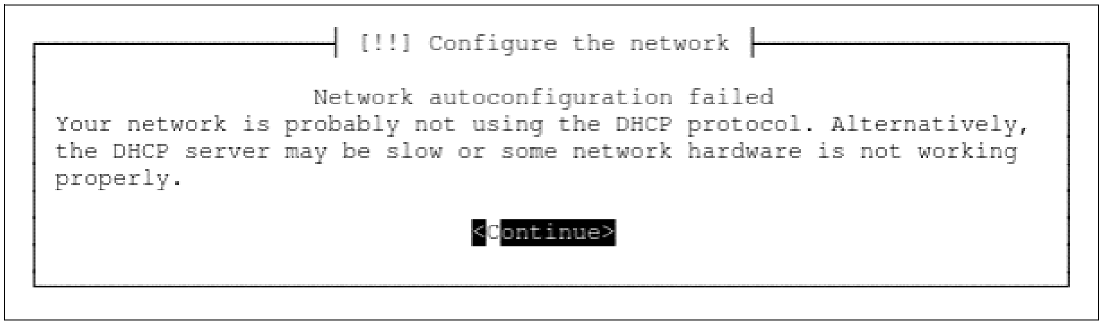
1. Select the network interface to be set as the primary network,[6] as shown in Figure 4-22.



```
[!!] Configure the network

Your system has multiple network interfaces. Choose the one to use as
the primary network interface during the installation. If possible,
the first connected network interface found has been selected.

Primary network interface:

 enP3p3s0f0: Broadcom Corporation NetXtreme BCM5719 Gigabit Ethern
 enP3p3s0f1: Broadcom Corporation NetXtreme BCM5719 Gigabit Ethern
 enP3p3s0f2: Broadcom Corporation NetXtreme BCM5719 Gigabit Ethern
 enP3p3s0f3: Broadcom Corporation NetXtreme BCM5719 Gigabit Ethern
 enP4p1s0f0: Broadcom Corporation NetXtreme II BCM57800 1/10 Gigab
 enP4p1s0f1: Broadcom Corporation NetXtreme II BCM57800 1/10 Gigab
 enP4p1s0f2: Broadcom Corporation NetXtreme II BCM57800 1/10 Gigab
 enP4p1s0f3: Broadcom Corporation NetXtreme II BCM57800 1/10 Gigab

    <Go Back>
```

Figure 4-22 Selecting the primary network interface

2. Configure the interface if no DHCP server is available in the network. The message that is shown in Figure 4-24 on page 73 is displayed. Manually continue to configure the network.

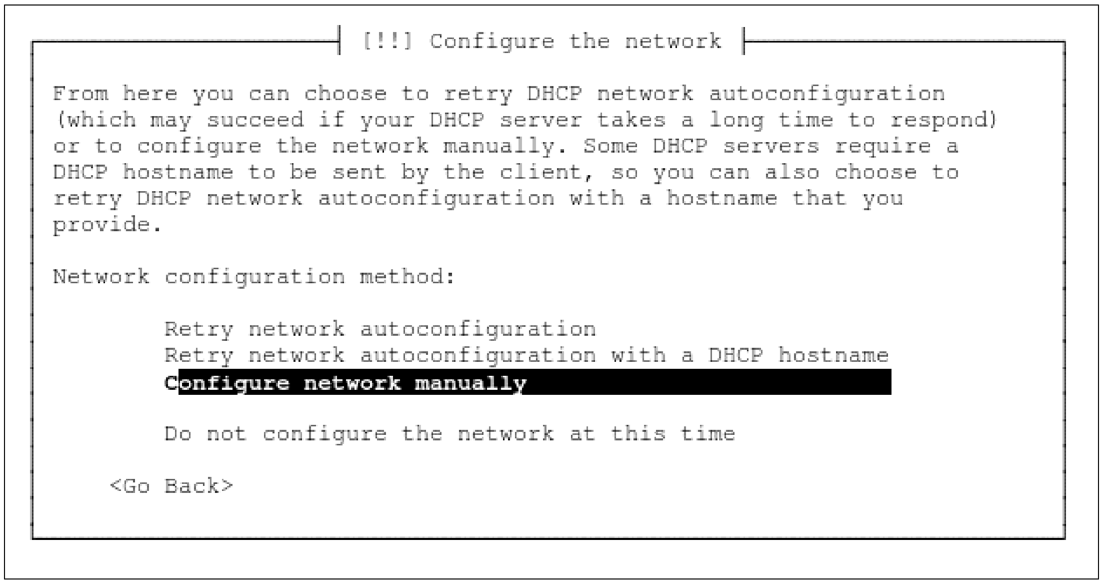


```
[!!] Configure the network

                 Network autoconfiguration failed
Your network is probably not using the DHCP protocol. Alternatively,
the DHCP server may be slow or some network hardware is not working
properly.

                            <Continue>
```

Figure 4-23 No DHCP server is present

3. Begin the manual network setup, as shown in Figure 4-24 on page 73.

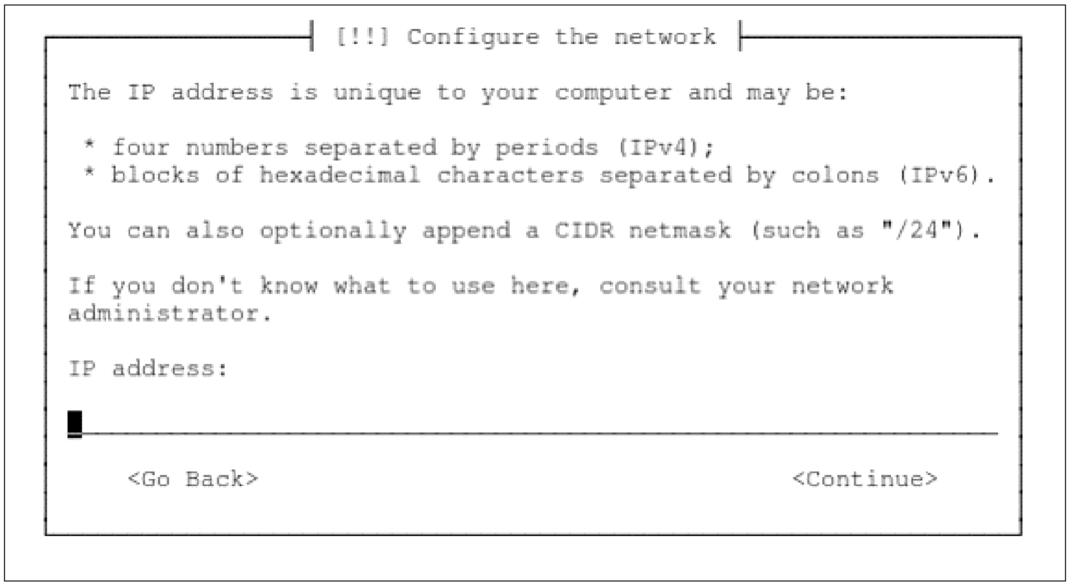
[6] If a DHCP server is present, after selecting the network interface, the setting of the primary network is completed automatically.



```
[!!] Configure the network

From here you can choose to retry DHCP network autoconfiguration
(which may succeed if your DHCP server takes a long time to respond)
or to configure the network manually. Some DHCP servers require a
DHCP hostname to be sent by the client, so you can also choose to
retry DHCP network autoconfiguration with a hostname that you
provide.

Network configuration method:

        Retry network autoconfiguration
        Retry network autoconfiguration with a DHCP hostname
        Configure network manually

        Do not configure the network at this time

    <Go Back>
```

Figure 4-24 Configuring the network manually

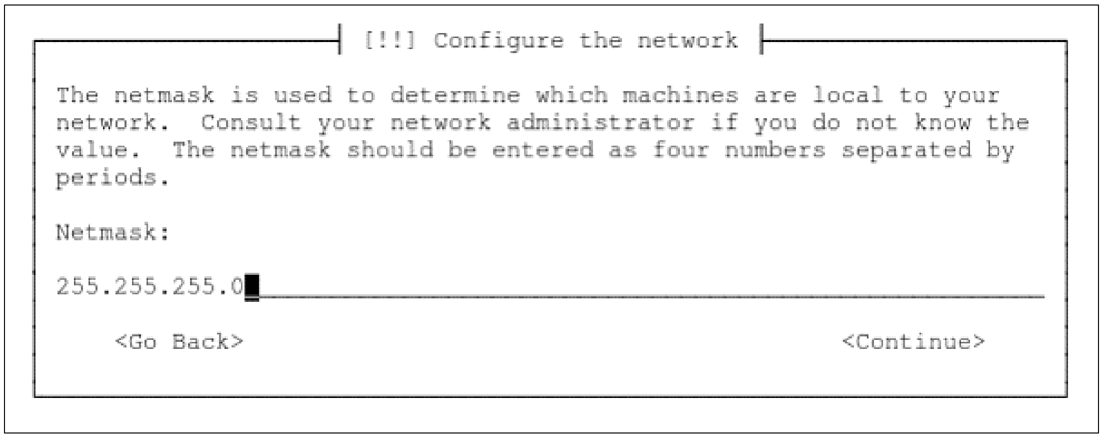
4. Configure the IP address, as shown in Figure 4-25.



```
[!!] Configure the network

The IP address is unique to your computer and may be:

 * four numbers separated by periods (IPv4);
 * blocks of hexadecimal characters separated by colons (IPv6).

You can also optionally append a CIDR netmask (such as "/24").

If you don't know what to use here, consult your network
administrator.

IP address:

_______________________________________________________________

    <Go Back>                                      <Continue>
```

Figure 4-25 Configuring the IP

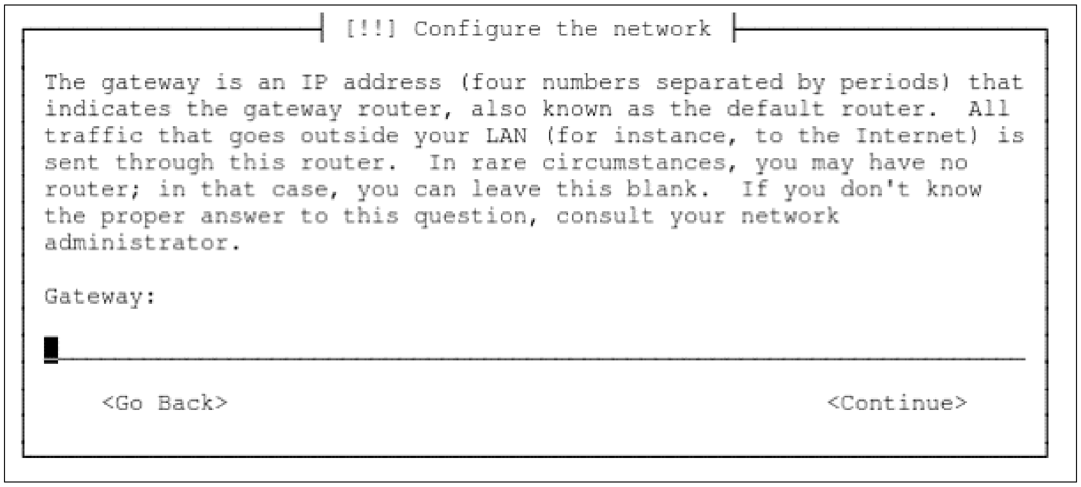
5. Configuring the netmask, as shown in Figure 4-26.



```
[!!] Configure the network

The netmask is used to determine which machines are local to your
network.  Consult your network administrator if you do not know the
value.  The netmask should be entered as four numbers separated by
periods.

Netmask:

255.255.255.0

    <Go Back>                                          <Continue>
```

Figure 4-26 Configuring the netmask

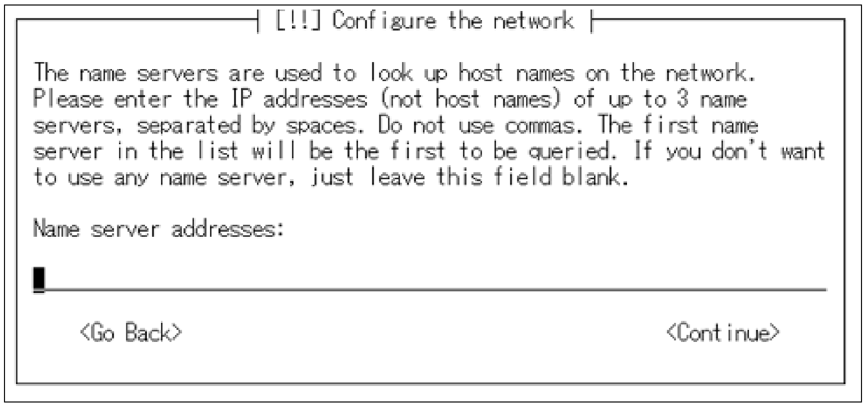
6. Configuring the default gateway, as shown in Figure 4-27.



```
[!!] Configure the network

The gateway is an IP address (four numbers separated by periods) that
indicates the gateway router, also known as the default router.  All
traffic that goes outside your LAN (for instance, to the Internet) is
sent through this router.  In rare circumstances, you may have no
router; in that case, you can leave this blank.  If you don't know
the proper answer to this question, consult your network
administrator.

Gateway:


    <Go Back>                                          <Continue>
```

Figure 4-27 Configuring the default gateway

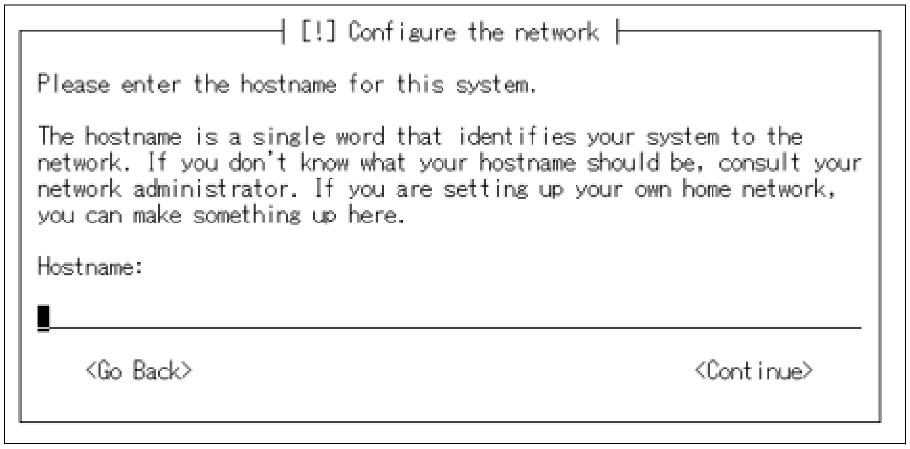
7. Configuring the name server (DNS), as shown in Figure 4-28.



```
[!!] Configure the network

The name servers are used to look up host names on the network.
Please enter the IP addresses (not host names) of up to 3 name
servers, separated by spaces. Do not use commas. The first name
server in the list will be the first to be queried. If you don't want
to use any name server, just leave this field blank.

Name server addresses:


    <Go Back>                                    <Continue>
```

Figure 4-28 Configuring the name server

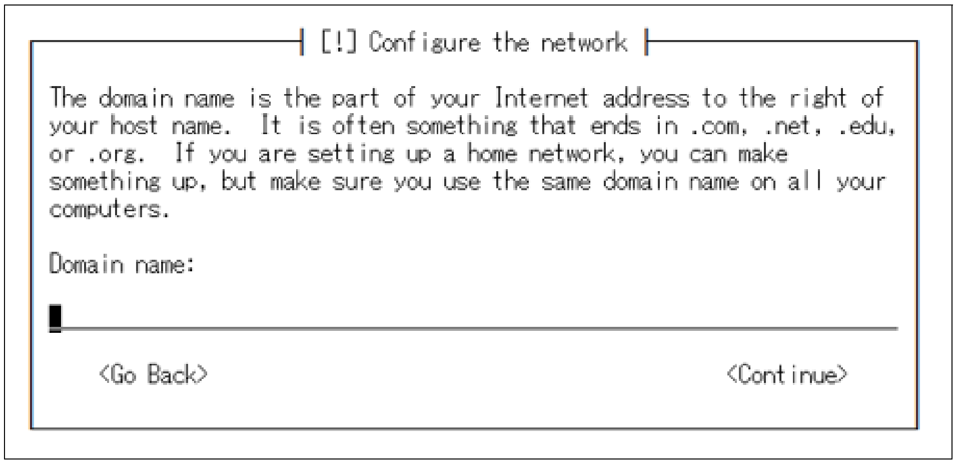
8. Choose a host name, as shown in Figure 4-29 on page 75.



```
[!] Configure the network

Please enter the hostname for this system.

The hostname is a single word that identifies your system to the
network. If you don't know what your hostname should be, consult your
network administrator. If you are setting up your own home network,
you can make something up here.

Hostname:

    <Go Back>                                       <Continue>
```

Figure 4-29 Choosing a host name

9. Configure the domain name, as shown in Figure 4-30.



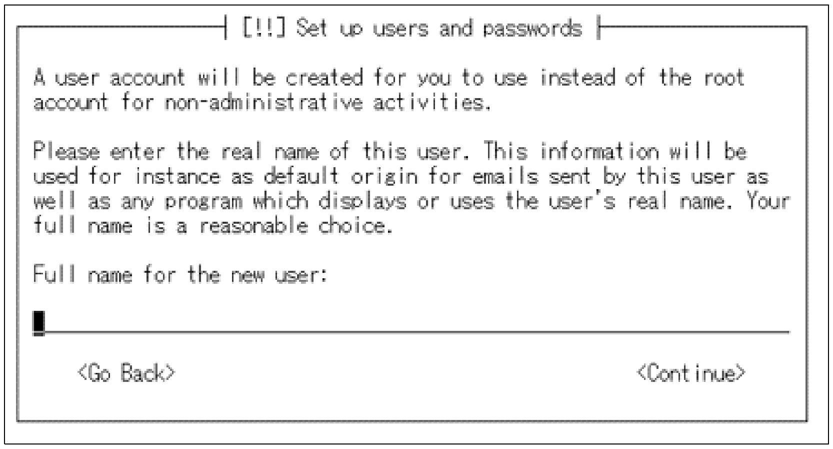
```
[!] Configure the network

The domain name is the part of your Internet address to the right of
your host name.  It is often something that ends in .com, .net, .edu,
or .org.  If you are setting up a home network, you can make
something up, but make sure you use the same domain name on all your
computers.

Domain name:

    <Go Back>                                       <Continue>
```

Figure 4-30 Configuring the domain name

Creating the default user

This section explains how to create the default user:

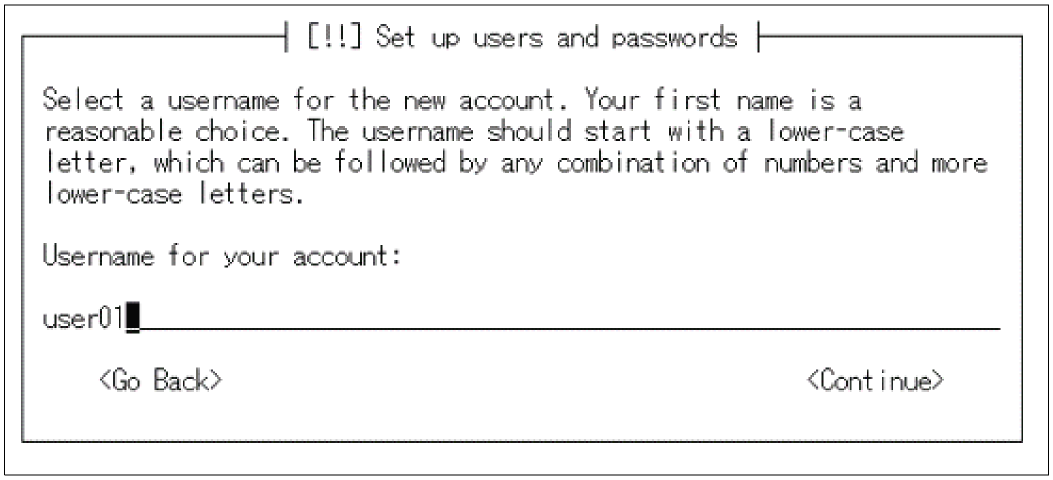
1. Set the full name and user name of the new user, as shown in Figure 4-31.



```
[!!] Set up users and passwords

A user account will be created for you to use instead of the root
account for non-administrative activities.

Please enter the real name of this user. This information will be
used for instance as default origin for emails sent by this user as
well as any program which displays or uses the user's real name. Your
full name is a reasonable choice.

Full name for the new user:

    <Go Back>                                       <Continue>
```

Figure 4-31 Providing the full name for the default user

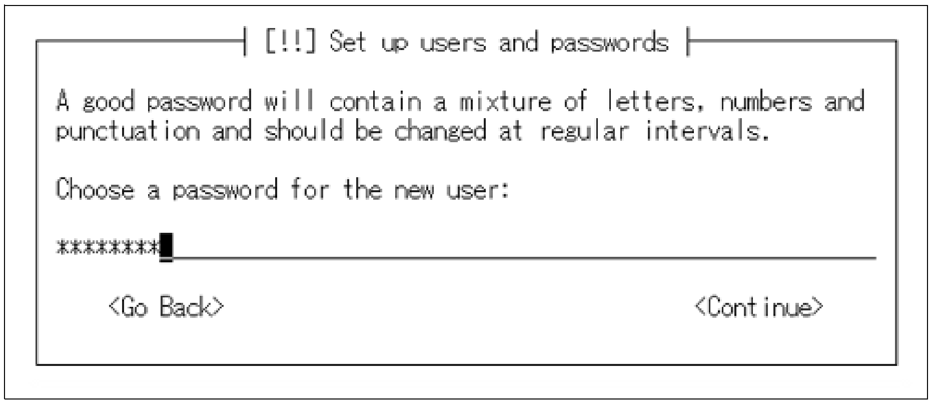
Figure 4-32 shows setting the default user name for the account.



```
[!!] Set up users and passwords

Select a username for the new account. Your first name is a
reasonable choice. The username should start with a lower-case
letter, which can be followed by any combination of numbers and more
lower-case letters.

Username for your account:

user01

    <Go Back>                                        <Continue>
```

Figure 4-32 Choosing the user name for the default user

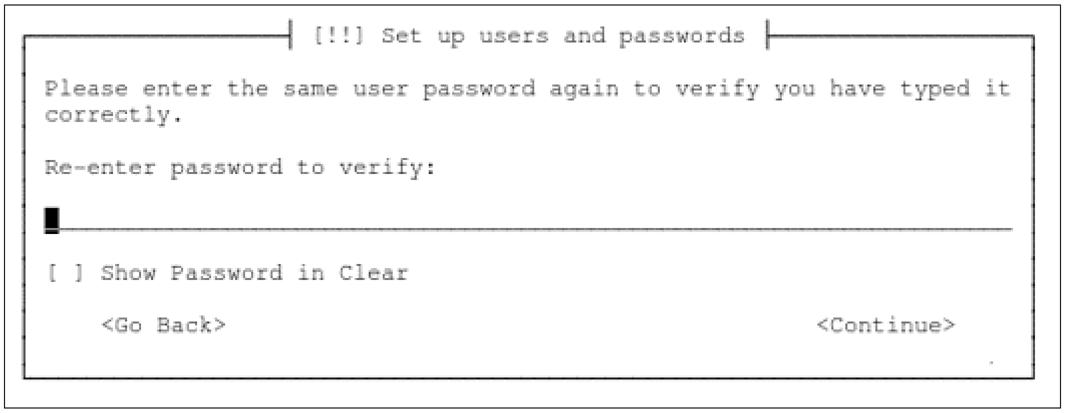
2. Choose a password for the default user, as shown in Figure 4-33.



```
[!!] Set up users and passwords

A good password will contain a mixture of letters, numbers and
punctuation and should be changed at regular intervals.

Choose a password for the new user:

********

    <Go Back>                                  <Continue>
```

Figure 4-33 Choosing a password

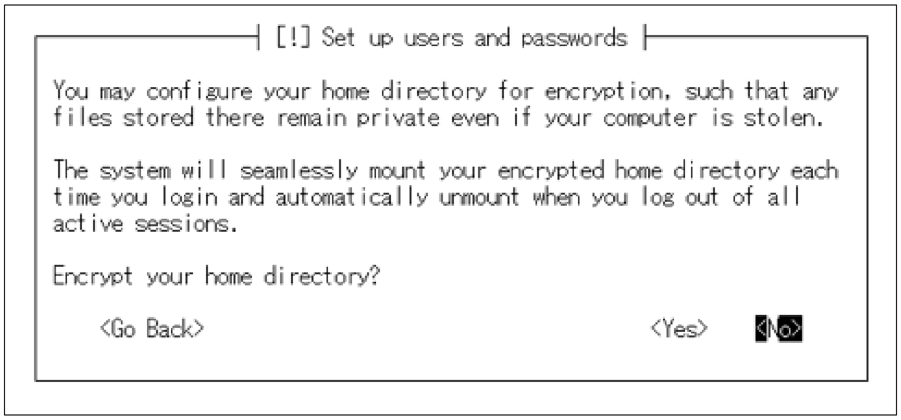
Figure 4-34 shows reentering the password for the default user account.



```
[!!] Set up users and passwords

Please enter the same user password again to verify you have typed it
correctly.

Re-enter password to verify:


[ ] Show Password in Clear

    <Go Back>                                          <Continue>
```

Figure 4-34 Confirming the password

3. You can choose to encrypt `$HOME` for this user, but we choose not to encrypt it in this example, as shown in Figure 4-35 on page 77.



```
[!] Set up users and passwords

You may configure your home directory for encryption, such that any
files stored there remain private even if your computer is stolen.

The system will seamlessly mount your encrypted home directory each
time you login and automatically unmount when you log out of all
active sessions.

Encrypt your home directory?

    <Go Back>                                        <Yes>    <No>
```

Figure 4-35 $HOME directory encryption

Preparing disks for a RAID array

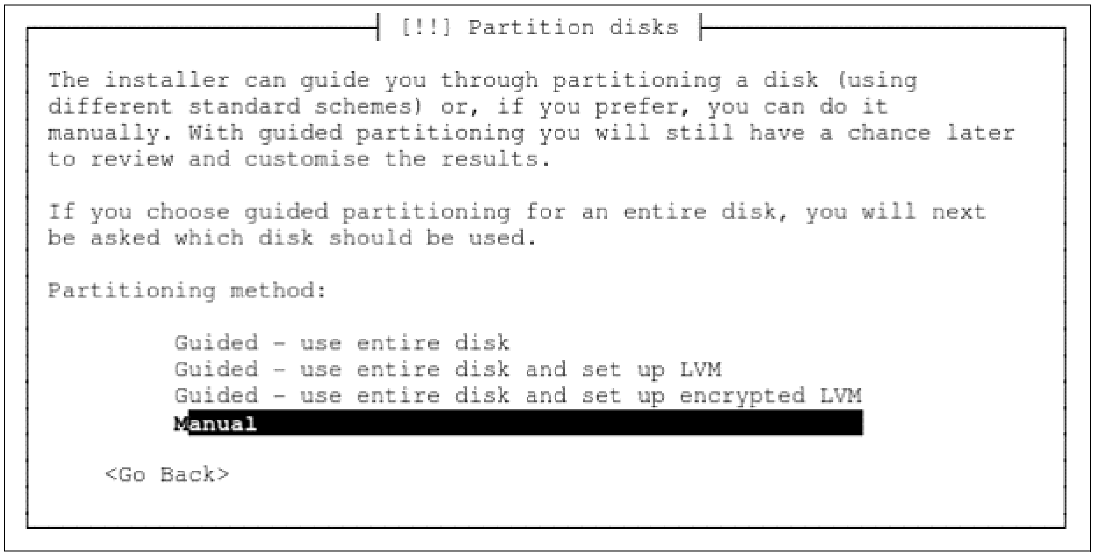
As the onboard SATA adapter does not provide any hardware RAID features, the following steps can be taken to configure software RAID. Although software RAID is not a requirement of IBM PowerAI, it is a preferred practice to mitigate against local disk failure. Creating a RAID array device is not mandatory for the installation of IBM PowerAI, although it is preferred because it provides high availability (HA) if there is a failure of one of the disks (SDDs or hard disk drives (HDDs)) that are part of the RAID array.

For more information, see the Ubuntu wiki.

If you choose not to define the RAID device, continue on to Figure 4-55 on page 86.

Complete the following steps:

1. Select `Manual` as the partitioning method, as shown in Figure 4-36.



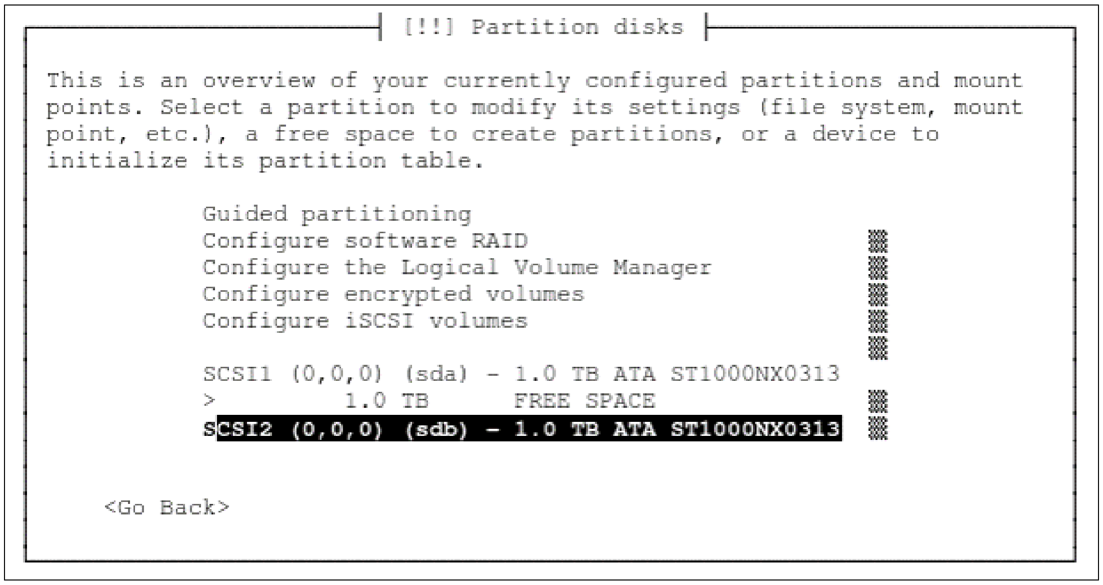
```
[!!] Partition disks

The installer can guide you through partitioning a disk (using
different standard schemes) or, if you prefer, you can do it
manually. With guided partitioning you will still have a chance later
to review and customise the results.

If you choose guided partitioning for an entire disk, you will next
be asked which disk should be used.

Partitioning method:

         Guided - use entire disk
         Guided - use entire disk and set up LVM
         Guided - use entire disk and set up encrypted LVM
         Manual

    <Go Back>
```

Figure 4-36 Selecting a partitioning method

2. Initialize the partition table of the device.

3. Create an empty partition table for each of the displayed disk devices:
   - SCSI1 (0,0,0) (sda)
   - SCSI2 (0,0,0) (sdb)

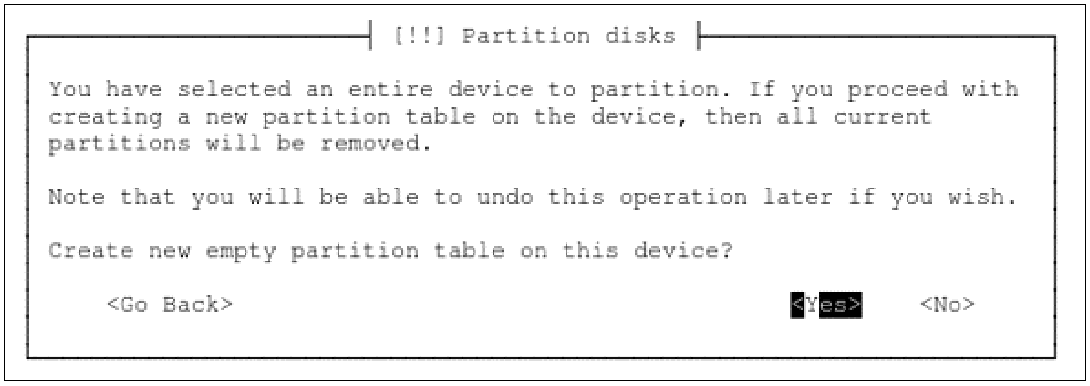
   Figure 4-37 shows the initialized partition table for a device.



```
[!!] Partition disks

This is an overview of your currently configured partitions and mount
points. Select a partition to modify its settings (file system, mount
point, etc.), a free space to create partitions, or a device to
initialize its partition table.

            Guided partitioning
            Configure software RAID
            Configure the Logical Volume Manager
            Configure encrypted volumes
            Configure iSCSI volumes

            SCSI1 (0,0,0) (sda) - 1.0 TB ATA ST1000NX0313
            >          1.0 TB      FREE SPACE
            SCSI2 (0,0,0) (sdb) - 1.0 TB ATA ST1000NX0313


    <Go Back>
```

Figure 4-37 Initializing a partition table for a device

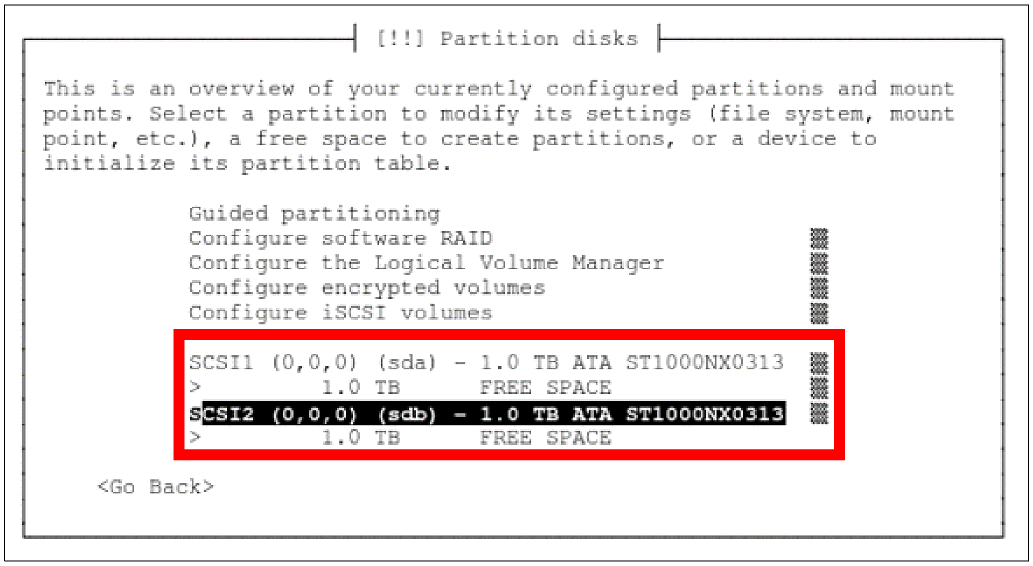
Figure 4-38 confirms the partitioning of the device.



```
[!!] Partition disks

You have selected an entire device to partition. If you proceed with
creating a new partition table on the device, then all current
partitions will be removed.

Note that you will be able to undo this operation later if you wish.

Create new empty partition table on this device?

    <Go Back>                                          <Yes>    <No>
```

Figure 4-38 Confirming the partitioning of a device

4. Check that the entire capacity of each disk (1 TB) is created as `FREE SPACE`, as shown in Figure 4-39 on page 79.



```
┌─────────────────────┤ [!!] Partition disks ├─────────────────────┐
│                                                                    │
│ This is an overview of your currently configured partitions and mount
│ points. Select a partition to modify its settings (file system, mount
│ point, etc.), a free space to create partitions, or a device to    │
│ initialize its partition table.                                    │
│                                                                    │
│            Guided partitioning                                     │
│            Configure software RAID                          ▒      │
│            Configure the Logical Volume Manager             ▒      │
│            Configure encrypted volumes                      ▒      │
│            Configure iSCSI volumes                          ▒      │
│                                                                    │
│            SCSI1 (0,0,0) (sda) - 1.0 TB ATA ST1000NX0313    ▒      │
│            >          1.0 TB      FREE SPACE                ▒      │
│            SCSI2 (0,0,0) (sdb) - 1.0 TB ATA ST1000NX0313    ▒      │
│            >          1.0 TB      FREE SPACE                       │
│                                                                    │
│     <Go Back>                                                      │
│                                                                    │
└────────────────────────────────────────────────────────────────────┘
```

Figure 4-39 Checking that the disk is properly initialized

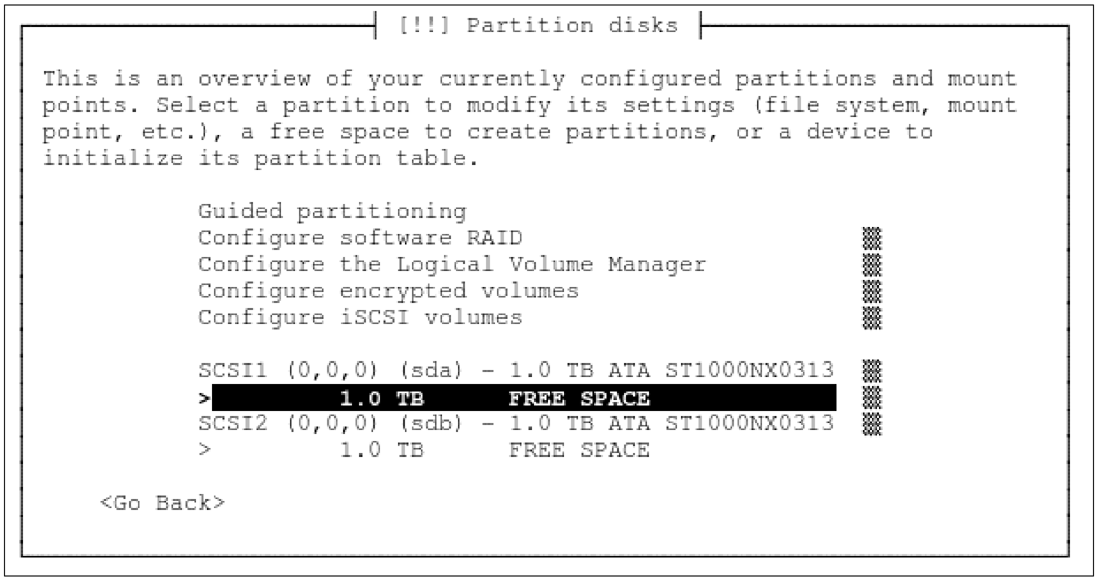
Creating a PReP boot partition

This section describes how to create a PReP boot partition.

Warning: First, create a free area of 8 MB on sda because it is not possible to create the PReP boot device on a software RAID device.

Complete the following steps:

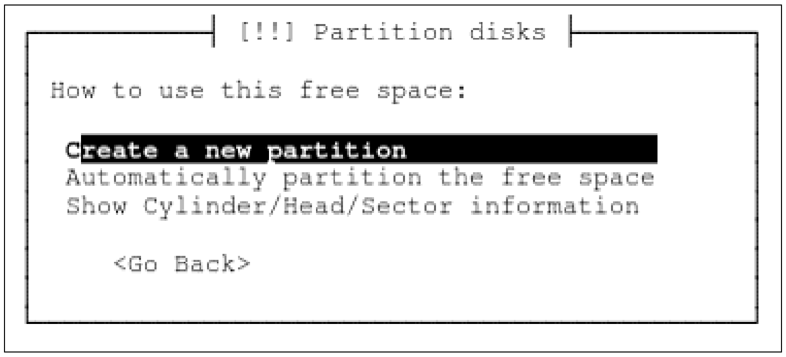
1. Select sda's `FREE SPACE` partition, as shown in Figure 4-40.



```
┌─────────────────────┤ [!!] Partition disks ├─────────────────────┐
│                                                                    │
│ This is an overview of your currently configured partitions and mount
│ points. Select a partition to modify its settings (file system, mount
│ point, etc.), a free space to create partitions, or a device to    │
│ initialize its partition table.                                    │
│                                                                    │
│            Guided partitioning                                     │
│            Configure software RAID                          ▒      │
│            Configure the Logical Volume Manager             ▒      │
│            Configure encrypted volumes                      ▒      │
│            Configure iSCSI volumes                          ▒      │
│                                                                    │
│            SCSI1 (0,0,0) (sda) - 1.0 TB ATA ST1000NX0313    ▒      │
│            >          1.0 TB      FREE SPACE                ▒      │
│            SCSI2 (0,0,0) (sdb) - 1.0 TB ATA ST1000NX0313    ▒      │
│            >          1.0 TB      FREE SPACE                       │
│                                                                    │
│     <Go Back>                                                      │
│                                                                    │
└────────────────────────────────────────────────────────────────────┘
```

Figure 4-40 Selecting the sda FREE SPACE

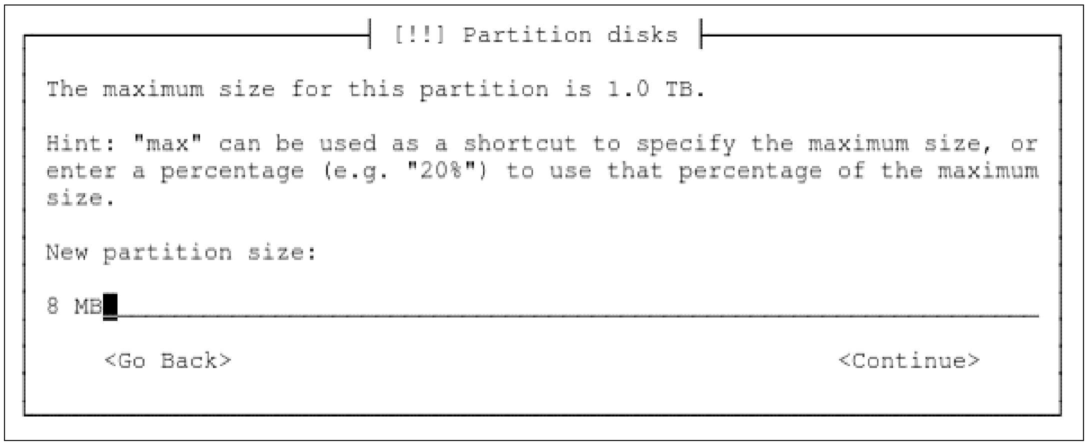
2. Click **Create a new partition**, as shown in Figure 4-41.



```
[!!] Partition disks

How to use this free space:

  Create a new partition
  Automatically partition the free space
  Show Cylinder/Head/Sector information

     <Go Back>
```

Figure 4-41 Create a new partition

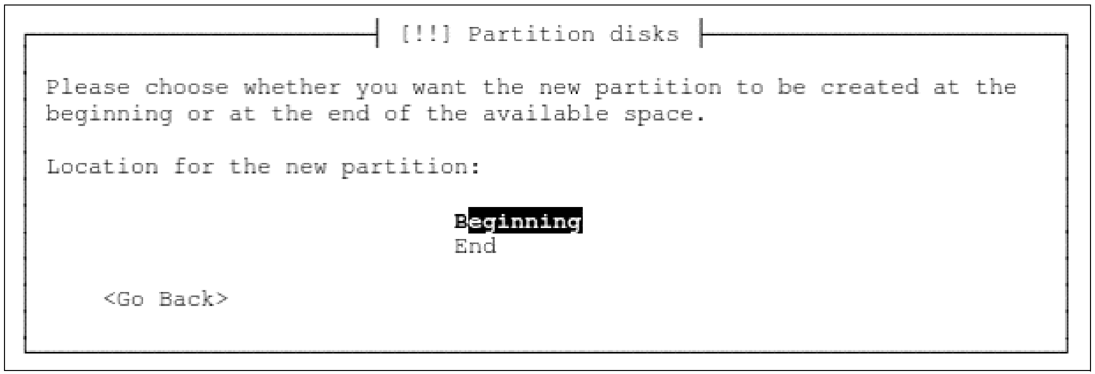
3. Specify `8 MB` as the size of the new partition, as shown in Figure 4-42.



```
[!!] Partition disks

The maximum size for this partition is 1.0 TB.

Hint: "max" can be used as a shortcut to specify the maximum size, or
enter a percentage (e.g. "20%") to use that percentage of the maximum
size.

New partition size:

8 MB

     <Go Back>                                          <Continue>
```

Figure 4-42 Specifying the PReP partition size

4. If the position of the new partition is requested, select `Beginning`, as shown in Figure 4-43.

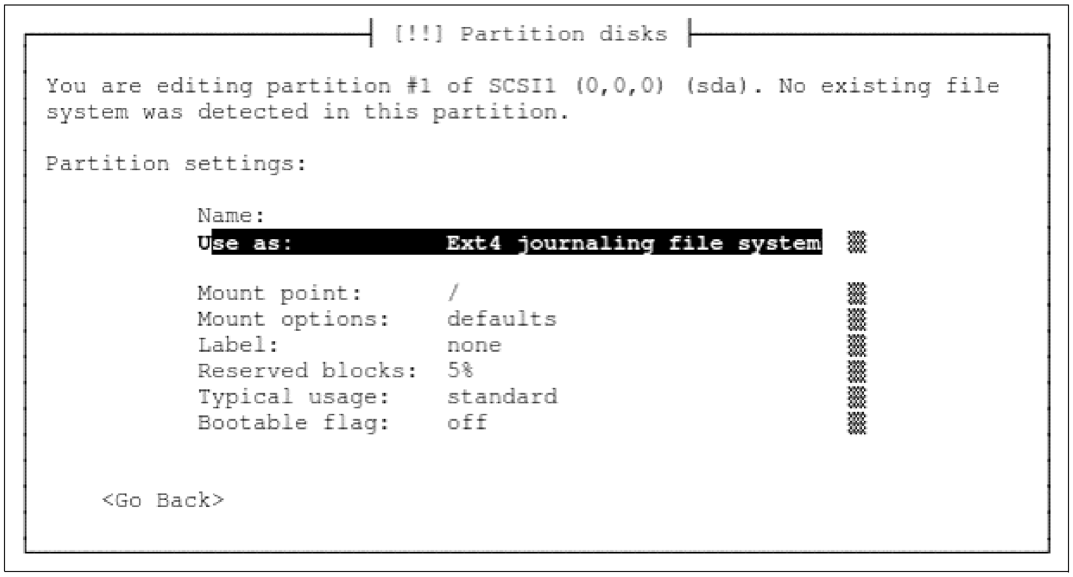


```
[!!] Partition disks

Please choose whether you want the new partition to be created at the
beginning or at the end of the available space.

Location for the new partition:

                           Beginning
                           End

     <Go Back>
```

Figure 4-43 Position of the new partition

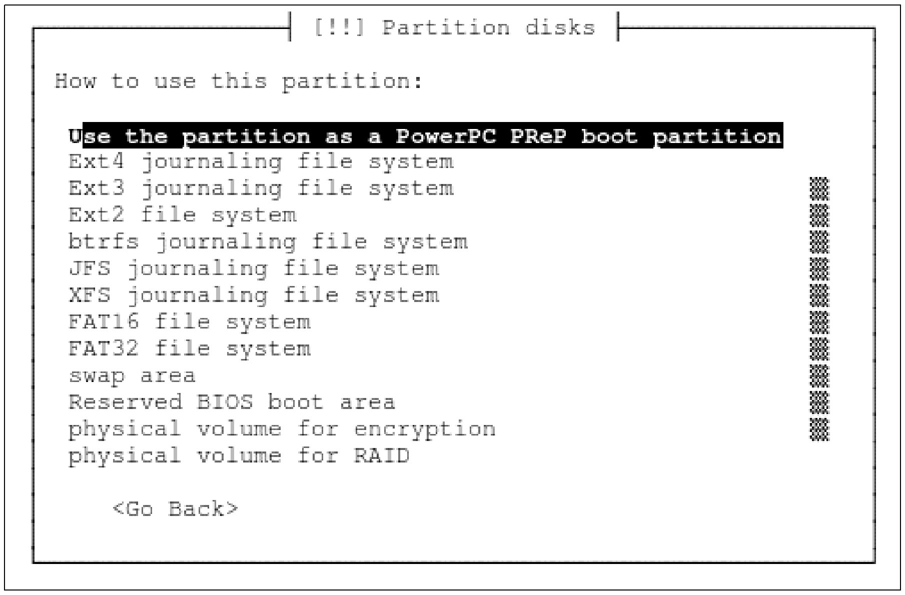
5. Define the partition file system, as shown in Figure 4-44 on page 81. Then, set the new partition use as an IBM PowerPC® PReP boot Partition,[7] as shown in Figure 4-45 on page 81.

[7] PReP is a raw and small (8 MB maximum) partition that carries only stage1 binary files.



```
[!!] Partition disks

You are editing partition #1 of SCSI1 (0,0,0) (sda). No existing file
system was detected in this partition.

Partition settings:

          Name:
          Use as:            Ext4 journaling file system

          Mount point:       /
          Mount options:     defaults
          Label:             none
          Reserved blocks:   5%
          Typical usage:     standard
          Bootable flag:     off


    <Go Back>
```

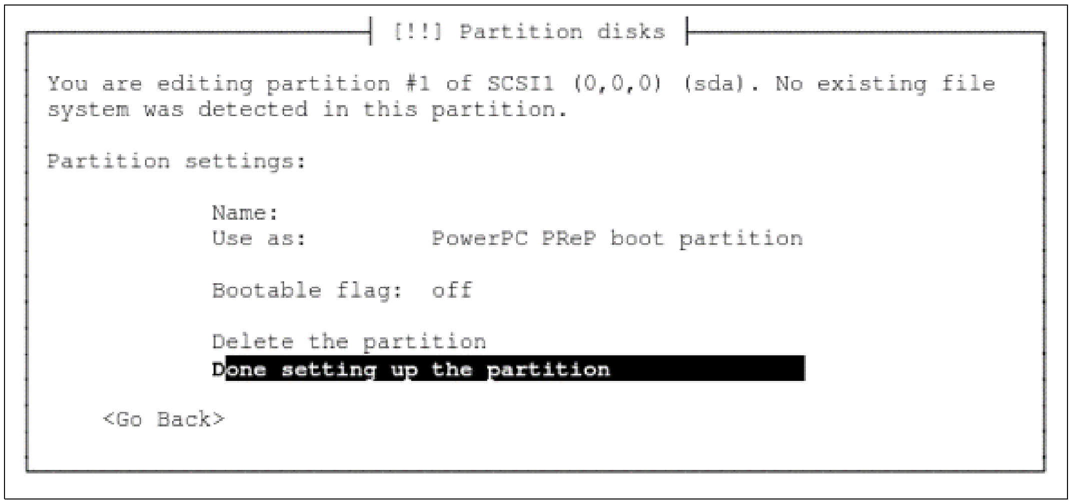
Figure 4-44 Defining the partition use



```
[!!] Partition disks

How to use this partition:

 Use the partition as a PowerPC PReP boot partition
 Ext4 journaling file system
 Ext3 journaling file system
 Ext2 file system
 btrfs journaling file system
 JFS journaling file system
 XFS journaling file system
 FAT16 file system
 FAT32 file system
 swap area
 Reserved BIOS boot area
 physical volume for encryption
 physical volume for RAID

    <Go Back>
```

Figure 4-45 Defining the partition as a PowerPC PReP boot partition

6. Click `Done setting up the partition`, as shown in Figure 4-46.



```
[!!] Partition disks

You are editing partition #1 of SCSI1 (0,0,0) (sda). No existing file
system was detected in this partition.

Partition settings:

          Name:
          Use as:          PowerPC PReP boot partition

          Bootable flag:   off

          Delete the partition
          Done setting up the partition

    <Go Back>
```

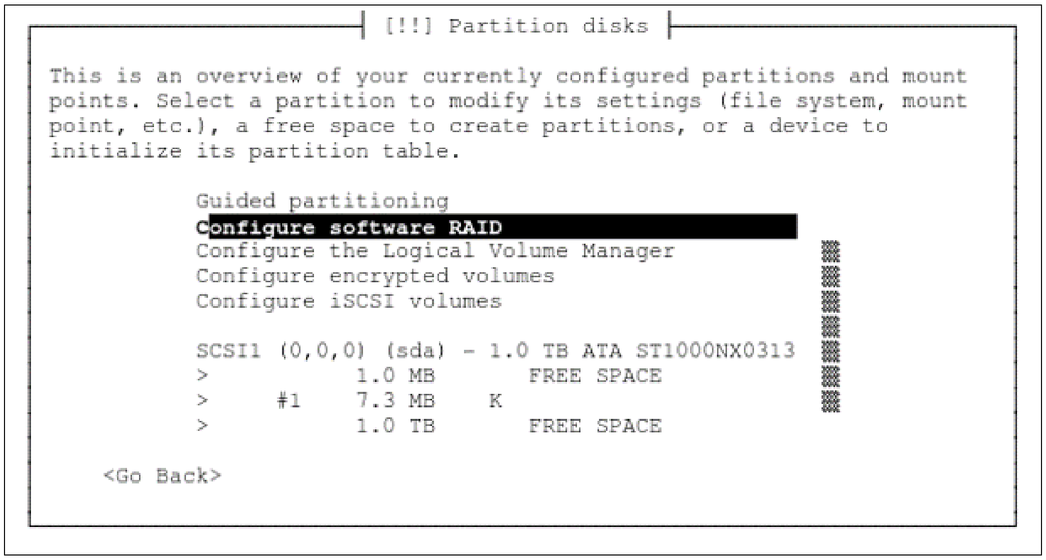
Figure 4-46 End creating boot partition

After the partition on sda is formatted, the same process must be applied for partition sdb so that both disks are prepared for the creation of the software RAID1. To achieve this task, perform steps 1 on page 79 - 6 on partition sdb, substituting sdb where necessary.

Creating a RAID1 array

This section shows how to create a RAID1 array configuration:

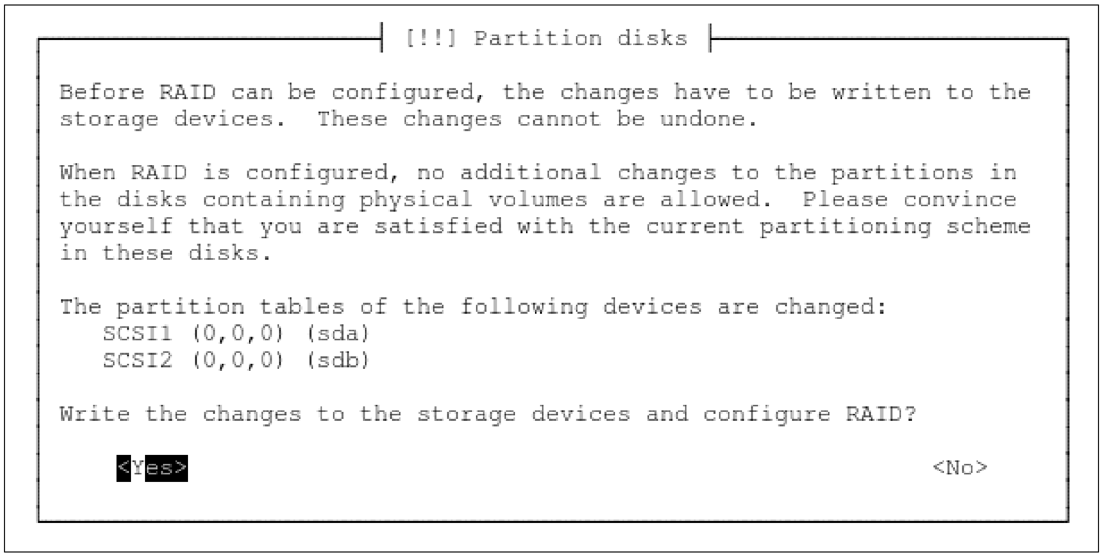
1. Select `Configure software RAID`, as shown in Figure 4-47.



```
[!!] Partition disks

This is an overview of your currently configured partitions and mount
points. Select a partition to modify its settings (file system, mount
point, etc.), a free space to create partitions, or a device to
initialize its partition table.

          Guided partitioning
          Configure software RAID
          Configure the Logical Volume Manager
          Configure encrypted volumes
          Configure iSCSI volumes

          SCSI1 (0,0,0) (sda) - 1.0 TB ATA ST1000NX0313
          >           1.0 MB       FREE SPACE
          >     #1    7.3 MB    K
          >           1.0 TB       FREE SPACE

    <Go Back>
```

Figure 4-47 Configure software RAID

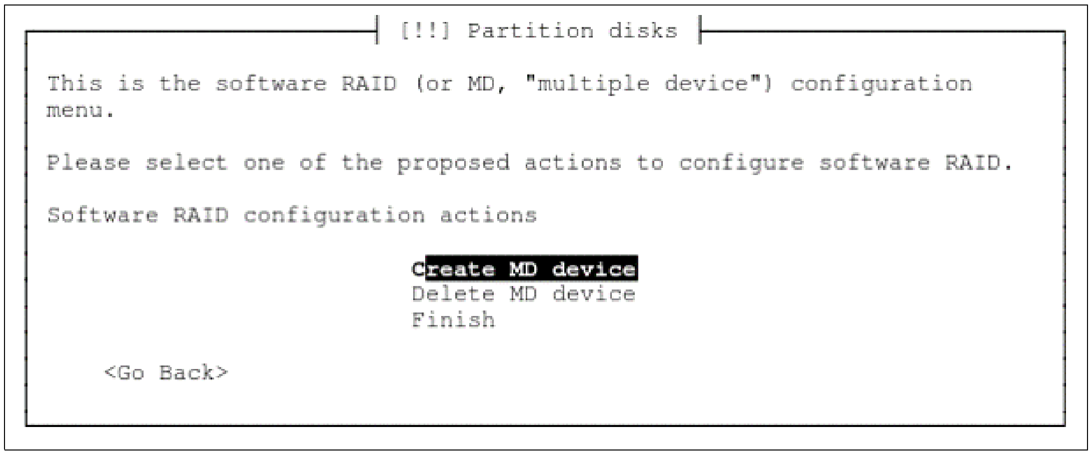
2. Confirm that you are using sda and sdb and click `Yes`, as shown in Figure 4-48 on page 83.



```
[!!] Partition disks

Before RAID can be configured, the changes have to be written to the
storage devices.  These changes cannot be undone.

When RAID is configured, no additional changes to the partitions in
the disks containing physical volumes are allowed.  Please convince
yourself that you are satisfied with the current partitioning scheme
in these disks.

The partition tables of the following devices are changed:
   SCSI1 (0,0,0) (sda)
   SCSI2 (0,0,0) (sdb)

Write the changes to the storage devices and configure RAID?

     <Yes>                                                       <No>
```

Figure 4-48   Confirming the devices selection

3. Select `Create MD device`,[8] as shown in Figure 4-49.

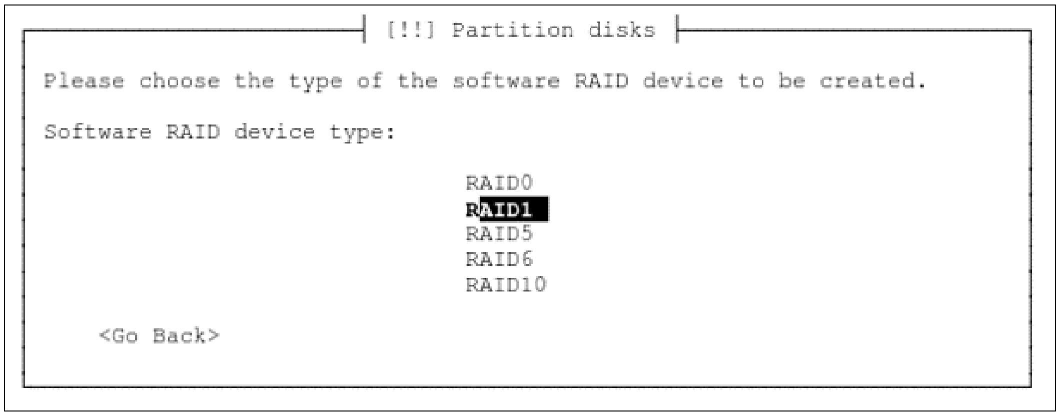


```
[!!] Partition disks

This is the software RAID (or MD, "multiple device") configuration
menu.

Please select one of the proposed actions to configure software RAID.

Software RAID configuration actions

                          Create MD device
                          Delete MD device
                          Finish

    <Go Back>
```

Figure 4-49   Create MD device

[8] The name is derived from the multiple device (MD) device nodes that are part of the RAID.

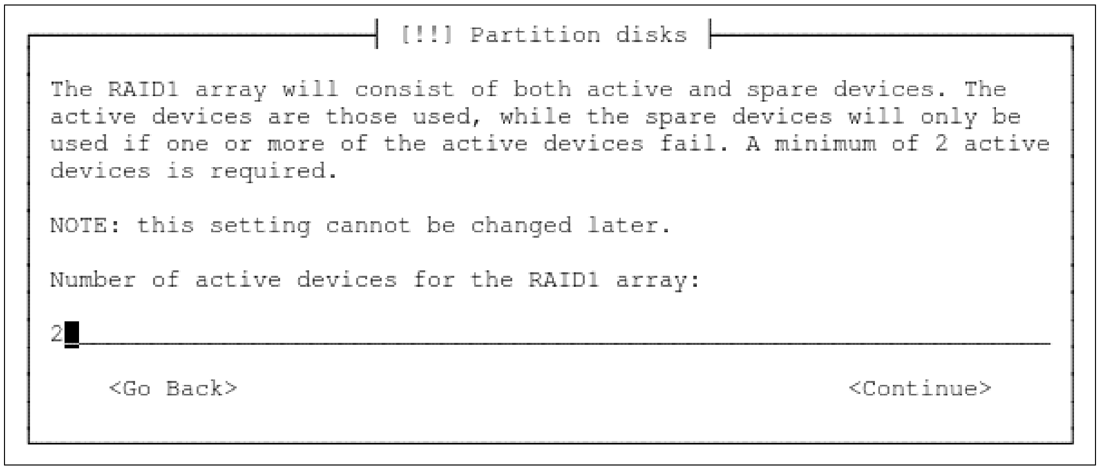
4. Select `RAID1` as the RAID array level to be used, as shown in Figure 4-50.



```
[!!] Partition disks

Please choose the type of the software RAID device to be created.

Software RAID device type:

                          RAID0
                          RAID1
                          RAID5
                          RAID6
                          RAID10

    <Go Back>
```

Figure 4-50 Selecting RAID1

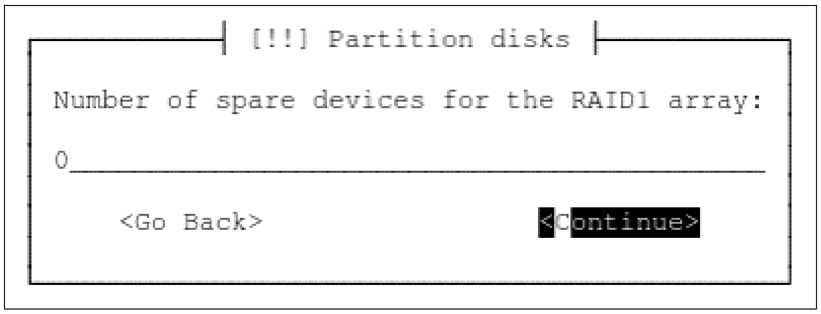
5. Specify (enter) `2` for the number of copies of the mirror to be used, as shown in Figure 4-51.



```
[!!] Partition disks

The RAID1 array will consist of both active and spare devices. The
active devices are those used, while the spare devices will only be
used if one or more of the active devices fail. A minimum of 2 active
devices is required.

NOTE: this setting cannot be changed later.

Number of active devices for the RAID1 array:

2____________________________________________________________________

    <Go Back>                                          <Continue>
```

Figure 4-51 Defining the number of copies of the mirror

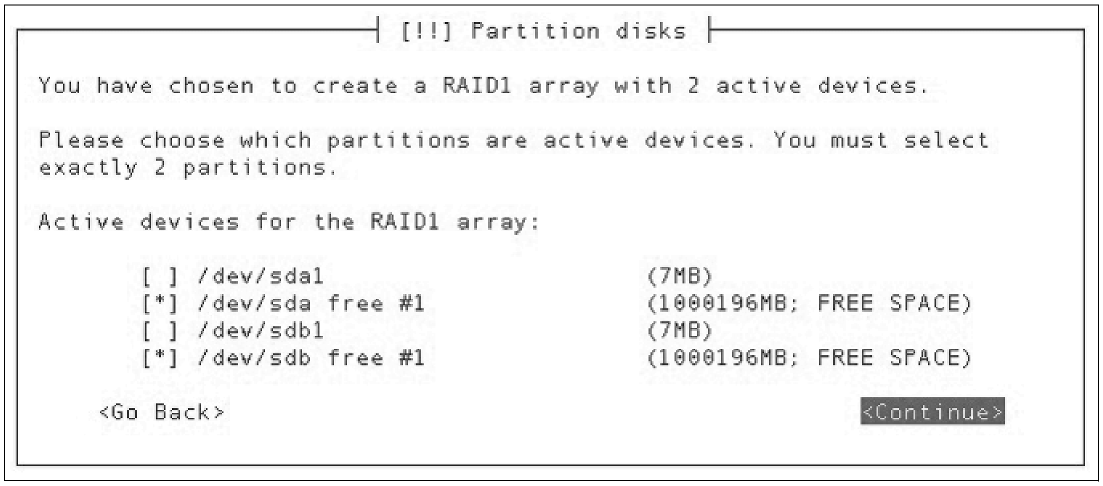
6. Specify (enter) `0` because there is no hot spare drive, as shown in Figure 4-52.



```
[!!] Partition disks

Number of spare devices for the RAID1 array:

0_____________________________________________

    <Go Back>                   <Continue>
```

Figure 4-52 Setting the number of spare devices to none

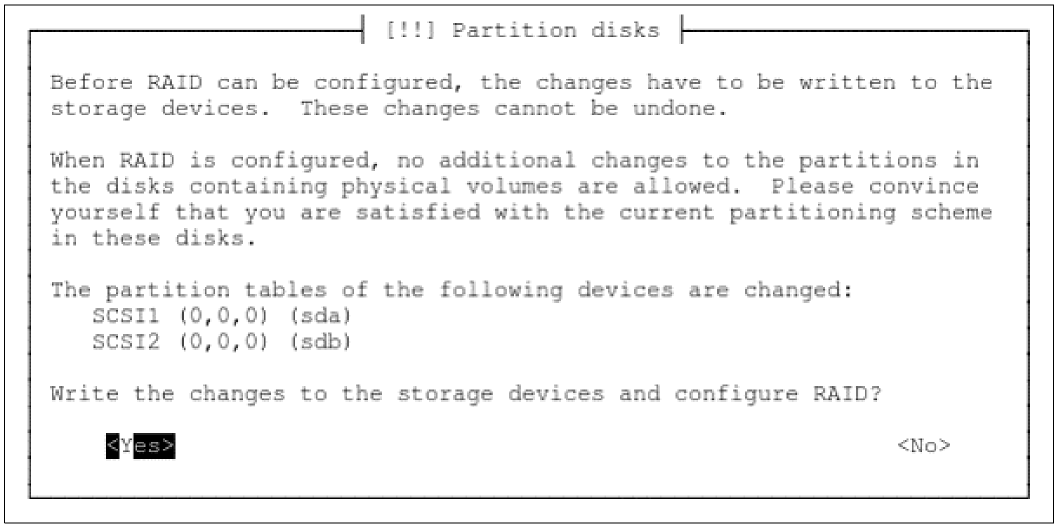
7. Select all `FREE SPACE` from `/dev/sda` and `/dev/sdb` and click `Continue`, as shown in Figure 4-53 on page 85.



```
[!!] Partition disks

You have chosen to create a RAID1 array with 2 active devices.

Please choose which partitions are active devices. You must select
exactly 2 partitions.

Active devices for the RAID1 array:

       [ ] /dev/sda1                  (7MB)
       [*] /dev/sda free #1           (1000196MB; FREE SPACE)
       [ ] /dev/sdb1                  (7MB)
       [*] /dev/sdb free #1           (1000196MB; FREE SPACE)

    <Go Back>                                         <Continue>
```

Figure 4-53 Selecting the free space to use for the RAID array

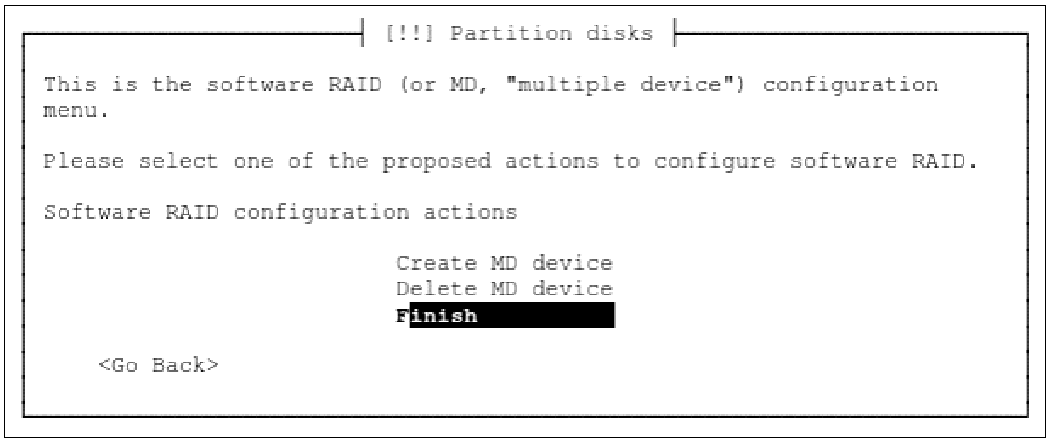
8. Apply the changes, as shown in Figure 4-54.



```
[!!] Partition disks

Before RAID can be configured, the changes have to be written to the
storage devices.  These changes cannot be undone.

When RAID is configured, no additional changes to the partitions in
the disks containing physical volumes are allowed.  Please convince
yourself that you are satisfied with the current partitioning scheme
in these disks.

The partition tables of the following devices are changed:
   SCSI1 (0,0,0) (sda)
   SCSI2 (0,0,0) (sdb)

Write the changes to the storage devices and configure RAID?

    <Yes>                                               <No>
```

Figure 4-54 Confirming the selection

9. Click `Finish` to complete the software RAID configuration, as shown in Figure 4-55.



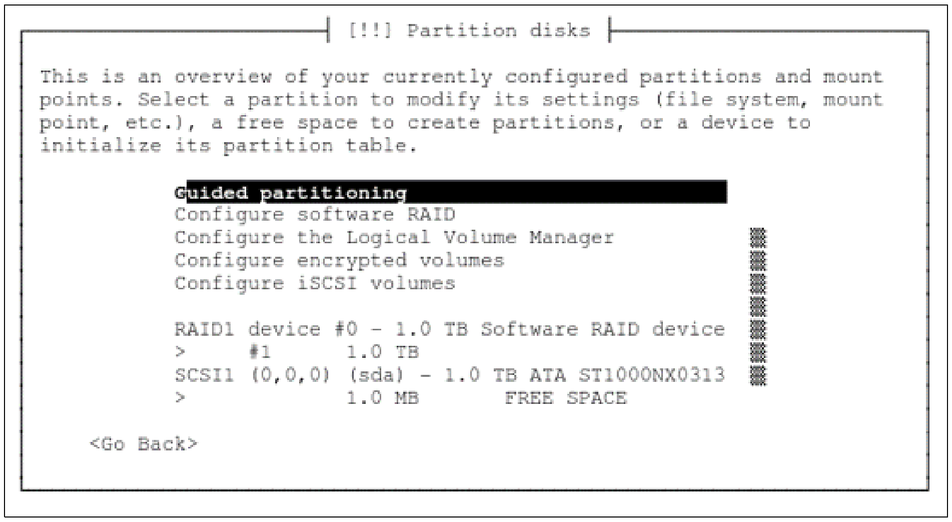
```
[!!] Partition disks

This is the software RAID (or MD, "multiple device") configuration
menu.

Please select one of the proposed actions to configure software RAID.

Software RAID configuration actions

                     Create MD device
                     Delete MD device
                     Finish

    <Go Back>
```

Figure 4-55 Confirming the creation of the multiple devices

Partitioning the RAID array for the operating system installation

This section describes how to partition the RAID array for the operating system installation:

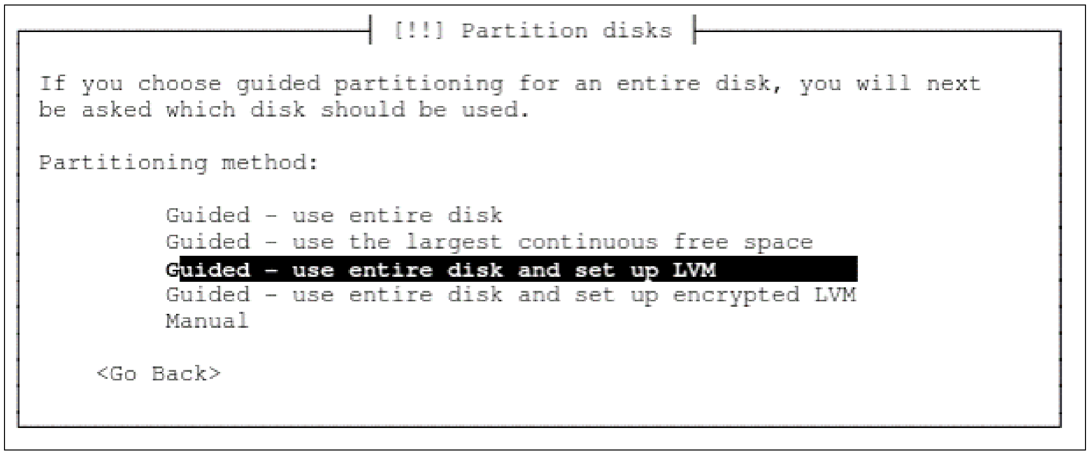
1. To configure LVM with the created MD and create partitions on it, select `Guided partitioning`, as shown in Figure 4-56.



```
[!!] Partition disks

This is an overview of your currently configured partitions and mount
points. Select a partition to modify its settings (file system, mount
point, etc.), a free space to create partitions, or a device to
initialize its partition table.

           Guided partitioning
           Configure software RAID
           Configure the Logical Volume Manager
           Configure encrypted volumes
           Configure iSCSI volumes

           RAID1 device #0 - 1.0 TB Software RAID device
           >     #1      1.0 TB
           SCSI1 (0,0,0) (sda) - 1.0 TB ATA ST1000NX0313
           >             1.0 MB       FREE SPACE

    <Go Back>
```

Figure 4-56 Selecting Guided partitioning

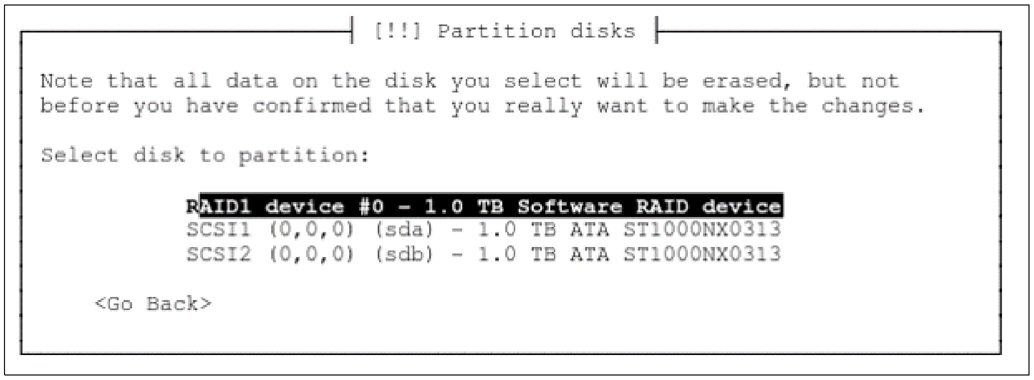
2. Select `Guided - use entire disk and set up LVM`, as shown in Figure 4-57 on page 87.



```
[!!] Partition disks

If you choose guided partitioning for an entire disk, you will next
be asked which disk should be used.

Partitioning method:

         Guided - use entire disk
         Guided - use the largest continuous free space
         Guided - use entire disk and set up LVM
         Guided - use entire disk and set up encrypted LVM
         Manual

    <Go Back>
```

Figure 4-57 Selecting the entire disk to set up LVM

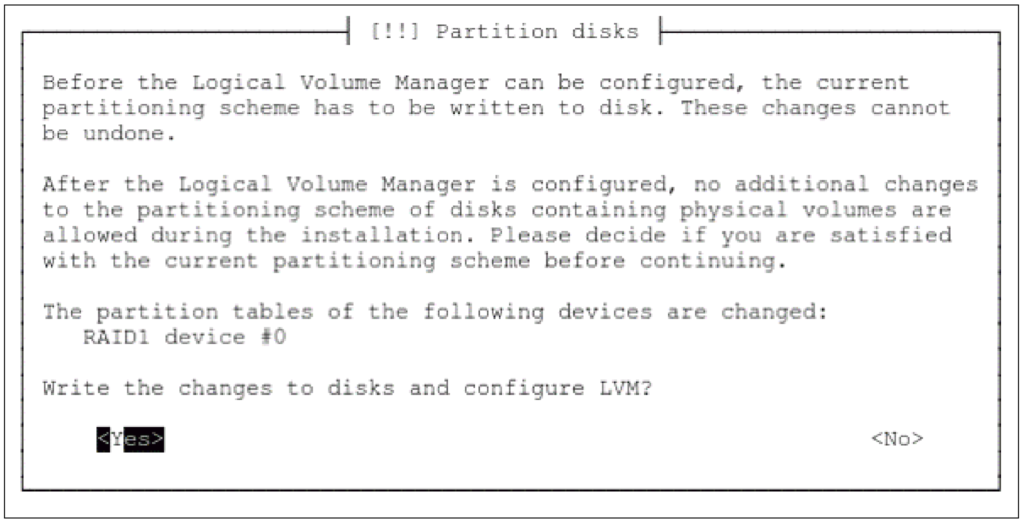
3. Select `RAID1 device # 0` as the device to be used, as shown in Figure 4-58.



```
[!!] Partition disks

Note that all data on the disk you select will be erased, but not
before you have confirmed that you really want to make the changes.

Select disk to partition:

          RAID1 device #0 - 1.0 TB Software RAID device
          SCSI1 (0,0,0) (sda) - 1.0 TB ATA ST1000NX0313
          SCSI2 (0,0,0) (sdb) - 1.0 TB ATA ST1000NX0313

    <Go Back>
```

Figure 4-58 Selecting the device

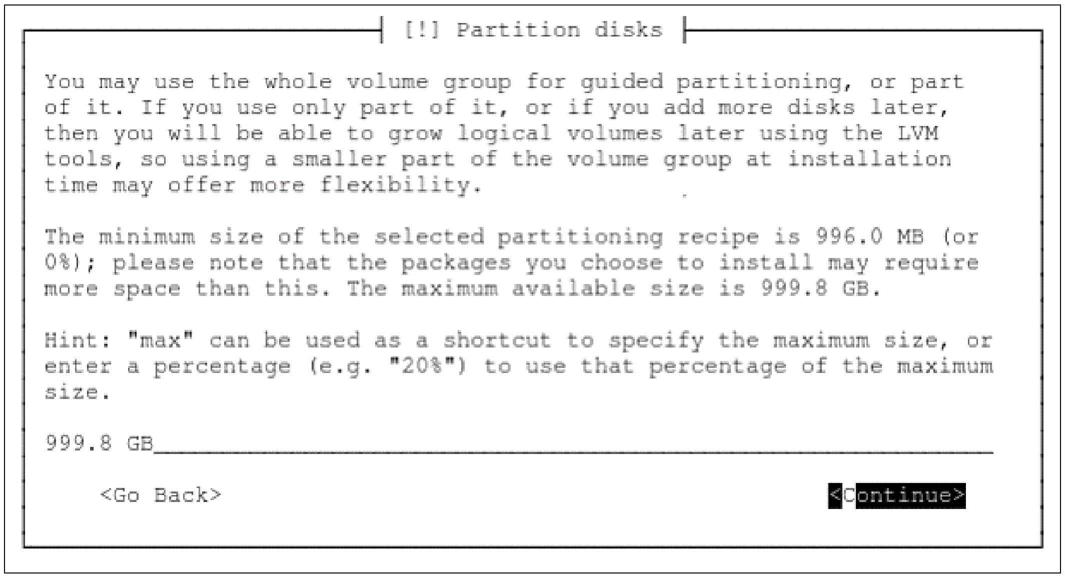
4. A confirmation window for the change opens. Select `Yes`, as shown in Figure 4-59.



```
[!!] Partition disks

Before the Logical Volume Manager can be configured, the current
partitioning scheme has to be written to disk. These changes cannot
be undone.

After the Logical Volume Manager is configured, no additional changes
to the partitioning scheme of disks containing physical volumes are
allowed during the installation. Please decide if you are satisfied
with the current partitioning scheme before continuing.

The partition tables of the following devices are changed:
   RAID1 device #0

Write the changes to disks and configure LVM?

    <Yes>                                                    <No>
```

Figure 4-59 Confirming your selection

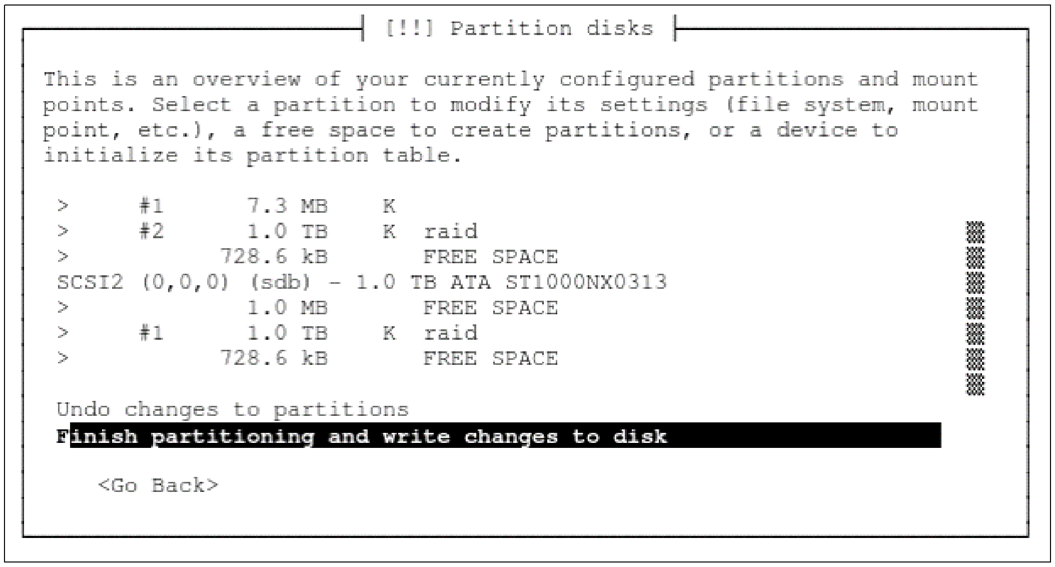
5. Specify the size of the volume group to be configured and select `Continue`, as shown in Figure 4-60.



```
[!] Partition disks

You may use the whole volume group for guided partitioning, or part
of it. If you use only part of it, or if you add more disks later,
then you will be able to grow logical volumes later using the LVM
tools, so using a smaller part of the volume group at installation
time may offer more flexibility.

The minimum size of the selected partitioning recipe is 996.0 MB (or
0%); please note that the packages you choose to install may require
more space than this. The maximum available size is 999.8 GB.

Hint: "max" can be used as a shortcut to specify the maximum size, or
enter a percentage (e.g. "20%") to use that percentage of the maximum
size.

999.8 GB______________________________________________________________

    <Go Back>                                              <Continue>
```

Figure 4-60 Setting the size of the volume

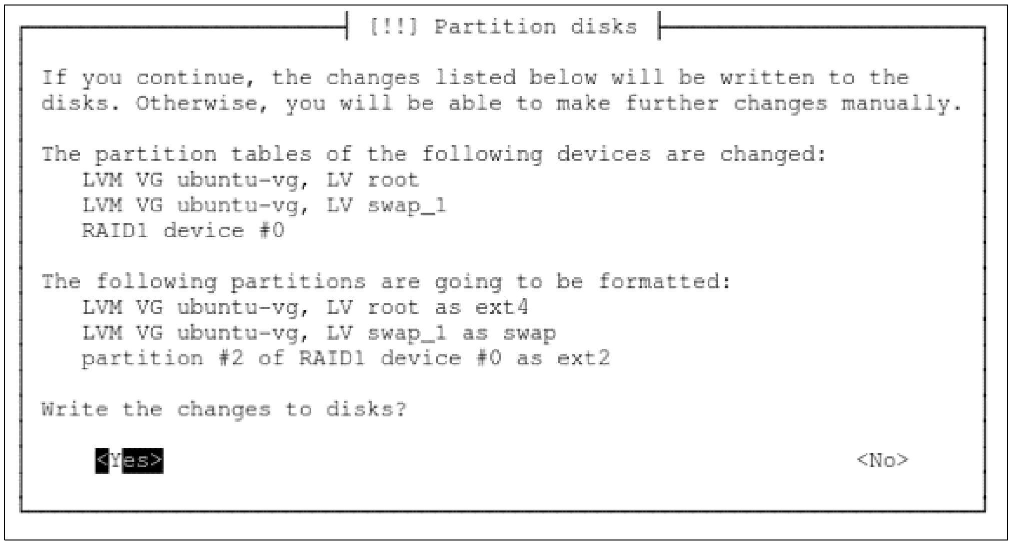
6. Select `Finish partitioning and write changes to disk`, as shown in Figure 4-61.



```
[!!] Partition disks

This is an overview of your currently configured partitions and mount
points. Select a partition to modify its settings (file system, mount
point, etc.), a free space to create partitions, or a device to
initialize its partition table.

 >     #1      7.3 MB    K
 >     #2      1.0 TB    K  raid
 >           728.6 kB       FREE SPACE
 SCSI2 (0,0,0) (sdb) - 1.0 TB ATA ST1000NX0313
 >             1.0 MB       FREE SPACE
 >     #1      1.0 TB    K  raid
 >           728.6 kB       FREE SPACE

 Undo changes to partitions
 Finish partitioning and write changes to disk

    <Go Back>
```

Figure 4-61 Save and finish

7. Check the LV or partition to be created and select `Yes`, as shown in Figure 4-62 on page 89.



```
[!!] Partition disks

If you continue, the changes listed below will be written to the
disks. Otherwise, you will be able to make further changes manually.

The partition tables of the following devices are changed:
   LVM VG ubuntu-vg, LV root
   LVM VG ubuntu-vg, LV swap_1
   RAID1 device #0

The following partitions are going to be formatted:
   LVM VG ubuntu-vg, LV root as ext4
   LVM VG ubuntu-vg, LV swap_1 as swap
   partition #2 of RAID1 device #0 as ext2

Write the changes to disks?

     <Yes>                                                   <No>
```

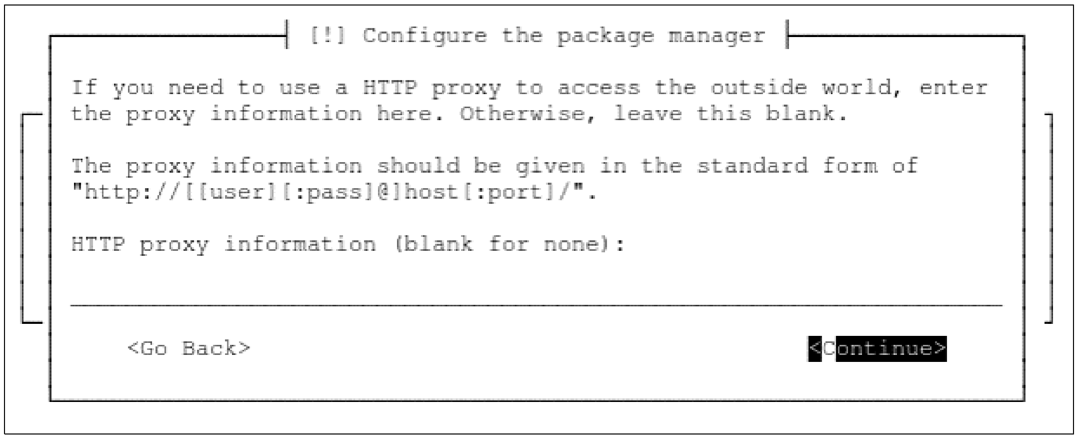
Figure 4-62 Applying the changes

Installing operating system packages

After the partitions are created, the installation of all operating system packages starts. For this guide, we assume that the system can access the internet or a repository with all of the operating system packages through an HTTP Proxy.

Complete the following steps:

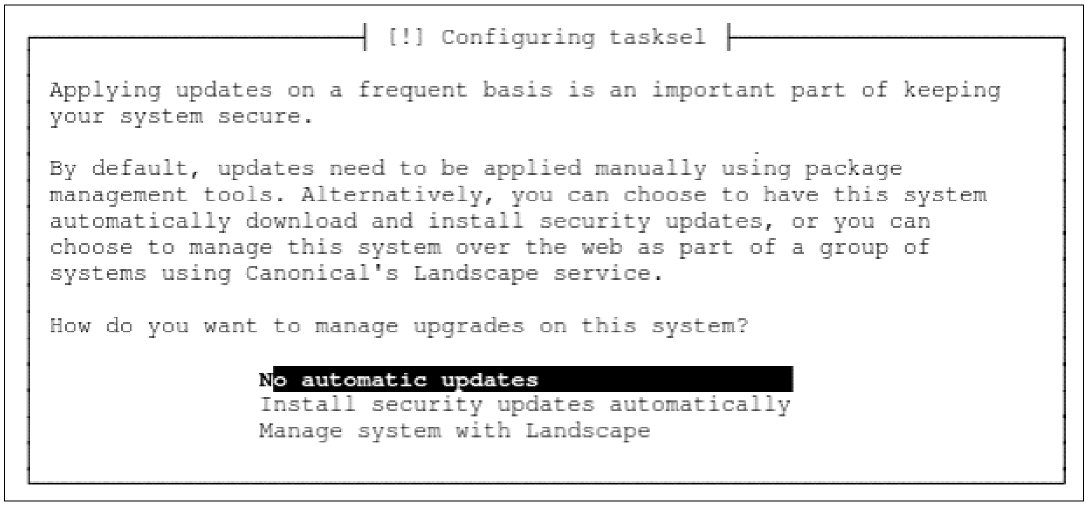
1. Set up the HTTP proxy, as shown in Figure 4-63.



```
[!] Configure the package manager

If you need to use a HTTP proxy to access the outside world, enter
the proxy information here. Otherwise, leave this blank.

The proxy information should be given in the standard form of
"http://[[user][:pass]@]host[:port]/".

HTTP proxy information (blank for none):

______________________________________________________________________

    <Go Back>                                           <Continue>
```

Figure 4-63 Setting up the HTTP proxy

2. Configure automatic updates, as shown in Figure 4-64.



```
[!] Configuring tasksel

Applying updates on a frequent basis is an important part of keeping
your system secure.

By default, updates need to be applied manually using package
management tools. Alternatively, you can choose to have this system
automatically download and install security updates, or you can
choose to manage this system over the web as part of a group of
systems using Canonical's Landscape service.

How do you want to manage upgrades on this system?

               No automatic updates
               Install security updates automatically
               Manage system with Landscape
```

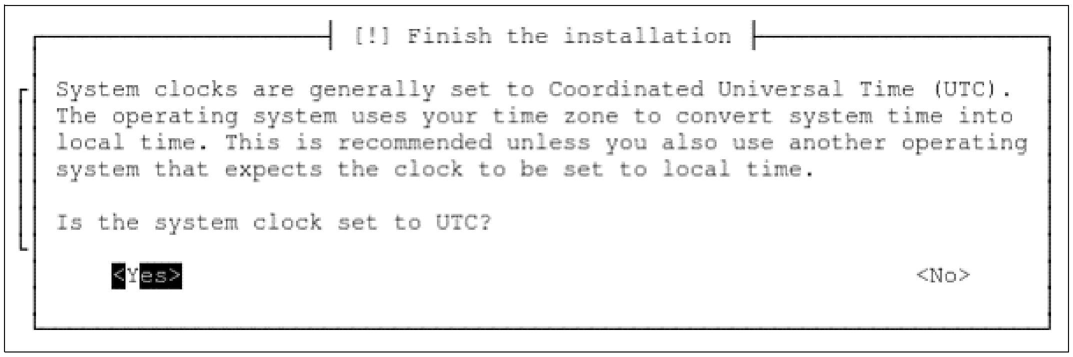
Figure 4-64 Setting up automatic updates

Note: To avoid conflicts with untested or unsupported versions of certain system libraries, disable automatic updates.

It is not mandatory to select to install security updates automatically. You, as the sysadmin, can determine whether your system is exposed to threats, and whether you need Linux updates to be deployed automatically.

Configuring system time zone

This section describes how to configure the system time zone. Choose a configuration that fits your needs, as shown in Figure 4-65.

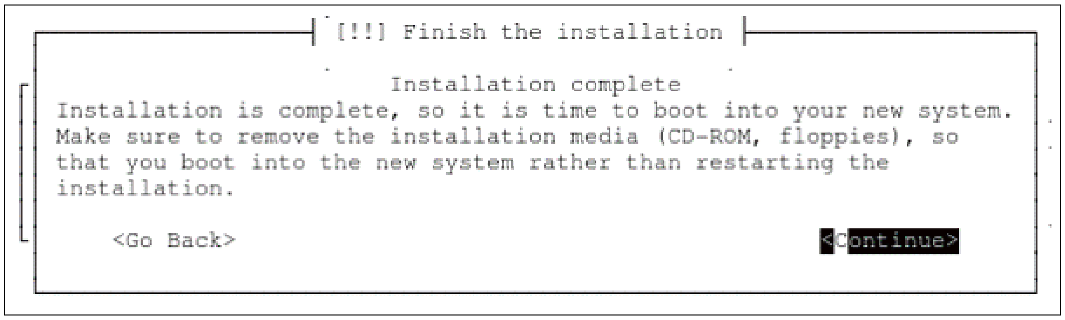


```
[!] Finish the installation

System clocks are generally set to Coordinated Universal Time (UTC).
The operating system uses your time zone to convert system time into
local time. This is recommended unless you also use another operating
system that expects the clock to be set to local time.

Is the system clock set to UTC?

    <Yes>                                                    <No>
```

Figure 4-65 Setting up a system clock

Finishing the operating system installation

This section shows how to complete the operating system installation. Choose `Continue` to finish the installation, as shown in Figure 4-66 on page 91.

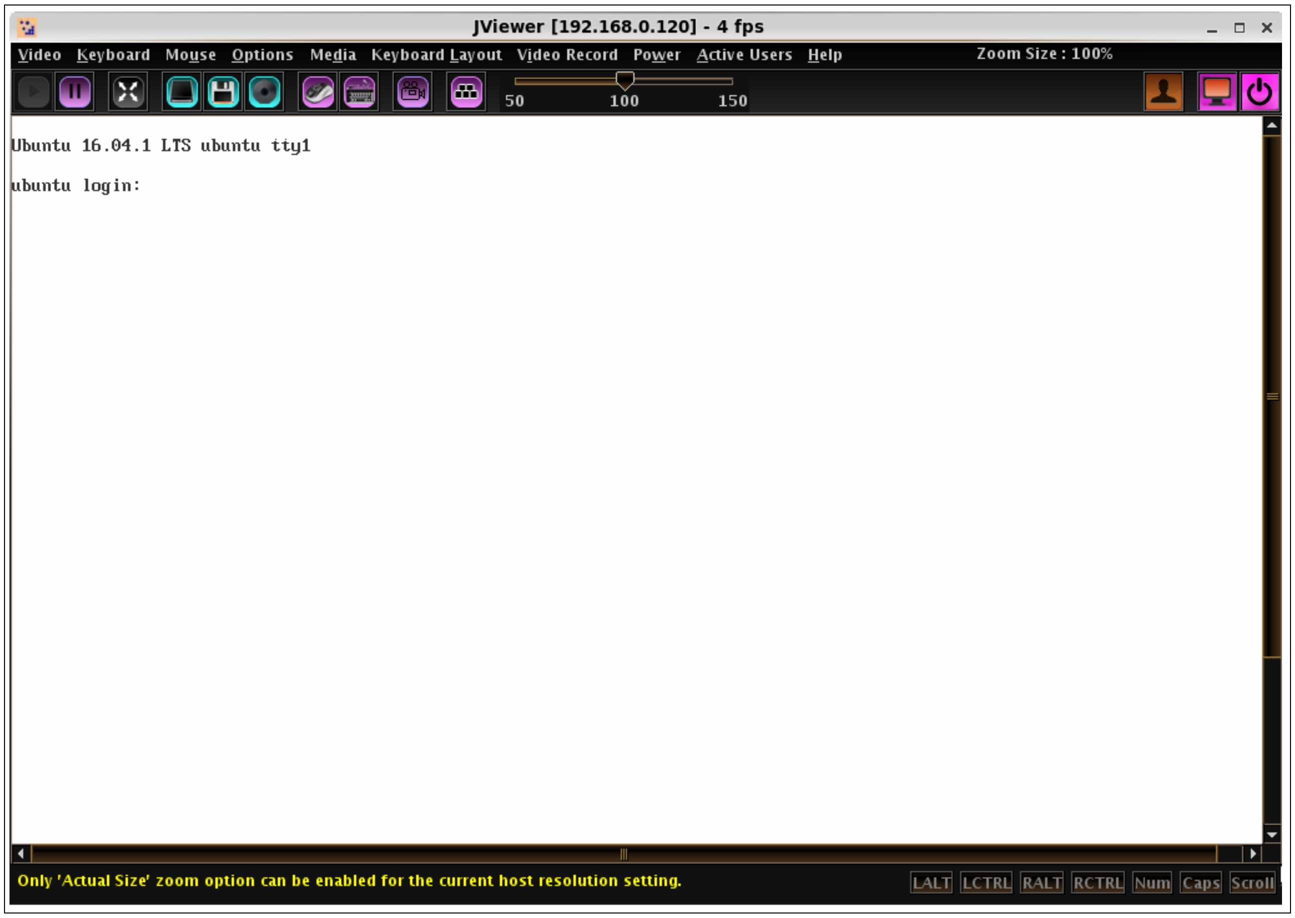


```
[!!] Finish the installation

                    Installation complete
Installation is complete, so it is time to boot into your new system.
Make sure to remove the installation media (CD-ROM, floppies), so
that you boot into the new system rather than restarting the
installation.

    <Go Back>                                          <Continue>
```

Figure 4-66 Installation complete

Operating system setup

After a restart, the system console is available for log in to the system, as shown in Figure 4-67.



JViewer [192.168.0.120] - 4 fps
Video Keyboard Mouse Options Media Keyboard Layout Video Record Power Active Users Help
Zoom Size : 100%
50 100 150

```
Ubuntu 16.04.1 LTS ubuntu tty1

ubuntu login:
```

Only 'Actual Size' zoom option can be enabled for the current host resolution setting.
LALT LCTRL RALT RCTRL Num Caps Scroll

Figure 4-67 Login window

Complete the following steps:

1. Log in as the user that was created at the time of installation.
2. If necessary, set up other network adapters. A sample configuration is provided as follows:

   ```
   $ sudo vi / etc / network / interfaces
   auto enP3p3s0f1
   iface enP3p3s0f1 inet static
   ```

```
    address 192.168.1.xxx
    netmask 255.255.255.0
    network 192.168.1.0
    gateway 192.168.1.1
    dns-nameservers 192.168.1.xxx

auto enP3p3s0f2
iface enP3p3s0f1 inet dhcp
```

3. Update the package list:
   - If there is an internet connection available, update from the default operating system repository by running the following command:

     ```
     $ sudo apt-get update
     ```

   - If there is no internet connection available, create a local repository by using a DVD image and update the list:
     i. Attach the DVD ISO image by using remote control virtual media.
     ii. Add the following entry to `/etc/fstab` and mount it:

        ```
        /dev/sr0/media/cdrom iso 9660 ro 0 0
        ```

     iii. Run the following command and add the DVD to the repository:

        ```
        $ sudo apt-cdrom -d /media/cdrom add 0
        ```

     iv. Update the package list:

        ```
        $ sudo apt-get update
        ```

   - Install additional packages. For example, install OpenSSH:

     ```
     $ sudo apt - get install - y openssh - server
     ```

Setting up a boot device

This section describes how to set up the boot device:

1. Specify PowerPC PReP boot type in `/dev/sdb1` as follows:

```
$ sudo fdisk /dev/sdb
   Welcome to fdisk (util-linux 2.27.1).
   Changes will remain in memory only until you decide to write them.
   Be careful before using the write command.

   Command (m for help): p

   Disk / dev / sdb: 931.5 GiB, 1000204886016 bytes, 1953525168 sectors
   Units: sectors of 1 * 512 = 512 bytes
   Sector size (logical / physical): 512 bytes / 512 bytes
   I / O size (minimum / optimal): 512 bytes / 512 bytes
   Disklabel type: gpt
   Disk identifier: FADBB 551 - 1006 - 4 D 21 - 8 BB 2 - 2 D 0 EF 16 D 893
   Device Start End Sectors Size Type
   / dev / sdb 1 2048 16383 14336 7 M Linux filesystem
   / dev / sdb 2 16384 195 35 23 7 11 195 3 50 7 32 8 931.5 G Linux RAID

   Command (m for help): t

   Partition number (1, 2, default 2): 1

   Hex code (type L to list all codes): 7
```

```
Changed type of partition 'Linux filesystem' to 'PowerPC PReP boot'.

Command (m for help): p
Disk / dev / sdb: 931.5 GiB, 1000204886016 bytes, 1953525168 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical / physical): 512 bytes / 512 bytes
I / O size (minimum / optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: FADBB 551 - 1006 - 4 D 21 - 8 BB 2 - 2 D 0 EF 16 D 893
Device Start End Sectors Size Type
/ dev / sdb 1 2048 16383 14336 7 M PowerPC PReP boot
/ dev / sdb 2 16384 195 35 23 7 11 195 3 50 7 32 8 931.5 G Linux RAID

Command (m for help): w
The partition table has been altered.
Calling ioctl () to re-read partition table.
Re-reading the partition table failed .: Device or resource busy
The new table will be used at the next reboot or after you run partprobe (8)
or kpartx (8).
```

2. In the initial installation state, if a failure occurs on the disk where the PReP boot area is created, the system cannot start properly. Therefore, copy the contents of PReP boot area of `/dev/sda1` to `/dev/sdb1`, which is created as an empty area at the time of installation.

```
$ sudo dd if=/dev/sda1 of=/dev/sdb1
   14336 + 0 records in
   14336 + 0 records out
   7340032 bytes (7.3 MB, 7.0 MiB) copied, 0.116626 s, 62.9 MB/s
```

4.1.5 Installing IBM PowerAI V1.4

IBM PowerAI provides a streamlined process to install all libraries and frameworks that are needed to deploy a DL workload. However, because of license limitations, some of the core components cannot be included in the meta-package. This section provides instructions about how to deploy IBM PowerAI V4.0 and every component that is required.

One of the co-authors of this book has created a short video to demonstrate the IBM PowerAI V1.4 installation process.

For reference, the official IBM PowerAI release notes include installation instructions for every version of the solution.

Checking the prerequisites

This section describes how to check for the prerequisites:

1. Confirm the version of the libc6 package by running the following command:

```
$ dpkg -s libc 6 | grep -i version
Version: 2.23-0ubuntu7
the system. This package includes shared version s of the standard C library
```

   If the version is 2.23-0ubuntu5 or earlier, update it.

2. Confirm whether you must install the python-socketio package. Only when you use Deep Learning GPU Training System (DIGITS) is necessary to uninstall the python-socketio package that is provided in Ubuntu to avoid package conflict.

To confirm the presence of the package, run the following command:

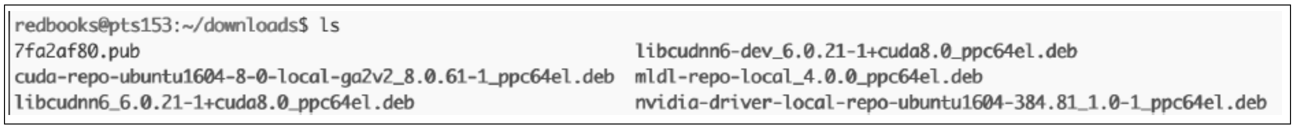
```
$ dpkg -r python-socketio
dpkg-query: package 'python-socketio' is not installed and no information is
available
```

Installing NVIDIA drivers and components

This section provides installation instructions for the NVIDIA drivers and components:

1. Download the NVIDIA required files:

   To install IBM PowerAI, you need copies of all of the following files, as shown in Figure 4-68.



```
redbooks@pts153:~/downloads$ ls
7fa2af80.pub                                          libcudnn6-dev_6.0.21-1+cuda8.0_ppc64el.deb
cuda-repo-ubuntu1604-8-0-local-ga2v2_8.0.61-1_ppc64el.deb  mldl-repo-local_4.0.0_ppc64el.deb
libcudnn6_6.0.21-1+cuda8.0_ppc64el.deb                nvidia-driver-local-repo-ubuntu1604-384.81_1.0-1_ppc64el.deb
```

Figure 4-68 IBM PowerAI and NVIDIA installation files

   – `7fa2af80.pub`: This GPG Key, which you get from accepting the NVIDIA license agreement, is required to install the NVIDIA CUDA libraries (for download instructions, see 3.1.5, “NVIDIA components” on page 42).
   – `cuda-repo-ubuntu1604-8-0-local-ga2v2_8.0.61-1_ppc64el.deb`: The local installation file for the NVIDIA CUDA API for GPU development. This file is available directly from NVIDIA and provides a collection of libraries and functions for GPU development that is needed for the IBM PowerAI frameworks (for download instructions, see “CUDA” on page 42, and “NVIDIA CUDA Toolkit” on page 43).
   – `libcudnn6*`:- NVIDIA cuDNN is a library of GPU-accelerated primitives for deep neural network development (for download instructions, see “NVIDIA cuDNN” on page 43).
   – `nvidia-driver-local-repo-ubuntu1604-384.81_1.0-1_ppc64el.deb`: The current NVIDIA GPU driver (for download instructions, see “NVIDIA drivers download” on page 44).
   – `mldl-repo-local_4.0.0_ppc64el.deb`; The IBM PowerAI installation file, which contains prebuilt versions of all your favorite artificial intelligence (AI) frameworks and their dependencies (for download instructions, see 3.1.7, “IBM PowerAI deep learning package” on page 45).

Tip: Although NVIDIA provides a network installation of the CUDA repository, use the local version. The network installer automatically updates the CUDA version, which can cause a driver mismatch.

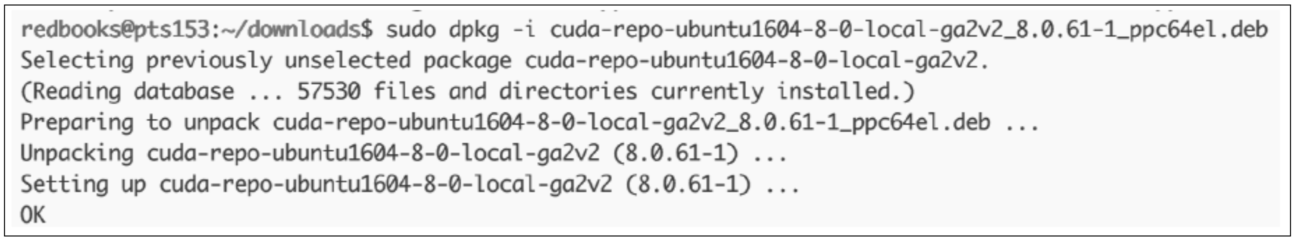
2. Install the NVIDIA device driver:

   a. Add the GPG key for the CUDA repository by running the following command:

   ```
   $ sudo apt-key add 7fa2af80.pub
   ```

   b. Use the Debian package manager (dpkg) to install CUDA first, as shown in Figure 4-69 on page 95:

   ```
   $ sudo dpkg -i cuda-repo-ubuntu1604-8-0-local-gav2_8.0.61-1_ppc64el.deb
   ```

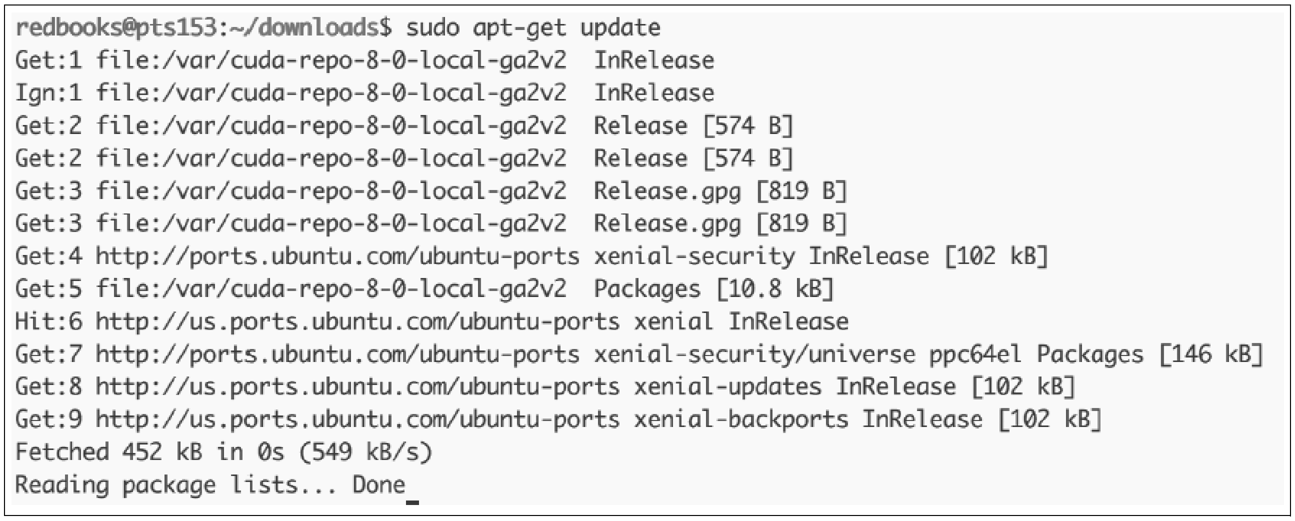


```
redbooks@pts153:~/downloads$ sudo dpkg -i cuda-repo-ubuntu1604-8-0-local-ga2v2_8.0.61-1_ppc64el.deb
Selecting previously unselected package cuda-repo-ubuntu1604-8-0-local-ga2v2.
(Reading database ... 57530 files and directories currently installed.)
Preparing to unpack cuda-repo-ubuntu1604-8-0-local-ga2v2_8.0.61-1_ppc64el.deb ...
Unpacking cuda-repo-ubuntu1604-8-0-local-ga2v2 (8.0.61-1) ...
Setting up cuda-repo-ubuntu1604-8-0-local-ga2v2 (8.0.61-1) ...
OK
```

Figure 4-69 Installing CUDA

c. Resynchronize the package index files, as shown in Figure 4-70:

```
$ sudo apt-get update
```



```
redbooks@pts153:~/downloads$ sudo apt-get update
Get:1 file:/var/cuda-repo-8-0-local-ga2v2  InRelease
Ign:1 file:/var/cuda-repo-8-0-local-ga2v2  InRelease
Get:2 file:/var/cuda-repo-8-0-local-ga2v2  Release [574 B]
Get:2 file:/var/cuda-repo-8-0-local-ga2v2  Release [574 B]
Get:3 file:/var/cuda-repo-8-0-local-ga2v2  Release.gpg [819 B]
Get:3 file:/var/cuda-repo-8-0-local-ga2v2  Release.gpg [819 B]
Get:4 http://ports.ubuntu.com/ubuntu-ports xenial-security InRelease [102 kB]
Get:5 file:/var/cuda-repo-8-0-local-ga2v2  Packages [10.8 kB]
Hit:6 http://us.ports.ubuntu.com/ubuntu-ports xenial InRelease
Get:7 http://ports.ubuntu.com/ubuntu-ports xenial-security/universe ppc64el Packages [146 kB]
Get:8 http://us.ports.ubuntu.com/ubuntu-ports xenial-updates InRelease [102 kB]
Get:9 http://us.ports.ubuntu.com/ubuntu-ports xenial-backports InRelease [102 kB]
Fetched 452 kB in 0s (549 kB/s)
Reading package lists... Done
```

Figure 4-70 The apt-get update sample output

d. Install the CUDA tools:

```
$ sudo apt-get install cuda
```

e. Export paths to the CUDA libraries:

```
$ export PATH=/usr/local/cuda-8.0/bin${PATH:+:${PATH}}
$ export LD_LIBRARY_PATH=/usr/local/cuda-8.0/\
lib64${LD_LIBRARY_PATH:+:{$LD_LIBRARY_PATH}}
```

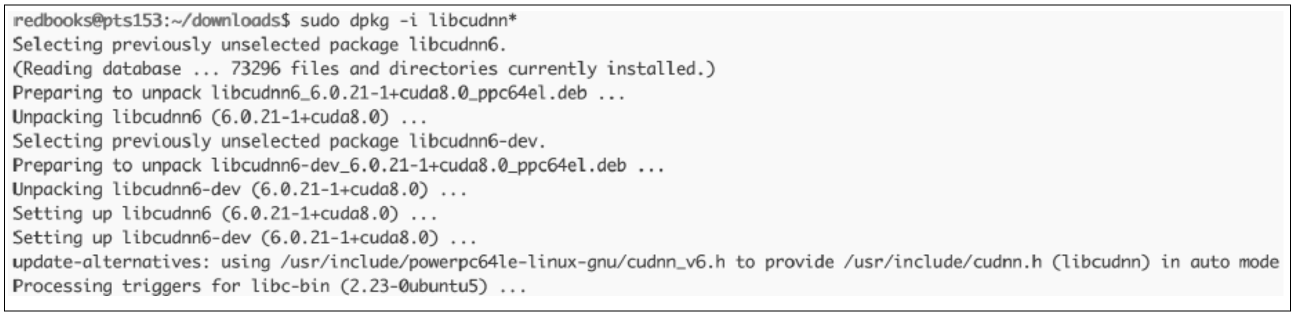
f. Set the PATH variable definition at boot time. Add these two lines to `/etc/bash.bashrc`:

```
PATH=/usr/local/cuda-8.0/bin${PATH:+:${PATH}}
LD_LIBRARY_PATH=/usr/local/cuda-8.0/lib64${LD_LIBRARY_PATH:+:{$LD_LIBRARY_PA
TH}}
```

Tip: This step is optional but preferable because IBM PowerAI will not have to export the variables for each login.

g. Install the cuDNN library by using the dpkg tool, as shown in Figure 4-71:

```
$ sudo dpkg -i libcudnn*
```



```
redbooks@pts153:~/downloads$ sudo dpkg -i libcudnn*
Selecting previously unselected package libcudnn6.
(Reading database ... 73296 files and directories currently installed.)
Preparing to unpack libcudnn6_6.0.21-1+cuda8.0_ppc64el.deb ...
Unpacking libcudnn6 (6.0.21-1+cuda8.0) ...
Selecting previously unselected package libcudnn6-dev.
Preparing to unpack libcudnn6-dev_6.0.21-1+cuda8.0_ppc64el.deb ...
Unpacking libcudnn6-dev (6.0.21-1+cuda8.0) ...
Setting up libcudnn6 (6.0.21-1+cuda8.0) ...
Setting up libcudnn6-dev (6.0.21-1+cuda8.0) ...
update-alternatives: using /usr/include/powerpc64le-linux-gnu/cudnn_v6.h to provide /usr/include/cudnn.h (libcudnn) in auto mode
Processing triggers for libc-bin (2.23-0ubuntu5) ...
```

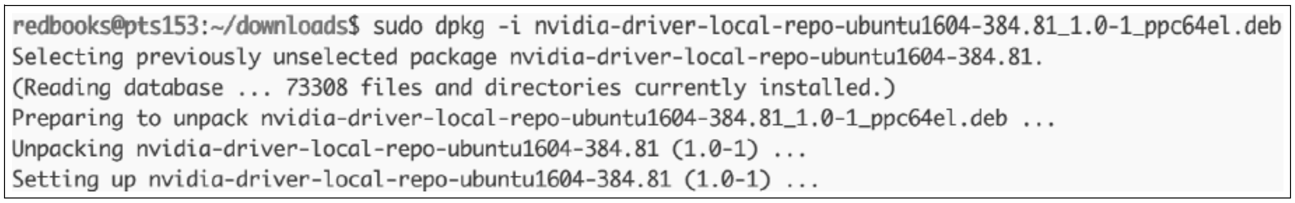
Figure 4-71 Installing the CUDA Deep Neural Network library

Note: These CUDA and cuDNN libraries include an outdated version of the NVIDIA driver. Install the current version for performance and support reasons. Table 3-7 on page 43 shows the minimum and recommended levels of NVIDIA drivers for every IBM PowerAI release.

3. Install the NVIDIA drivers:

a. Unpack the driver installation file, as shown in Figure 4-72:

```
$ sudo dpkg -i\ nvidia-driver-local-repo-ubuntu1604-384.81_1.0-1_ppc64el.deb
```



```
redbooks@pts153:~/downloads$ sudo dpkg -i nvidia-driver-local-repo-ubuntu1604-384.81_1.0-1_ppc64el.deb
Selecting previously unselected package nvidia-driver-local-repo-ubuntu1604-384.81.
(Reading database ... 73308 files and directories currently installed.)
Preparing to unpack nvidia-driver-local-repo-ubuntu1604-384.81_1.0-1_ppc64el.deb ...
Unpacking nvidia-driver-local-repo-ubuntu1604-384.81 (1.0-1) ...
Setting up nvidia-driver-local-repo-ubuntu1604-384.81 (1.0-1) ...
```

Figure 4-72 Unpacking the NVIDIA drivers

b. Resynchronize the package index files from their sources:

```
$ sudo apt-get update
```

c. Install the driver:

```
$ sudo apt-get install cuda-drivers
```

d. Restart:

```
$ sudo reboot
```

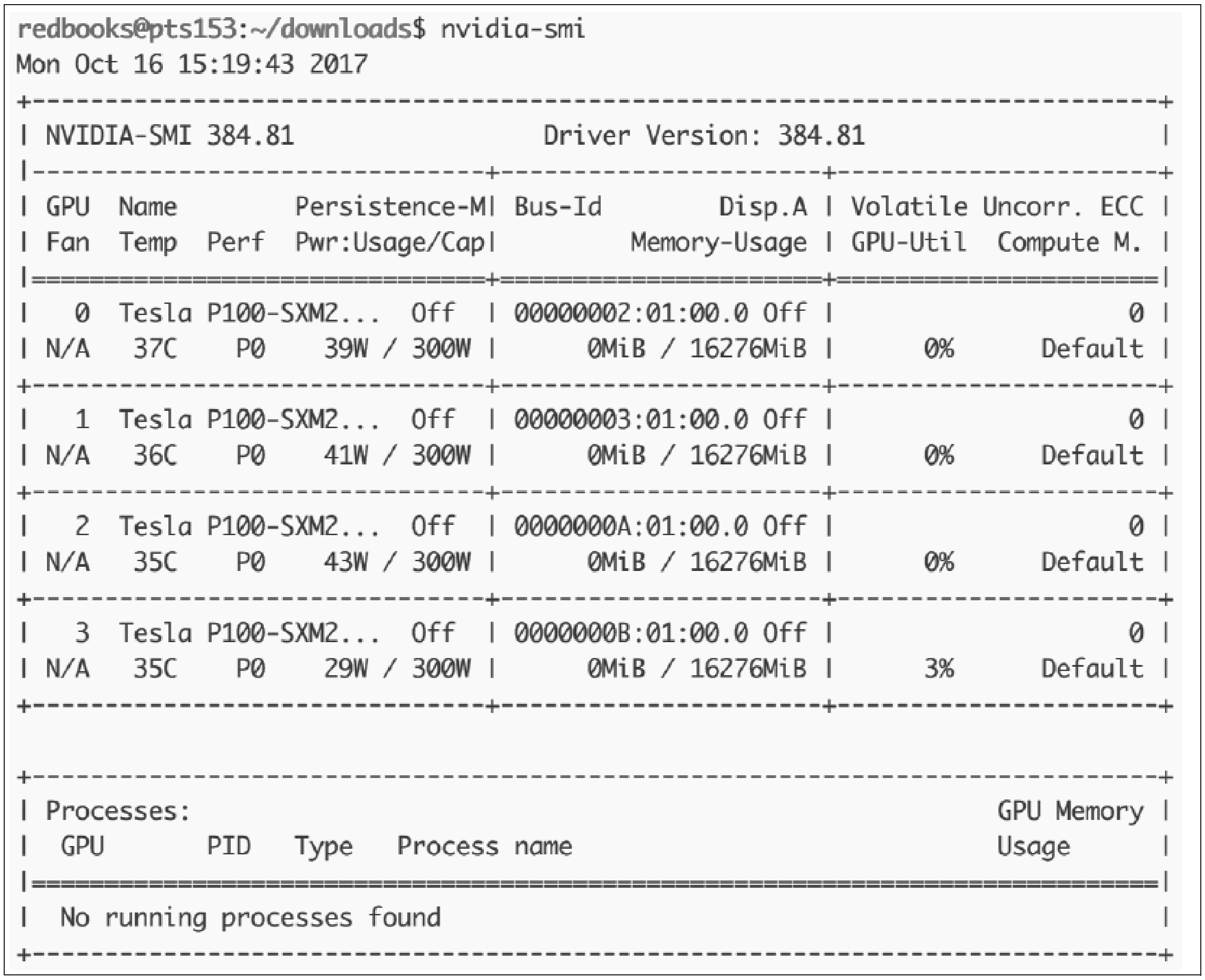
e. Check that the PATH variables are set and the components are installed:

i. Run the NVIDIA CUDA compiler:

```
$ nvcc - V
nvcc: NVIDIA (R) Cuda compiler driverCopyright (c) 2005-2016 NVIDIA
Corporation Buildt Tue_Jan_10_13: 28: 28_CST_2017
Cuda compilation tools, release 8.0, V 8.0.61
```

ii. After the system restarts, run **nvidia-smi** to check the status of the GPUs, as shown in Figure 4-73 on page 97:

```
$ nvidia-smi
```



```
redbooks@pts153:~/downloads$ nvidia-smi
Mon Oct 16 15:19:43 2017
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.81                 Driver Version: 384.81                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P100-SXM2...  Off  | 00000002:01:00.0 Off |                    0 |
| N/A   37C    P0    39W / 300W |      0MiB / 16276MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla P100-SXM2...  Off  | 00000003:01:00.0 Off |                    0 |
| N/A   36C    P0    41W / 300W |      0MiB / 16276MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla P100-SXM2...  Off  | 0000000A:01:00.0 Off |                    0 |
| N/A   35C    P0    43W / 300W |      0MiB / 16276MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla P100-SXM2...  Off  | 0000000B:01:00.0 Off |                    0 |
| N/A   35C    P0    29W / 300W |      0MiB / 16276MiB |      3%      Default |
+-------------------------------+----------------------+----------------------+


+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Figure 4-73 The nvidia-smi sample output

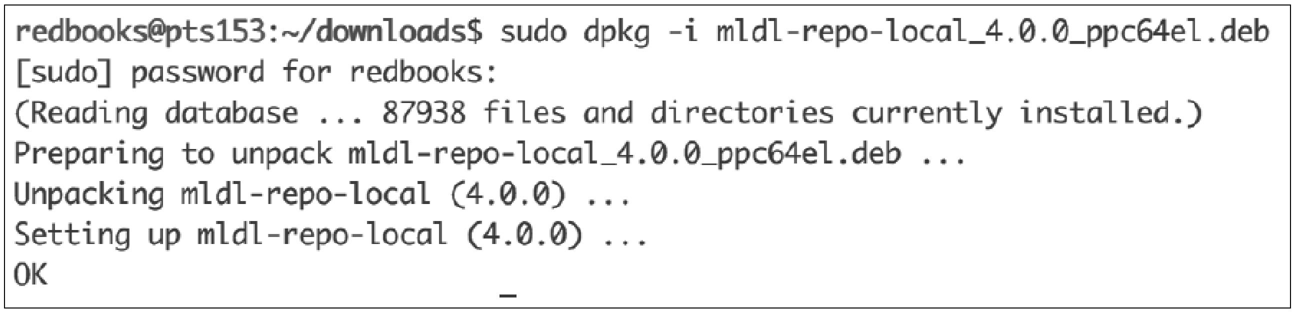
Tip: For a complete **nvidia-smi** command options reference guide, see the Ubuntu man page for this command.

Installing IBM PowerAI

This section describes the installation of IBM PowerAI:

1. Extract IBM PowerAI installer, as shown in Figure 4-74:

   ```
   sudo dpkg -i mldl-repo-local-4.0.0_ppc64el.deb
   ```

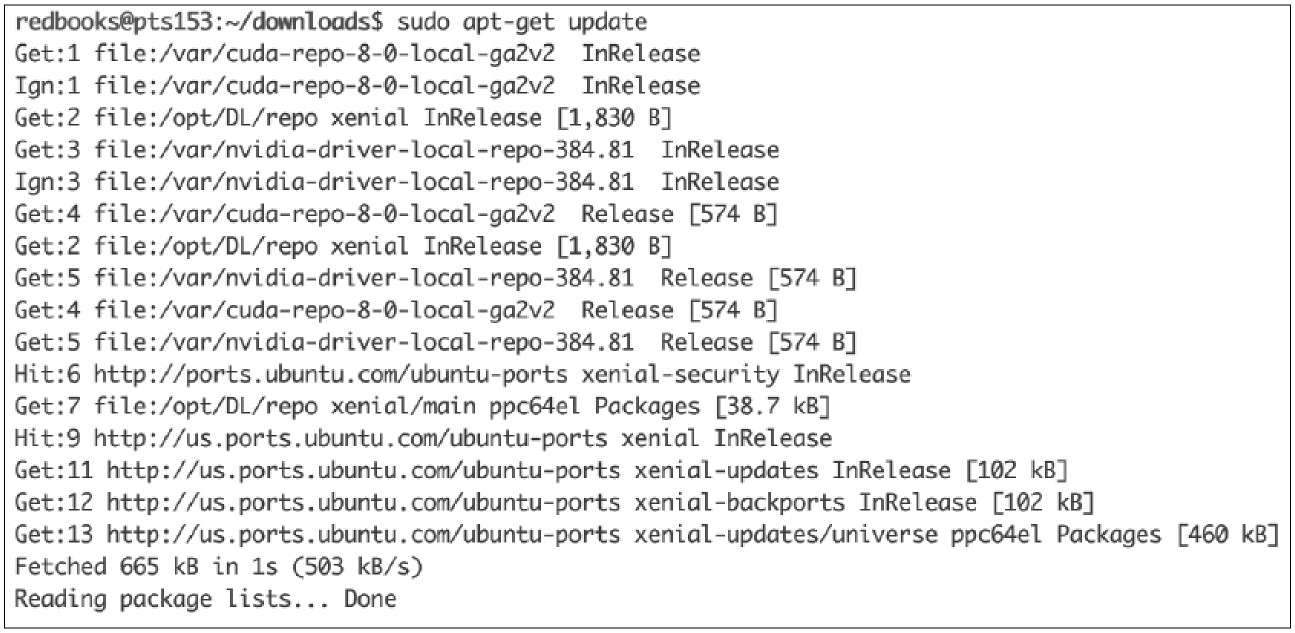


```
redbooks@pts153:~/downloads$ sudo dpkg -i mldl-repo-local_4.0.0_ppc64el.deb
[sudo] password for redbooks:
(Reading database ... 87938 files and directories currently installed.)
Preparing to unpack mldl-repo-local_4.0.0_ppc64el.deb ...
Unpacking mldl-repo-local (4.0.0) ...
Setting up mldl-repo-local (4.0.0) ...
OK
```

Figure 4-74 Unpacking the IBM PowerAI meta-package

2. Resynchronize the package index files from their sources, as shown in Figure 4-75:

   ```
   $ sudo apt-get update
   ```



```
redbooks@pts153:~/downloads$ sudo apt-get update
Get:1 file:/var/cuda-repo-8-0-local-ga2v2  InRelease
Ign:1 file:/var/cuda-repo-8-0-local-ga2v2  InRelease
Get:2 file:/opt/DL/repo xenial InRelease [1,830 B]
Get:3 file:/var/nvidia-driver-local-repo-384.81  InRelease
Ign:3 file:/var/nvidia-driver-local-repo-384.81  InRelease
Get:4 file:/var/cuda-repo-8-0-local-ga2v2  Release [574 B]
Get:2 file:/opt/DL/repo xenial InRelease [1,830 B]
Get:5 file:/var/nvidia-driver-local-repo-384.81  Release [574 B]
Get:4 file:/var/cuda-repo-8-0-local-ga2v2  Release [574 B]
Get:5 file:/var/nvidia-driver-local-repo-384.81  Release [574 B]
Hit:6 http://ports.ubuntu.com/ubuntu-ports xenial-security InRelease
Get:7 file:/opt/DL/repo xenial/main ppc64el Packages [38.7 kB]
Hit:9 http://us.ports.ubuntu.com/ubuntu-ports xenial InRelease
Get:11 http://us.ports.ubuntu.com/ubuntu-ports xenial-updates InRelease [102 kB]
Get:12 http://us.ports.ubuntu.com/ubuntu-ports xenial-backports InRelease [102 kB]
Get:13 http://us.ports.ubuntu.com/ubuntu-ports xenial-updates/universe ppc64el Packages [460 kB]
Fetched 665 kB in 1s (503 kB/s)
Reading package lists... Done
```

Figure 4-75 Updating the package information

3. If you plan to use certain frameworks, there are extra steps to perform:

   a. Installation note for IBM Caffe and Distributed Deep Learning (DDL) custom operator for TensorFlow

      The `caffe-ibm` and `ddl-tensorflow` packages require the IBM PowerAI OpenMPI package, which is built with NVIDIA CUDA support. That OpenMPI package conflicts with Ubuntu non-CUDA-enabled OpenMPI packages.

      Uninstall any `openmpi` or `libopenmpi` packages before installing IBM Caffe or DDL custom operator for TensorFlow. Purge any configuration files to avoid interference:

      ```
      $ dpkg -l | grep openmpi
         $ sudo apt-get purge ...
      ```

   b. Installation note for DIGITS

      The `digits` and `python-socketio-server` packages conflict with the Ubuntu older `python-socketio` package. Uninstall the `python-socketio` package before installing DIGITS.

4. Install IBM PowerAI libraries and frameworks:

```
$ sudo apt-get install power-mldl
```

Tip: It is also possible to install individual frameworks instead of the complete package by running the following command:

```
$ sudo apt-get install <framework name>
```

The framework packages that are included in IBM PowerAI V4.0 are the following ones:

- `caffe-bvlc`L Berkeley Vision and Learning Center (BVLC) upstream Caffe V1.0.0
- `caffe-ibm`: IBM Optimized version of BVLC Caffe V1.0.0
- `caffe-nv`: NVIDIA fork of Caffe V0.15.14
- `chainer`: Chainer V1.23.0
- `digits`: DIGITS V5.0.0
- `tensorflow`: Google TensorFlow V1.1.0
- `ddl-tensorflow`: DDL custom operator for TensorFlow
- `theano`: Theano V0.9.0
- `torch`: Torch V7

For a list of components that are included in all IBM PowerAI releases, see 3.2, "IBM PowerAI compatibility matrix" on page 58.

5. Depending on the IBM PowerAI release and the packages that are installed in step 3 on page 98, the content of this directory can be different. Check that the components are installed by running the following command:

```
$ ls /opt/DL/
bazel
caffe
caffe-bvlc
caffe-ibm
caffe-nv
chainer
ddl
ddl-tensorflow
digits
nccl
openml
...
```

Note about Python setup tools and easy_install:

The python **`easy_install`** utility can interfere with the proper functioning of some of the IBM PowerAI framework packages, including TensorFlow and Chainer.

The IBM PowerAI packages include local copies of Python modules such as protobuf (TensorFlow) and pillow (Chainer) because they require versions newer than those that are provided by Canonical and Ubuntu. The `<framework>-activate` scripts set up the pathing that is needed to make it work (they set `PYTHONPATH` to give the local copies priority over the system default versions).

The **`easy_install`** utility adds a script that can cause the system's default paths to be searched ahead of `PYTHONPATH` entries, which might result in protobuf or pillow related failures in TensorFlow and Chainer.

If easy-install runs as root, the problematic script can be found in the following directory:

```
/usr/local/lib/python2.7/dist-packages/easy-install.pth
```

Optimizing the environment

To get the most out of the system, there is more tuning to perform, and in some cases this tuning depends on the framework that is used:

- Enable Performance Governor:

  ```
  $ sudo apt-get install linux-tools-common linux-tools-generic cpufrequtils
  lsb-release
  $ sudo cpupower -c all frequency-set -g performance
  ```

- SMT value for TensorFlow:

  This is a specific parameter to set for TensorFlow workloads. This reduces the CPU parallelism by changing the value of SMT from default (8) to the recommended value (2).

  ```
  $ sudo apt-get install powerpc-ibm-utils
  $ sudo ppc64_cpu --smt=2
  ```

  If you are going to run other workloads in the system, it can negatively impact these workloads.

- HW parameters valid for all frameworks:

  – Enable NVIDIA driver persistence mode:

    This prevents the driver from *hanging* during the run time, which removes any latency while waiting for driver initialization between epochs:

    ```
    $ sudo nvidia-smi -pm ENABLED
    Enabled persistence mode for GPU 00000002:01:00.0.
    Enabled persistence mode for GPU 00000003:01:00.0.
    Enabled persistence mode for GPU 0000000A:01:00.0.
    Enabled persistence mode for GPU 0000000B:01:00.0.
    All done.
    ```

  – Set all GPU clocks to the optimum value:

    ```
    $ sudo nvidia-smi -ac 715,1480
    Applications clocks set to “(MEM 715, SM 1480)” for GPU 00000002:01:00.0
    Applications clocks set to “(MEM 715, SM 1480)” for GPU 00000002:03:00.0
    Applications clocks set to “(MEM 715, SM 1480)” for GPU 00000002:0A:00.0
    Applications clocks set to “(MEM 715, SM 1480)” for GPU 00000002:0B:00.0
    ```

Note: For more information about NVIDIA driver persistence mode, see GPn GPU Deployment and Management Documentation.

4.2 Testing IBM PowerAI V1.4

As with other machine learning (ML) environments, before you can use the framework, you must first source it so that your run time knows that the library is available. This task can be completed on an individual basis through the shell, globally in `/etc/bash.basrc`, or on a per user basis in `~/.bashrc`:

```
$ source /opt/DL/<framework>/bin/<framework>-activate
```

All the frameworks in IBM PowerAI come with a simple test that you can run to ensure that everything is in working order. To run the test, ensure that you have sourced the framework, and then run the following command:

```
$ <framework>-test
```

The `<framework>-test` syntax is the same for all of the bundled frameworks.

For more information about the tutorial for using each framework, see Getting Started with <framework in the IBM PowerAI manual.

4.2.1 First test: TensorFlow test program

To test TensorFlow (or any other framework), complete the following steps:

1. Source the framework:

   ```
   $ source /opt/DL/tensorflow/bin/tensorflow-activate
   ```

2. Run the test:

   ```
   $ tensorflow-test
   tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library
   libcupti.so.8.0 locally0.919
   ========================================================================DONE!!!
   ```

This is a technical verification that everything is installed correctly. Now, you can run some workloads.

4.2.2 Utilization of a multilayer perceptron on a sample data set

The first test runs on an open source data set. For this test, as shown in Figure 4-76, an MNIST database of handwritten numbers was chosen.

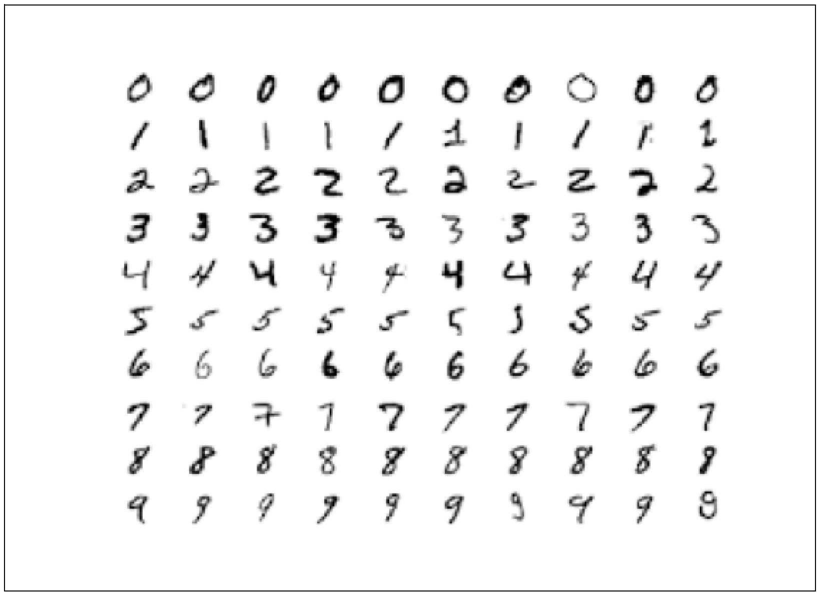


Figure 4-76 MNIST data set sample

MNIST data set

Here is the MNIS data set.

This data set has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits are size-normalized and centered in a fixed-size image.

The original black and white (bilevel) images from NIST are size-normalized to fit in a 20 x 20 pixel box while preserving their aspect ratio. The resulting images contain gray levels because of the anti-aliasing technique that is used by the normalization algorithm. The images are centered in a 28 x 28 image by computing the center of mass of the pixels, and converting the image to position this point at the center of the 28 x 28 field.

Multilayer perceptron

The most basic and most commonly used forward-propagation network among neural networks is multilayer perceptron (MLP). An MLP is a class of feed-forward artificial neural network. An MLP consists of at least three layers of nodes. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP uses a supervised learning technique that is called *back propagation* for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

MLPs are sometimes colloquially referred to as *vanilla* neural networks, especially when they have a single hidden layer.

Figure 4-77 on page 103 shows a possible network architecture for an MLP that is used for character recognition over an MNIST data set.

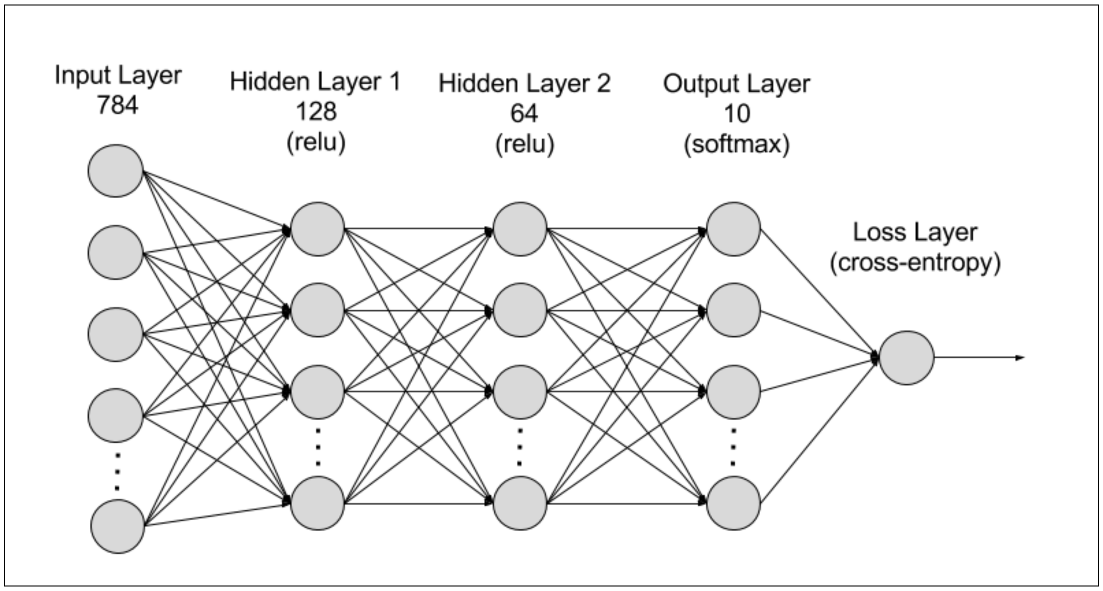


Figure 4-77 Multilayer perceptron network architecture for MNIST

4.2.3 Using Caffe with MNIST

This section describes how to use Caffe with MNIST:[9]

1. Activate Caffe:

```
$ source /opt/DL/caffe-bvlc/bin/caffe-activate
```

2. Copy a sample of Caffe to any test directory (in this example, `./caffe`) by using the sample copy script that is included with IBM PowerAI:

```
$ caffe-install-samples ./caffe
$ cd ./caffe
```

3. Download the MNIST data:

```
$ ./data/mnist/get_mnist.sh
```

4. Convert the MNIST data to LMDB format by using Caffe:

```
$ ./examples/mnist/create_mnist.sh
```

5. Run the training and testing:

```
$ ./examples/mnist/train_lenet.sh
I0609 12:02:41.426894 72649 data_layer.cpp:73]
Restarting data prefetching from start.
I0609 12:02:41.427927 72583 solver.cpp: 398]
Test net output #0: accuracy = 0.9915I0609
12:02:41.427958 72583 solver.cpp: 398] )
Test net output #1: loss = 0.027027
(* 1 = 0.027027 loss)
```

[9] For more information, see http://caffe.berkeleyvision.org/gathered/examples/mnist.html.

4.2.4 Using Caffe with TensorFlow

This section describes how to use Caffe with TensorFlow:[10]

1. Activate TensorFlow:

```
$ source / opt / DL / tensorflow / bin / tensorflow-activate
```

2. Download TensorFlow from GitHub and expand the tutorial script group to an arbitrary directory (`~ / tensorflow` in this case):

```
$ mkdir ~/tensorflow
$ git clone https://github.com/tensorflow/tensorflow.git
$ cd clone/tensorflow/examples/tutorials/mnist/
```

3. Run the training and testing:

```
$ cd clone/tensorflow/examples/
$ python mnist_softmax.py
[tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library
libcublas.so.8.0
locally [tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA
library libcudnn.so.5
locally [tensorflow/stream_executor/dso_loader.cc:135] Successfully opened CUDA
library libcurand.so.8.0 successfully opened CUDA library libcuda.so.1
locally [tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA
library libcurand.so.8.0
locally
Extracting /tmp/tensorflow/mnist/input_data/train-images-idx3-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data/train-labels-idx1-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data/t10k-images-idx3-ubyte.gz
:
[tensorflow/compiler/xla/service/service.cc:187] StreamExecutor device (2):
Tesla P100-SXM2-16GB, Compute Capability 6.0
[tensorflow/compiler/xla/service/service.cc:187] StreamExecutor device (3):
Tesla P100-SXM2-16GB, Compute Capability 6.0

0.9176
```

4.3 Setting up IBM PowerAI V1.5.0 on a POWER S822LC for High Performance Computing server

This setup guide is based on the official IBM PowerAI release notes.

4.3.1 Deep learning software packages

IBM PowerAI V1.5 provides software packages for several DL frameworks, supporting libraries, and tools:

- DL frameworks
  - Caffe: BVLC and IBM variants
  - TensorFlow
- Supporting libraries:
  - NVIDIA Collective Communications Library (NCCL)
  - OpenBLAS

[10] The tests that follow the TensorFlow MNIST tutorial can be found at `https://www.tensorflow.org/tutorials/`.

- Tools”:
  - Bazel
  - TensorBoard

Note: For more information, see IBM PowerAI. Developer resources can be found at IBM PowerAI Developer.

It also includes a technology preview of IBM PowerAI DDL. DDL supports distributed (multi-host) model training. TensorFlow support is provided by a separate package that is included in the IBM PowerAI distribution.

All the packages are intended for use with the IBM Power Systems S822LC for High Performance Computing with NVIDIA Tesla P100 GPUs servers, RHEL 7.4 LE with NVIDIA CUDA 9.0, and cuDNN V7.0 packages.

Note: To avoid conflicts with untested or unsupported versions of certain system libraries, disable automatic updates.

It is not mandatory to install security updates automatically. You, as the sysadmin, can determine whether your system is exposed to threats, and whether you need Linux updates to be deployed automatically.

4.3.2 System setup

This section describes how to perform the system setup.

Operating system repository setup

This section describes how to set up the operating system repository:

1. Enable the optional and extra repository channels:

```
$ sudo subscription-manager repos --enable=rhel-7-for-power-le-optional-rpms
$ sudo subscription-manager repos --enable=rhel-7-for-power-le-extras-rpms
```

2. Enable the EPEL repository:

```
$ wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
$ sudo rpm -ihv epel-release-latest-7.noarch.rpm
```

NVIDIA components

The DL packages require CUDA, cuDNN, and GPU driver packages from NVIDIA.

The required and recommended versions of these components are shown in Table 4-1.

Table 4-1 NVIDIA components for IBM PowerAI V1.5

Component	Required	Recommended
CUDA Toolkit	9.0.176	
cuDNN	7.0.4	
GPU Driver	384.81	

To install these components, complete the following steps:

1. Download and install NIVDIA CUDA 9.0:
   a. Select Operating System: **Linux**
   b. Select Architecture: **ppc64le**
   c. Select Distribution **RHEL**
   d. Select Version **7**
   e. Select the Installer Type that fits your needs.
   f. Follow the IBM PowerLinux™ installation instructions in the CUDA Quick Start Guide found at NIVDIA CUDA 9.0, including the steps describing how to set up the CUDA development environment by updating `PATH` and `LD_LIBRARY_PATH`.
2. Download NVIDIA cuDNN 7.0.4 for CUDA 9.0 (cuDNN V7.0.4 Library for Linux (Power)).

   Note: Registration in the NVIDIA Accelerated Computing Developer Program is required.

3. Install the cuDNN V7.0 packages:

```
$ sudo tar -C /usr/local --no-same-owner -xzvf cudnn-9.0-linux-ppc64le-v7.tgz
```

Anaconda

A number of the DL frameworks require Anaconda. Anaconda is a platform-neutral data science distribution with a collection of 1,000+ open source packages with complementary community support.

You need to download and install Anaconda. The installation requires input for a license agreement, installation location (the default is `$HOME/anaconda2`), and permission to modify the `PATH` environment variable (by way of `.bashrc`) as follows:

```
$ wget https://repo.continuum.io/archive/Anaconda2-5.0.0-Linux-ppc64le.sh
$ sudo yum install bzip2
$ bash Anaconda2-5.0.0-Linux-ppc64le.sh
$ source ~/.bashrc
```

Note: `Anaconda2-5.0.0-Linux-ppc64le.sh` is a large file (283 MB) and has an md5sum of 157890d591c61a9b511f8452476d6d19.

If multiple users are using the same system, each user must install Anaconda individually.

Note: Here is a framework-specific Anaconda setup for IBM PowerAI V1.5 for POWER9.

Anaconda is required for the following items:

- TensorFlow ("tensorflow")
- TensorBoard ("tensorboard")

Each of those packages includes an `install_dependencies` script that installs the necessary packages in the user's Anaconda environment:

```
$ /opt/DL/<framework>/bin/install_dependencies
```

Then, the framework can be used as usual:

```
$ source /opt/DL/<framework>/bin/<framework>-activate
$ <framework>-test
```

4.3.3 Installing the deep learning frameworks

This section describes how to install the DL frameworks.

IBM Spectrum MPI installation

The IBM PowerAI Deep Learning packages depend on IBM Spectrum MPI. To install it, complete the following steps:

1. Download IBM Spectrum MPI by completing the following steps:
   a. Sign in with your IBM ID.
   b. Select **IBM Spectrum MPI v10.1** and click **Continue**.
   c. View and agree to the license.
   d. Select **ibm_smpi_lic_s-10.1Eval-rh7_Aug11.ppc64le.rpm**.
   e. Select **ibm_smpi-10.1.1.0Eval-rh7_Aug11.ppc64le.rpm**.
   f. Click **Download now**.
2. Install the RPMs on your system by running the following command:

```
$ sudo rpm -ihv ibm_smpi_lic_s-10.1Eval-rh7_Aug11.ppc64le.rpm
ibm_smpi-10.1.1.0Eval-rh7_Aug11.ppc64le.rpm
```

Software repository setup

The IBM PowerAI Deep Learning packages are distributed in a `tar.gz` file containing an RPM and a readme file. The `tar.gz` file must be extracted on the local machine. Installing the RPM creates an installation repository on the local machine.

Install the repository package by running the following command:

```
$ sudo rpm -ihv mldl-repo-*.rpm
```

Installing all frameworks concurrently

The DL frameworks can be installed concurrently by using the power-mldl meta-package:

```
$ sudo yum install power-mldl-cuda9.0
```

Installing frameworks individually

The DL frameworks can be installed individually if you prefer. Here are the framework packages:

- `caffe-bvlc`: BVLC upstream Caffe V1.0.0
- `caffe-ibm`: IBM Optimized version of BVLC Caffe V1.0.0
- `tensorflow`: Google TensorFlow V1.4.0
- `tensorboard`: Web Applications for Inspecting TensorFlow Runs and Graphs V0.4.0rc3
- `ddl-tensorflow`: DDL custom operator for TensorFlow

Each can be installed by running the following command:

```
$ sudo yum install <framework>-cuda9.0
```

Accepting the license agreement

Read the license agreements and accept the terms and conditions before using IBM Spectrum MPI or any of the frameworks by running the following commands:

```
$ sudo IBM_SPECTRUM_MPI_LICENSE_ACCEPT=no
/opt/ibm/spectrum_mpi/lap_se/bin/accept_spectrum_mpi_license.sh

$ sudo /opt/DL/license/bin/accept-powerai-license.sh
```

After reading the license agreements, future installation s can be automated to silently accept the license agreements by running the following commands:

```
$ sudo IBM_SPECTRUM_MPI_LICENSE_ACCEPT=yes
/opt/ibm/spectrum_mpi/lap_se/bin/accept_spectrum_mpi_license.sh

$ sudo IBM_POWERAI_LICENSE_ACCEPT=yes
/opt/DL/license/bin/accept-powerai-license.sh
```

4.3.4 Tuning recommendations

Here are the preferred settings for optimal DL performance on the Power S822LC for High Performance Computing:

- Enable Performance Governor:

  ```
  $ sudo yum install kernel-tools
  $ sudo cpupower -c all frequency-set -g performance
  ```

- Enable GPU persistence mode:

  ```
  $ sudo systemctl enable nvidia-persistenced
  $ sudo systemctl start nvidia-persistenced
  ```

- Set GPU memory and graphics clocks (P100 GPU only):

  ```
  $ sudo nvidia-smi -ac 715,1480
  ```

- For TensorFlow, set the SMT mode:

  ```
  $ sudo ppc64_cpu --smt=2
  ```

4.3.5 Getting started with machine learning and deep learning frameworks

This section shows how to get started with the machine learning and deep learning (MLDL) frameworks.

General setup

Most of the IBM PowerAI packages install outside the normal system search paths (to `/opt/DL/...`), so each framework package provides a shell script to simplify environmental setup (for example, `PATH`, `LD_LIBRARY_PATH`, `PYTHONPATH`).

Update your shell rc file (`.bashrc`) to source the necessary setup scripts. For example:

```
source /opt/DL/<framework>/bin/<framework>-activate
```

Any errors that occur when you run the activate scripts must be resolved before you use the framework.

Each framework also provides a test script to verify basic functions:

```
$ <framework>-test
```

Dependencies

Many IBM PowerAI frameworks (for example, TensorFlow, TensorBoard, and Caffe) have their dependencies satisfied by Anaconda packages. These dependencies are validated by the `<framework>-activate` script to ensure that they are installed. If they are not installed, the script fails.

For these frameworks, the `/opt/DL/<framework>/bin/install_dependencies` script must be run before activation to install the required packages.

For example:

```
$ source /opt/DL/tensorflow/bin/tensorflow-activate
Missing dependencies ['backports.weakref', 'mock', 'protobuf']
Run "/opt/DL/tensorflow/bin/install_dependencies" to resolve this problem.

$ /opt/DL/tensorflow/bin/install_dependencies
Fetching package metadata ...........
Solving package specifications: .

Package plan for installation in environment /home/rhel/anaconda2:

The following NEW packages will be INSTALLED:

    backports.weakref: 1.0rc1-py27_0
    libprotobuf:       3.4.0-hd26fab5_0
    mock:              2.0.0-py27_0
    pbr:               1.10.0-py27_0
    protobuf:          3.4.0-py27h7448ec6_0

Proceed ([y]/n)? y

libprotobuf-3. 100% |###############################| Time: 0:00:02   2.04 MB/s
backports.weak 100% |###############################| Time: 0:00:00  12.83 MB/s
protobuf-3.4.0 100% |###############################| Time: 0:00:00   2.20 MB/s
pbr-1.10.0-py2 100% |###############################| Time: 0:00:00   3.35 MB/s
mock-2.0.0-py2 100% |###############################| Time: 0:00:00   3.26 MB/s
```

```
$ source /opt/DL/tensorflow/bin/tensorflow-activate
$
```

Getting started with Caffe

This section describes how to get started with Caffe.

Caffe alternatives

Packages are provided for upstream BVLC Caffe (`/opt/DL/caffe-bvlc`) and IBM optimized Caffe (`/opt/DL/caffe-ibm`). The system default Caffe (`/opt/DL/caffe`) can be selected by using the operating system's alternatives as follows:

```
$ sudo update-alternatives --config caffe
There are 2 programs which provide 'caffe'.

Selection    Command
-----------------------------------------------
1           /opt/DL/caffe-bvlc
*+ 2        /opt/DL/caffe-ibm
Enter to keep the current selection[+], or type selection number:
```

Users can activate the system default Caffe:

```
source /opt/DL/caffe/bin/caffe-activate
```

Users can activate a specific variant. For example:

```
source /opt/DL/caffe-bvlc/bin/caffe-activate
```

Attempting to activate multiple Caffe packages in a single login session causes unpredictable behavior.

Note: The `caffe-test` failure in `DeconvolutionLayerTest`

Running `caffe-test` can fail with the test case `5 tests from DeconvolutionLayerTest/0, where TypeParam = caffe::CPUDevice`. This is a known issue that is documented in Caffe Issue 4083. The workaround is to set the variable `OMP_NUM_THREADS` to the value of `1` before running the test:

```
$ source /opt/DL/caffe/bin/caffe-activate
$ export OMP_NUM_THREADS=1
$ caffe-test
```

Caffe samples and examples

Each Caffe package includes example scripts and sample models. A script is provided to copy the sample content into a specified directory:

```
$ caffe-install-samples <somedir>
```

References

For tutorials and example programs that you can run to get started, see Caffe.

Here are links to example programs:

- LeNet MNIST Tutorial trains a neural network to understand handwritten digits.
- CIFAR-10 Tutorial trains a CNN to classify small images.

Optimizations in IBM Caffe

The IBM Caffe package (`caffe-ibm`) in IBM PowerAI is based on BVLC Caffe and includes optimizations and enhancements from IBM:

- CPU and GPU layer-wise reduction
- Large Model Support (LMS)
- IBM PowerAI DDL

Command-line options

IBM Caffe supports all BVLC Caffe options and adds new ones to control the enhancements:

- **`-bvlc`**: Disables CPU and GPU layer-wise reduction.
- **`-threshold`**: Tunes CPU and GPU layer-wise reduction. If the number of parameters for one layer is greater than or equal to the threshold, their accumulation on the CPU is done in parallel. Otherwise, the accumulation is done by using one thread. By default, it is set to 2,000,000.
- **`-ddl ["-option1 param -option2 param"]`**: Enables DDL with an optional space-delimited parameter string. The supported parameters are as follows:
  - **`mode <mode>`**
  - **`dumo_iter <N>`**
  - **`dev_sync <0`**, **`1`**, or **`2>`**
  - **`rebind_iter <N>`**
  - **`dbg_level <0`**, **`1`**, or **`2>`**
- **`-ddl_update`**: This option instructs Caffe to use a new custom version of the **`ApplyUpdate`** function that is optimized for DDL. It is faster, although it does not support gradient clipping, so it is off by default. It can be used in networks that do not support clipping.
- **`-ddl_align`**: This option ensures that the gradient buffers have a length that is a multiple of 256 bytes and have start addresses that are multiples of 256, which ensures cache line alignment on multiple platforms and alignment with NCCL slices. This option is off by default.
- **`-ddl_database_restart`**: This option ensures that every learner always looks at the same data set during an epoch, which enables a system to cache only the pages that are touched by the learners that are contained within it. It can help size the number of learners that are needed for a given data set size by establishing a known database footprint per system. This option is off by default.
- **`-lms <size>`**: This option enables LMS with a threshold of `<size>`.
- **`-lms_frac <fraction>`**: This option tunes LMS memory usage between the CPU and GPU.

Table 4-2 shows the command-line options.

Table 4-2 IBM Caffe command options

Feature	`-blvc`	`-ddl`	`-lms`	`-gpu`
CPU/GPU layer-wise reduction	Do not specify.	Do not care.	Do not care.	Multiple GPUs.
DDL	Do not care.	Do specify.	Do not care.	Do not specify.
LMS	Do not care.	Do not care.	Do specify.	Do not care.

LMS is effective regardless of the other options that are used if **`-lms`** is specified. For example, you can use DDL and LMS together.

CPU and GPU layer-wise reduction is enabled only if multiple GPUs are specified and `layer_wise_reduce` is `false`.

Using multiple GPUs with DDL is specified by the Message Passing Interface (MPI) rank file, so the **`-gpu`** flag cannot be used to specify multiple GPUs for DDL.

About CPU and GPU layer-wise reduction

This optimization aims to reduce the running time of a multiple-GPU training by using CPUs. In particular, gradient accumulation is offloaded to CPUs and done in parallel with the training. To gain the best performance with IBM Caffe, close unnecessary applications that consume a high percentage of CPU.

Note: If you use a single GPU, IBM Caffe and BVLC Caffe have similar performance.

The optimizations in IBM Caffe do not change the convergence of a neural network during training. IBM Caffe and BVLC Caffe produce the same convergence results.

CPU and GPU layer-wise reduction is enabled unless the **`-bvlc`** command-line flag is used.

About IBM PowerAI Distributed Deep Learning

For more information about using IBM PowerAI DDL, see `/opt/DL/ddl-doc/doc/README.md`.

About Large Model Support

IBM Caffe with LMS loads the neural model and data set in system memory and caches activity to GPU memory, enabling models and the training batch size to scale significantly beyond what was previously possible. LMS is available as a technology preview.

You can enable the LMS by adding **`-lms <size in KB>`**, for example **`-lms 1000`**. Then, any memory chunk larger than 1000 KB is kept in CPU memory, and fetched to GPU memory only when needed for computation. Thus, if you pass a large value such as **`-lms 10000000000`**, it effectively disables the feature, and a small value means more aggressive LMS. The value is used to control the performance tradeoff.

As a secondary option, there is **`-lms_frac <0~1.0>`**. For example, with **`-lms_frac 0.4`**, LMS does not take effect until more than at least 40% of GPU memory is expected to be taken. This is useful for disabling LMS for a small network.

Combining Large Model Support and Distributed Deep Learning

LMS and DDL can be combined, for example by running the following command:

```
$ mpirun -x PATH -x LD_LIBRARY_PATH -rf 4x4x2.rf -n 8 caffe train -solver
alexnet_solver.prototxt -d
```

Getting started with TensorFlow

The TensorFlow home page has a variety of information, including tutorials, how to instructions, and a getting started guide.

For more tutorials and examples, see the following resources:

- TensorFlow Tutorials
- TensorFlow Tutorial and Examples for Beginners with Latest APIs

Distributed Deep Learning custom operator for TensorFlow

This release of IBM PowerAI includes a technology preview of the IBM PowerAI DDL custom operator for TensorFlow. The DDL custom operator uses IBM Spectrum MPI and NCCL to provide high-speed communications for distributed TensorFlow.

The DDL custom operator can be found in the `ddl-tensorflow` package. For more information about DDL and about the TensorFlow operator, see the following readme files:

- `/opt/DL/ddl-doc/doc/README.md`
- `/opt/DL/ddl-tensorflow/doc/README.md`
- `/opt/DL/ddl-tensorflow/doc/README-API.md`

The DDL TensorFlow operator makes it easy to enable Slim-style models for distribution. The following package includes examples of Slim models that are enabled with DDL:

```
$ source /opt/DL/ddl-tensorflow/bin/ddl-tensorflow-activate
$ ddl-tensorflow-install-samples <somedir>
```

These examples are based on a specific commit of the TensorFlow models repository with a small adjustment. If you prefer to work from an upstream clone rather than the packaged examples, run the following commands:

```
$ git clone https://github.com/tensorflow/models.git
$ cd models
$ git checkout 11883ec6461afe961def44221486053a59f90a1b
$ git revert fc7342bf047ec5fc7a707202adaf108661bd373d
$ cp /opt/DL/ddl-tensorflow/examples/slim/train_image_classifier.py slim/
```

Additional TensorFlow features

The IBM PowerAI TensorFlow packages include TensorBoard.

TensorFlow V1.4.0 package includes support for additional features:

- HDFS
- NCCL
- Experimental XLA JIT compilation

4.3.6 Uninstalling machine learning and deep learning frameworks

The MLDL framework packages can be uninstalled individually the same way that they were installed. To uninstall all MLDL packages and the repositories that were used to install them, run the following commands:

```
$ sudo yum remove powerai-license
$ sudo yum remove mldl-repo-local-cuda9.0
```

4.4 IBM PowerAI V1.5.0 setup guide for POWER AC922

This section is based on the official IBM PowerAI release notes.

4.4.1 Deep learning software packages

IBM TensorFlow ESP for Power AC922 provides software packages for several DL frameworks, supporting libraries, and tools:

- Bazel
- NCCL
- OpenBLAS
- TensorFlow
- TensorBoard

It also includes a technology preview of IBM PowerAI DDL. DDL supports distributed (multi-host) model training. TensorFlow support is provided by a separate package that is included in the IBM PowerAI distribution.

Note: The IBM TensorFlow ESP for Power AC922 packages, the IBM Spectrum MPI packages, and a readme file are available at Box. Contact your ESP representative for access.

All the packages are intended for use with the Power AC922 with NVIDIA Tesla V100 GPUs, RHEL 7.4 LE with NVIDIA CUDA 9.1, and cuDNN V7.0 packages.

For more information about IBM PowerAI, see IBM PowerAI.

Developer resources can be found at Deep Learning and PowerAI Development.

4.4.2 System setup

This section describes setting up of the IBM PowerAI system.

Operating system

The DL packages require RHEL 7.4 for IBM POWER9. The RHEL installation image and license must be acquired from Red Hat.

Operating system and repository setup

To set up the OS and repository, complete the following steps:

1. Enable the 'optional' and 'extra' repository channels:

   ```
   $ sudo subscription-manager repos --enable=rhel-7-for-power-9-optional-rpms
   $ sudo subscription-manager repos --enable=rhel-7-for-power-9-extras-rpms
   ```

2. Install the packages that are needed for the installation:

   ```
   $ sudo yum -y install wget nano bzip2
   ```

3. Enable the EPEL repository:

   ```
   $ wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
   $ sudo rpm -ihv epel-release-latest-7.noarch.rpm
   ```

4. Load the current kernel:

```
$ sudo yum update kernel kernel-tools kernel-tools-libs kernel-bootwrapper
$ reboot # This reboot may be deferred until after the NVIDIA steps below.
```

   Or do a full update:

```
$ sudo yum update
$ sudo reboot # This reboot may be deferred until after the NVIDIA steps below.
```

NVIDIA components

Before you install the NVIDIA components, the udev Memory Auto-Onlining Rule must be disabled for the CUDA driver to function properly. To disable it, complete the following steps:

1. Edit the `/lib/udev/rules.d/40-redhat.rules` file:

```
$ sudo nano /lib/udev/rules.d/40-redhat.rules
```

2. Comment out the following line and save the change:

```
SUBSYSTEM=="memory", ACTION=="add", PROGRAM="/bin/uname -p", RESULT!="s390*",
ATTR{state}=="offline", ATTR{state}="online"
```

- Restart the system for the changes to take effect:

```
$ sudo reboot
```

The DL packages require the CUDA, cuDNN, and GPU driver packages from NVIDIA.

The required and recommended versions of these components are shown in Table 4-3.

Table 4-3 NVIDIA components for IBM PowerAI V1.5l for POWER9

Component	Required	Recommended
CUDA Toolkit	9.1	9.1.85
cuDNN	7.0.5	
GPU Driver	387.36	

To install these components, complete the following steps:

1. Download and install NVIDIA CUDA 9.1 from CUDA Downloads:
   a. Select Operating System: **Linux**.
   b. Select Architecture: **ppc64le**.
   c. Select Distribution **RHEL**.
   d. Select Version **7**.
   e. Select the Installer Type that best fits your needs.
   f. Follow the Linux POWER9 installation instructions in the CUDA Quick Start Guide, including the steps that describe how to set up the CUDA development environment by updating `PATH` and `LD_LIBRARY_PATH`.
2. Download NVIDIA cuDNN V7.0.5 for CUDA 9.1 from NVIDIA cuDNN. (Registration in the NVIDIA Accelerated Computing Developer Program is required.) Ensure that you download cuDNN V7.0.5 Library for Linux (Power8/Power9).
3. Install the cuDNN V7.0 packages:

```
$ sudo tar -C /usr/local --no-same-owner -xzvf
cudnn-9.1-linux-ppc64le-v7.0.5.tgz
```

Anaconda

A number of the DL frameworks require Anaconda. Anaconda is a platform-neutral data science distribution with a collection of 1,000+ open source packages with no-charge community support.

Download and install Anaconda. The installation requires input for a license agreement, the installation location (the default is `$HOME/anaconda2`), and permission to modify the `PATH` environment variable (by way of `.bashrc`). Use the following commands:

```
$ wget https://repo.continuum.io/archive/Anaconda2-5.0.0-Linux-ppc64le.sh
$ bash Anaconda2-5.0.0-Linux-ppc64le.sh
$ source ~/.bashrc
```

4.4.3 Installing the deep learning frameworks

This section describes how to install the DL frameworks.

IBM Spectrum MPI installation

Complete the following steps:

1. Download IBM Spectrum MPI from the ESP download site.
2. Install the RPMs:

   ```
   $ sudo rpm -ihv ibm_smpi_lic_s-10.02.00*.ppc64le.rpm
   ibm_smpi-10.02.00*.ppc64le.rpm
   ```

Software repository setup

IBM TensorFlow ESP for Power AC922 Deep Learning packages are distributed in an RPM file and are available from the ESP download site. Installing the RPM creates an installation repository on the local machine. To install the repository package, run the following command:

```
$ sudo rpm -ihv mldl-repo-local*.rpm
```

Installing all frameworks together

All the DL frameworks can be installed together by using the `power-mldl` meta-package:

```
$ sudo yum install power-mldl-esp
```

Installing frameworks individually

The DL frameworks can be installed individually if you prefer. The framework packages are:

- `tensorflow`: Google TensorFlow V1.4.0
- `tensorboard`: Web Applications for Inspecting Tensorflow Runs and Graphs V0.4.0rc3
- `ddl-tensorflow`: DDL custom operator for TensorFlow

Each can be installed by running the following command:

```
$ sudo yum install <framework>-cuda9.1
```

Accepting the license agreements

Read the license agreements and accept the terms and conditions before using IBM Spectrum MPI or any of the frameworks by running the following commands:

```
$ sudo IBM_SPECTRUM_MPI_LICENSE_ACCEPT=no
/opt/ibm/spectrum_mpi/lap_se/bin/accept_spectrum_mpi_license.sh
$ sudo /opt/DL/license/bin/accept-powerai-license.sh
```

After reading the license agreements, future installations may be automated to silently accept the license agreements by using the following commands:

```
$ sudo IBM_SPECTRUM_MPI_LICENSE_ACCEPT=yes
/opt/ibm/spectrum_mpi/lap_se/bin/accept_spectrum_mpi_license.sh
$ sudo IBM_POWERAI_LICENSE_ACCEPT=yes
/opt/DL/license/bin/accept-powerai-license.sh
```

4.4.4 Tuning recommendations

The recommended settings for optimal DL performance on the Power AC922 are as follows:

- Enable Performance Governor:

  ```
  $ sudo yum install kernel-tools
  $ sudo cpupower -c all frequency-set -g performance
  ```

- Enable the GPU persistence mode:

  ```
  $ sudo systemctl enable nvidia-persistenced
  $ sudo systemctl start nvidia-persistenced
  ```

- For TensorFlow, set the SMT mode:

  ```
  $ sudo ppc64_cpu --smt=4
  ```

- For TensorFlow with DDL, set the SMT mode:

  ```
  $ sudo ppc64_cpu --smt=2
  ```

4.4.5 Getting started with machine learning and deep learning frameworks

This section describes how to start using MLDL frameworks.

General setup

Most of the IBM PowerAI packages install outside the normal system search paths (to `/opt/DL/...`), so each framework package provides a shell script to simplify environmental setup (for example, `PATH, LD_LIBRARY_PATH, PYTHONPATH`).

You should update your shell rc file (for example, `.bashrc`) to source the setup scripts. For example:

```
$ source /opt/DL/<framework>/bin/<framework>-activate
```

Each framework also provides a test script to verify basic functions:

```
$ <framework>-test
```

Dependencies

Many IBM PowerAI frameworks (for example, TensorFlow and TensorBoard) have their dependencies satisfied by Anaconda packages. These dependencies are validated by the `<framework>-activate` script to ensure that they are installed; if not, the script fails.

For these frameworks, the `/opt/DL/<framework>/bin/install_dependencies` script must be run before activation to install the required packages. For example:

```
$ source /opt/DL/tensorflow/bin/tensorflow-activate
Missing dependencies ['backports.weakref', 'mock', 'protobuf']
Run "/opt/DL/tensorflow/bin/install_dependencies" to resolve this problem.

$ /opt/DL/tensorflow/bin/install_dependencies
Fetching package metadata ...........
Solving package specifications: .

Package plan for installation in environment /home/rhel/anaconda2:

The following NEW packages will be INSTALLED:

    backports.weakref: 1.0rc1-py27_0
    libprotobuf:       3.4.0-hd26fab5_0
    mock:              2.0.0-py27_0
    pbr:               1.10.0-py27_0
    protobuf:          3.4.0-py27h7448ec6_0

Proceed ([y]/n)? y

libprotobuf-3. 100% |###############################| Time: 0:00:02   2.04 MB/s
backports.weak 100% |###############################| Time: 0:00:00  12.83 MB/s
protobuf-3.4.0 100% |###############################| Time: 0:00:00   2.20 MB/s
pbr-1.10.0-py2 100% |###############################| Time: 0:00:00   3.35 MB/s
mock-2.0.0-py2 100% |###############################| Time: 0:00:00   3.26 MB/s

$ source /opt/DL/tensorflow/bin/tensorflow-activate
$
```

Getting started with TensorFlow

The TensorFlow home page has a variety of information, including tutorials, how tos, and a Getting Started guide.

For more tutorials and examples, see the following resources:

- TensorFlow Tutorials
- TensorFlow Tutorial and Examples for Beginners with Latest APIs

Distributed Deep Learning Custom Operator for TensorFlow

IBM TensorFlow ESP for Power AC922 includes a technology preview of the IBM PowerAI DDL custom operator for TensorFlow. The DDL custom operator uses IBM Spectrum MPI and NCCL to provide high-speed communications for distributed TensorFlow.

The DDL custom operator can be found in the `ddl-tensorflow` package. For more information about DDL and about the TensorFlow operator, see the following documentation:

```
/opt/DL/ddl-doc/doc/README.md
/opt/DL/ddl-tensorflow/doc/README.md
/opt/DL/ddl-tensorflow/doc/README-API.md
```

The DDL TensorFlow operator makes it easy to enable Slim-style models for distribution. The package includes examples of Slim models that are enabled with DDL:

```
$ source /opt/DL/ddl-tensorflow/bin/ddl-tensorflow-activate
$ ddl-tensorflow-install-samples <somedir>
```

These examples are based on a specific commit of the TensorFlow models repository with a small adjustment. If you prefer to work from an upstream clone, rather than the packaged examples, run the following commands:

```
$ git clone https://github.com/tensorflow/models.git
$ cd models
$ git checkout 11883ec6461afe961def44221486053a59f90a1b
$ git revert fc7342bf047ec5fc7a707202adaf108661bd373d
$ cp /opt/DL/ddl-tensorflow/examples/slim/train_image_classifier.py slim/
```

Additional TensorFlow features

The IBM PowerAI TensorFlow packages include TensorBoard.

TensorFlow V1.4.0 package includes support for additional features:

- HDFS
- NCCL
- Experimental XLA JIT compilation

4.4.6 Uninstalling machine learning and deep learning frameworks

The MLDL framework packages can be uninstalled individually the same way they were installed. To uninstall all MLDL packages and the repository that was used to install them, run the following command:

```
$ sudo yum remove powerai-license
$ sudo yum remove mldl-repo-local-esp
```

5

Working with data and creating models in IBM PowerAI

This chapter illustrates practical examples of neural networks, how to implement them, and the advantages of using IBM PowerAI. It describes why you must understand the context of your project and learn that model definition is just one piece of the overall project.

This chapter describes understanding a project in detail, from a business perspective to the technical levels. This context enables the technical team to verify and confirm the project feasibility before any initial decisions are implemented.

The chapter then describes data preparation. After the project goals and direction are finalized, this stage is the next step. It is an initial critical project phase that if incorrectly done directly impacts the model's accuracy. This chapter highlights many aspects and describes how a different model can change the method that used for data preparation.

The final sections of the chapter illustrate example coded neural network models and the process of data preparation and how that process provides the foundation to grow and advance the model.

This chapter contains the following topics:

5.1 Knowing your requirements and data

When a department in a company decides to implement a cognitive solution by using machine learning (ML) capabilities, it is a common oversight to not think about the data they have: what kind of data, age of that data, governance, and sensitivity. Also, it is tempting for the technical team to start deciding about algorithms without taking time to appreciate what the data is.

Although it is possible to implement a cognitive system without understanding your data, the result might not be the most accurate or efficient. The first critical task is to understand and agree on the actual business needs and requirements. Employee design thinking sessions can assist with this part of the process. All parties that are involved in the project can highlight and discuss requirements, priorities, concerns, and expectations.

With the business requirements and expectations discussed and agreed upon, the next phase focuses on the data itself. There must be a clear understanding across the involved teams to appreciate the data that is used. People must have agreed expectations of what information the data provides.

This data analysis phase should be done during the discovery and viability of the project and not after committing to the project. Making decisions and dependencies based on assumptions of the data can be an inhibitor to a successful project.

An example of a situation where data can be a barrier for the project is in a case where the system must be able to predict whether an operation is fraudulent and expectations are made that historic data exists. However, when the data is checked, the team finds that there are no connections between the historic data, making it impossible to identify whether a specific transaction is fraudulent. The data that they have is simple field, such as amount, name, and address. There are no *fraud tables* registering old fraud or indications that characterize fraud.

If there is such a situation, it does not mean that the project can never happen, but it might require more phases or even a project on the existing systems to generate more detailed data and start registering fraud data in the expected format. It might be possible after some data generation to use that data to feed the ML algorithm.

After analysis of the data concludes that the source meets requirements and can be used to fulfill expectations, the next step is the ML algorithm definition. This chapter does not go into the details of which kind of ML algorithm is best for each case or which neural network architecture fits better for such cases. Those topics are described in 1.2, “Neural networks overview” on page 5.

Regarding neural networks, there is typically an architecture that is focused on specific areas. Testing with more than one option can help your decision-making process.

5.2 Why is it so important to prepare your data

After the technical team has a clear objective about what to predict or classify (verified data, data in one place, and the neural network architecture (or architectures) to use), it is time to prepare your data to feed your neural network.

To prepare the data, think of two phases: the first phase is where all the transformation and processing is applied to the data set to extract and load it into an accessible environment or proper database. This process is a classical extract, transform, and load (ETL) process that is required to isolate the data with which you want to work.

The second phase is about making the transformed data ready to be imported by the neural network. This process usually requires considerable processing, and it is normal to have some data expansion because you must represent your data in different formats. It is beyond the scope of this section to describe specific details, so this phase's required approach depends on the format of data (images, sound, text, or numbers) and the architecture of the selected neural network.

A neural network is more efficient when it uses numbers as input because it is a mathematical model. In this context, it is necessary to convert your data into numbers while keeping it all organized, structured, and in the format that the neural network architecture expects, which is usually a vector array.

For example, if your data was an image, you might convert it to grayscale, then extract its matrix and convert it into an array. This process must be done on all images so that each row of your data represents an image. In addition, it is necessary to create the labels so that the neural network can learn from the data. These labels must be converted into numbers and stored as a hot vector so that they can be fed into the neural network.

This approach is a high-level path of several options to handle the data and get it ready. The following sections show a couple of scenarios to prepare the data and create a neural network model with the data.

5.3 Sentiment analysis by using TensorFlow on IBM PowerAI

This section illustrates the feed-forward neural network (FFNN) model that identifies whether a sentence is negative or positive. This sample uses TensorFlow on IBM PowerAI to create and train the model.

The model was created by using TensorFlow.

5.3.1 Example data set

The data set that is used is from a Stanford project called Sentiment140. This is also where the data set can be downloaded from, and it contains 1.6 million rows with positive and negative sentences.

Note: At the time of writing, the project website mentioned that there can be negative, positive, and neutral sentences. After we analyzed the data set, we found only positive and negative sentences, so we used only those two options to create and train our model.

All the information that is related to the data set format and the code that was created was valid at the time of writing. If you attempt to work through the same example, you must make changes if the data set layout is later updated.

The data set contains two CSV files with six fields each, one for training data and a smaller one for testing. Both files are structured according to the following format (taken from the project website):

0	The polarity of the tweet (0 = negative, 2 = neutral, 4 = positive).
1	The ID of the tweet (2087).
2	The date of the tweet (Sat May 16 23:58:44 Coordinated Universal Time 2009).

3	The query (lyx). If there is no query, then this value is `NO_QUERY`.
4	The user that tweeted (robotickilldozr).
5	The text of the tweet (Lyx is cool).

All the sentences in the data set are extracted from Twitter, and their sentiment has not been identified by a human who later annotated the file with the correct label. A code was used to identify sentences with any positive emoticons, and they are assumed to be positive and negative emoticons with negative sentiment. You can use your own sentence data set and label them as needed. For more information about the data set and the project, see the Sentiment140 Project.

5.3.2 How the code is structured

The code is divided in to three files: One is about preparing the data, the second is about creating the FFNN model, and the last file is using the created and trained model. For more information, see Figure 5-1.

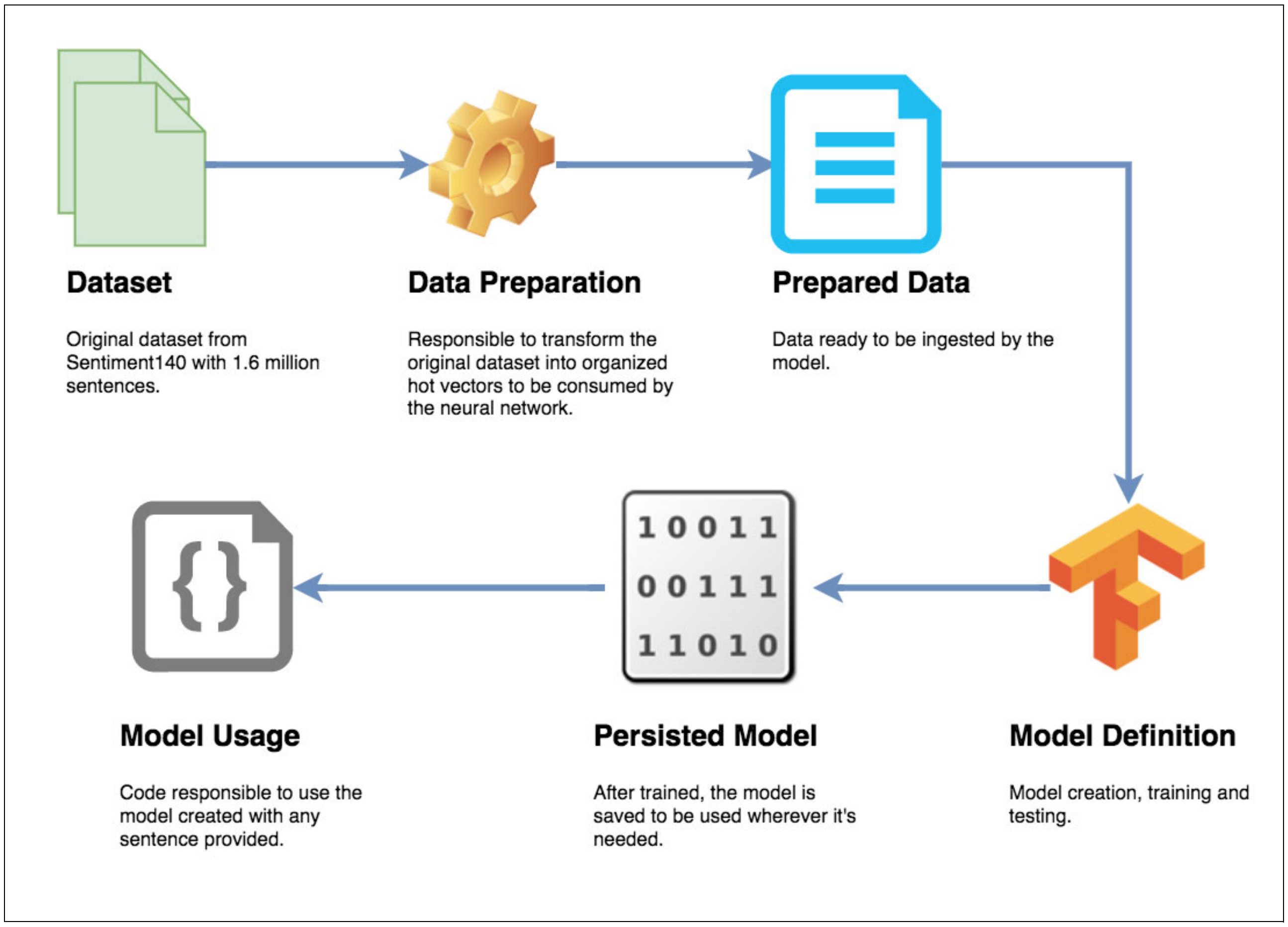

Dataset
Original dataset from Sentiment140 with 1.6 million sentences.
Data Preparation
Responsible to transform the original dataset into organized hot vectors to be consumed by the neural network.
Prepared Data
Data ready to be ingested by the model.
Model Definition
Model creation, training and testing.
10011
00111
11010
Persisted Model
After trained, the model is saved to be used wherever it's needed.
Model Usage
Code responsible to use the model created with any sentence provided.

Figure 5-1 How the code is structured

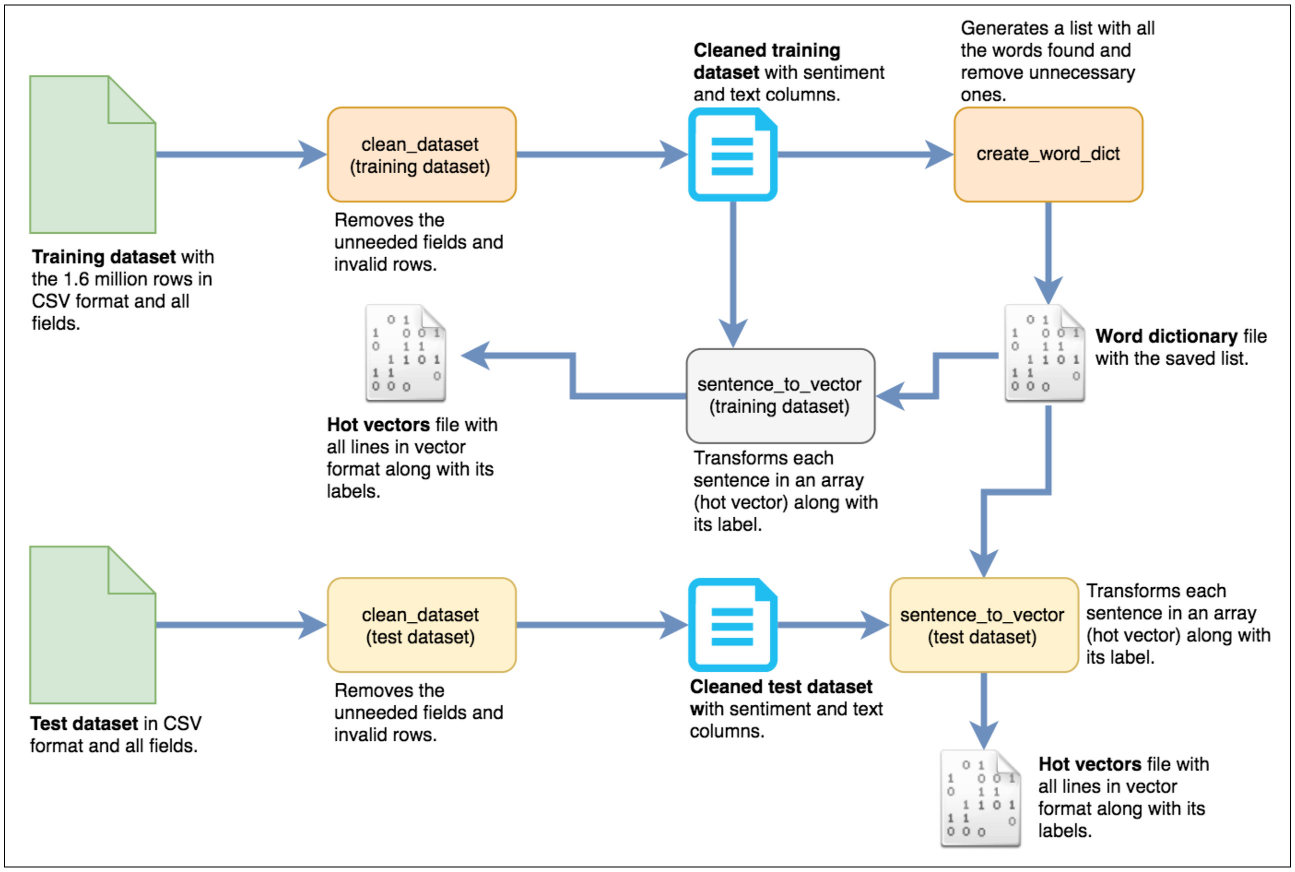
Considering that you are dealing with 1.6 million rows of data, it is important to use persistence to ensure that all data is not kept in memory, and in case of failure that there is no need to start the process from the beginning.

The following sections describe each one of the steps and show some snippets of code.

5.3.3 Data preparation

This phase ensures that the data is transformed and at the end of the flow is in the proper format that is expected by the FFNN.

Figure 5-2 shows the data preparation phase as a general view to describe the code details.



Training dataset with the 1.6 million rows in CSV format and all fields.
clean_dataset (training dataset)
Removes the unneeded fields and invalid rows.
Cleaned training dataset with sentiment and text columns.
Generates a list with all the words found and remove unnecessary ones.
create_word_dict
Word dictionary file with the saved list.
sentence_to_vector (training dataset)
Transforms each sentence in an array (hot vector) along with its label.
Hot vectors file with all lines in vector format along with its labels.
Test dataset in CSV format and all fields.
clean_dataset (test dataset)
Removes the unneeded fields and invalid rows.
Cleaned test dataset with sentiment and text columns.
sentence_to_vector (test dataset)
Transforms each sentence in an array (hot vector) along with its label.
Hot vectors file with all lines in vector format along with its labels.

Figure 5-2 Data preparation flow

Cleaning the data set

The first thing that you must do is to remove the unnecessary columns, such as the Twitter ID, date, and others, and keep only the sentiment and the Twitter text. To do this task, use the **`clean_dataset`** function, which is shown in Example 5-1.

Example 5-1 Functions that are responsible for cleaning the original data set

```
def shuffler(input_ds, output_ds):
    df_source = pd.read_csv(input_ds, '<SP>', error_bad_lines=False)
    df_shuffled = df_source.iloc[np.random.permutation(len(df_source))]
    df_shuffled.to_csv(output_ds, 'µ', index=False)

def clean_dataset(ds, ods):
    with open(ds, 'r', 30000, 'latin-1') as raw_ds:
        with open('tempds.csv', 'w', 20000) as cleaned_ds:
            for line in raw_ds:
                result = re.search('^"(\d)",.*,"(.*)"$', line)
                new_line = result.group(1) + '<SP>' + result.group(2) + '\n'
                cleaned_ds.write(new_line)
        shuffler('tempds.csv', ods)
        os.remove('tempds.csv')
    print("data set cleanup done”)
```

The function expects a file as input and generates a CSV file that is separated by a *<SP>* string to avoid character conflicts. Even though the data set is a CSV file, there are commas in the Twitter text field, so just using the split function does not work, so you must use a regular expression to get what was needed.

The **clean_dataset** function goes through every line of the data set and applies the regular expression by writing the new line into a temporary file. The last step is to shuffle the generated data set to ensure that the neural network is trained properly. For this example, we use the Pandas library and then write to the output file that is provided for the **ods** parameter.

This approach is used to avoid too much data being kept in memory, especially for large data sets. A buffer of 30000 bytes is defined and the file is read in chunks, processed, and written to the target file. For the shuffle process, the Pandas library is designed to handle large files.

Generating the dictionary of words

With the cleaned data set generated, the next step is to create the dictionary of words (known as a *lexicon*). In this example, we used is the **create_word_dict** function, as shown in Example 5-2.

Example 5-2 Code that is used to create the dictionary of words

```
def create_word_dict(source_ds):
    word_dict = []
    with open(source_ds, 'r', 30000, 'latin-1') as ds:
        for line in ds:
            text = line.split('µ')[1]
            words = word_tokenize(text.lower())
            lemm_words = [lemm.lemmatize(w) for w in words]
            word_dict += list(lemm_words)
        word_count = Counter(word_dict)
    cleaned_word_dict =
    [word for word in word_count if 1000 > word_count[word] > 60]
    dict_size = len(cleaned_word_dict)

    print("Word dictionary size: {}".format(dict_size))
    with open('word_dict.pickle', 'wb') as wd:
        pickle.dump(cleaned_word_dict, wd)

    print("Word dictionary generated and saved")
    return dict_size
```

In this phase, we go sentence by sentence, extracting all words from it by using the **word_tokenize** function from the NLTK library, and then passing each one of the words through either stemming or lemmatization, which removes the words that are not needed and then generates a new list with one occurrence for each word. This is necessary to generate the sentence vector for each one of the sentences in the next step of the flow.

Steaming and lemmatization: Both processes focus on word standardization and transforming those words into a common word base. Our example neural network expects numeric values as input, and each different word increases the size of the vector that is generated by one. For example, if we have the words *focus*, *focuses*, *focused*, and *focusing*, all of them have the same general meaning and there is no relevance about the way each is being used.

So, each of the processes (steaming and lemmatization) achieves this task in a different way. Stemming usually works by chopping off the end of the words to reach the same *default word*. Lemmatization uses a more robust set of rules to parse a word by using vocabulary and morphological analysis of that word to generate a new default word, which is usually the radical of that word.

Which processor to use depends on your needs and your processing power. Stemming is less processor-intensive, but generates less accurate results. In contrast, lemmatization requires more processing power, but provides more accurate results.

For more information, see Stanford's Stemming and Lemmatization.

The **`create_word_dict`** function starts by reading the CSV file that is generated in 5.3.3, "Data preparation" on page 125 and split the lines to get only the sentences. The sentences go through the **`word_tokenizer`** function from the NLTK library, and generate a list of words to which the lemmatize function is applied.

With all the words in their standard form, count their occurrences in the data set to remove the ones that occur too much (usually some words such as *a*, which have no useful meaning in a sentiment analysis) and the ones with small occurrences, which means that they make no difference in the model. In this example, we use the Counter function from the collection library, which generates a dictionary with each word as a key and its respective count as a value. Then, we save a list with one occurrence of each word as our dictionary of words into the `word_dict.pickle` file as a list.

Preparing the sentences

The last phase of the process is to read each sentence and transform it into a numeric vector. In this process, each sentence becomes a vector of zeros with the same size as the dictionary of words. This vector contains the number of each word in the dictionary of words. This number is in the same index position as the word is in the dictionary of words vector.

For example, assume that the data set contains two sentences:

- I want to drink water with you.
- They like to drink coffee with me at home.

From the data set, we create a dictionary of words (do not apply any kind of process or exclude words for simplicity). The result can be the following list, assuming that you start from the first sentence:

```
[I, want, to, drink, water, with, you, they, like, coffee, me, at, home]
```

Next, transform the first sentence into a hot vector. Create a list (or vector) with the same size as the word dictionary, in this case a list of size 13 filled with zeros:

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Then, iterate through the sentence word by word and check whether it exists in the dictionary of words. If so, add 1 to the new sentence vector in the same index position it is in the dictionary list.

The letter *I* exists in the dictionary and is on index 0, so add 1 to the sentence vector in position 0:

```
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Continue repeating the process until you reach the end of the sentence and have then processed the second sentence. By the end of the process, you have the following two vectors:

- Sentence 1: `[1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]`
- Sentence 2: `[0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1]`

The code reads the sentence training data set and the word dictionary file as input, takes the sentences and passes them through the word tokenization function, lemmatizes the words, and starts creating the vector. The code also gets sentiment labels and transforms them into a list of size two because there are positive and negative sentiments. The sentence vector and sentiment vector are put into a list and saved into a binary file to be read by the model in the following step.

Example 5-3 shows the **`sentence_to_vector`** function.

Example 5-3 Code that is used to transform sentences into vectors

```
def sentence_to_vector(word_dict_file, cleaned_ds, output_file):
    with open(cleaned_ds, 'r', 30000, 'latin-1') as ds:
        with open(word_dict_file, 'rb') as wd:
            word_dict = pickle.load(wd)
            num_lines = 0
            with open(output_file, 'wb') as hv:
                for line in ds:
                    hot_vector = np.zeros(len(word_dict))
                    if line.count('µ') == 1:
                        sentiment, text = line.split('µ')
                        words = word_tokenize(text.lower())
                        lemm_words = [lemm.lemmatize(w) for w in words]
                        for word in lemm_words:
                            if word in word_dict:
                                hot_vector[word_dict.index(word)] += 1
                        hot_vector = list(hot_vector)
                        clean_sentiment = re.search('.*(\d).*', sentiment)

                        if int(clean_sentiment.group(1)) == 0:
                            sentiment = [1, 0]
                        else:
                            sentiment = [0, 1]

                        num_lines += 1
                        pickle.dump([hot_vector, sentiment], hv)
                print('Hot vectors file generated with {}
                       lines'.format(num_lines))
    return num_lines
```

Some considerations

After finishing the training data set process, the test data set goes through the same process so that it has the same format and can be used to check the model's accuracy. As shown in Example 5-3 on page 128, the test data set must also use the same dictionary of words that is generated by the training data set.

Regarding the data set size, this scenario uses Python and some libraries to do the data handling job and some strategies, such as using files to manage a larger data set in a more efficient way. If you must scale and handle larger data sets, think about using a distributed framework over a distributed architecture to work on all the data to get it ready for a neural network. You can implement a distributed solution by using IBM Power Systems servers with IBM Spectrum Scale to deliver a reliable and scalable environment. This scenario does not go into these details because they are beyond the scope of this book. However, this book describes how TensorFlow can take advantage of distributed processing by using several graphical processing units (GPUs).

5.3.4 Model creation

Now, we must build the example TensorFlow-based model. In the data preparation phase, we saved some information about the dictionary of word's list size and how many lines there are in the vector files that were generated. That information is going to be used in this phase. Example 5-4 shows the function that is responsible for creating the neural network architecture.

Example 5-4 Creating a neural network architecture

```
def ff_neural_net(input_data):
    neurons_hl1 = 1500
    neurons_hl2 = 1500
    neurons_hl3 = 1500

    output_neurons = 2

    l1_weight = tf.Variable(tf.random_normal([line_sizes['dict'], neurons_hl1]), name='w1')
    l1_bias = tf.Variable(tf.random_normal([neurons_hl1]), name='b1')

    l2_weight = tf.Variable(tf.random_normal([neurons_hl1, neurons_hl2]), name='w2')
    l2_bias = tf.Variable(tf.random_normal([neurons_hl2]), name='b2')

    l3_weight = tf.Variable(tf.random_normal([neurons_hl2, neurons_hl3]), name='w3')
    l3_bias = tf.Variable(tf.random_normal([neurons_hl3]), name='b3')

    output_weight = tf.Variable(tf.random_normal([neurons_hl3, output_neurons]), name='wo')
    output_bias = tf.Variable(tf.random_normal([output_neurons]), name='bo')

    l1 = tf.add(tf.matmul(input_data, l1_weight), l1_bias)
    l1 = tf.nn.relu(l1)

    l2 = tf.add(tf.matmul(l1, l2_weight), l2_bias)
    l2 = tf.nn.relu(l2)

    l3 = tf.add(tf.matmul(l2, l3_weight), l3_bias)
    l3 = tf.nn.relu(l3)

    output = tf.matmul(l3, output_weight) + output_bias

    return output
```

This section does not provide details about TensorFlow because this framework was introduced in "TensorFlow" on page 19.

The **`ff_neural_net`** function receives the data as input (a TensorFlow placeholder) and then start defining some characteristics of the neural network. Define the number of neurons and layers for the architecture.

When talking about how many hidden layers and how many neurons each layer must have, the answer is never an exact one, and most of the time it is defined empirically through testing with different options. This case uses an FFNN with two hidden layers. The input layer and the hidden layers have 1500 neurons each, and an output layer with two neurons because there are two values to classify (positive and negative).

Figure 5-3 shows the neural network for this scenario.

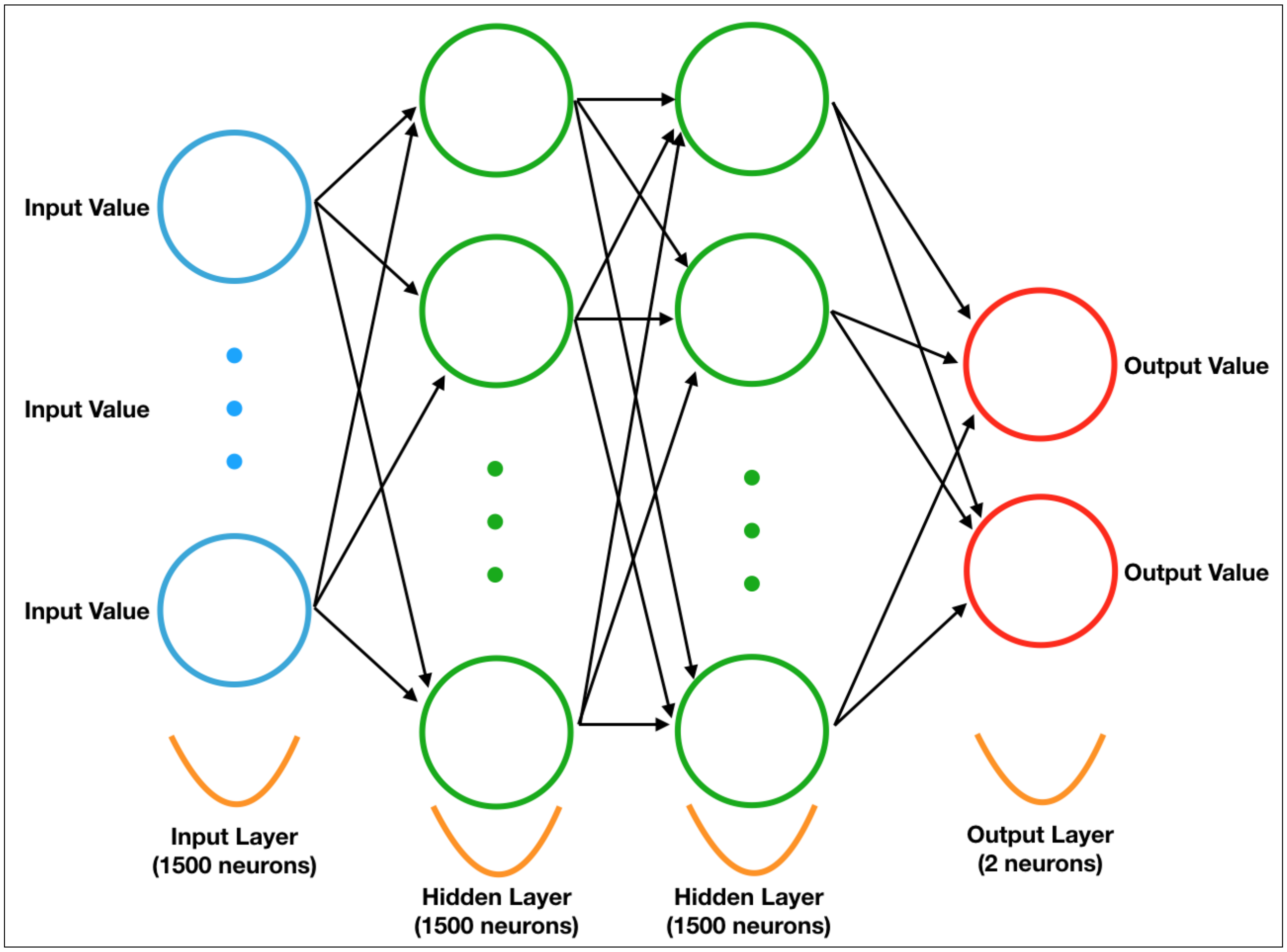


Figure 5-3 Architecture of the feed forward neural network being used

Define the weights and biases for each one of the layers by creating TensorFlow variables and specifying their sizes. Each variable is a matrix, and the shape of the first matrix is the size of the sentence vector by how many neurons the first layer has. The second layer has a matrix with the shape of the first layer size by the second layer size, the same for the third layer and the output layer matrix has the shape of the third layer size by the output layer size. The column size of the matrix in the first layer must be the same as the row size of the matrix in the second layer because the neural network performs several matrix multiplication operations.

This is all that you must do to create the feed-forward architecture. TensorFlow knows to adjust the variables that are created in its graph through the training iterations.

The function starts by calling the neural network model and creates a saver object so that the graph and its variables can be saved later.

Example 5-5 shows the function that performs all the training of the model.

Example 5-5 Code that is used to optimize and train the model

```
def training(in_placeholder):
    nn_output = ff_neural_net(in_placeholder)
    saver = tf.train.Saver()
    # We are using cross entropy to calculate the cost
    cost =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=nn_output,
labels=y))

    optimizer = tf.train.AdamOptimizer(learning_rate=0.001).minimize(cost)

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for epoch in range(num_epochs):
            epoch_loss = 0
            buffer_train = []
            buffer_label = []
            with open('train_hot_vectors.pickle', 'rb') as train_hot_vec:
                for i in range(line_sizes['train']):
                    hot_vector_line = pickle.load(train_hot_vec)
                    buffer_train.append(hot_vector_line[0])
                    buffer_label.append(hot_vector_line[1])

                    if len(buffer_train) >= batch_size:
                        _, cost_iter = sess.run([optimizer, cost],
                        feed_dict={in_placeholder: buffer_train, y: buffer_label})
                        epoch_loss += cost_iter
                        buffer_train = []
                        buffer_label = []
            print('Epoch {} completed. Total loss: {}'
                 .format(epoch+1, epoch_loss))

        correct = tf.equal(tf.argmax(nn_output, 1), tf.argmax(y, 1))
        accuracy = tf.reduce_mean(tf.cast(correct, 'float'))

        with open('test_hot_vectors.pickle', 'rb') as train_hot_vec:
            buffer_test = []
            buffer_test_label = []
            for i in range(line_sizes['test']):
                test_hot_vector_line = pickle.load(train_hot_vec)
                buffer_test.append(test_hot_vector_line[0])
                buffer_test_label.append(test_hot_vector_line[1])

        # the accuracy is the percentage of hits
        print('Accuracy using test data set: {}'
      .format(accuracy.eval({in_placeholder: buffer_test, y: buffer_test_label})))
        saver.save(sess, "model.ckpt")
```

Calculate the cost or error that is generated between the model output and the correct labels. During the calculation of the cost (error), an optimizer that is called Adam is applied, which is a stochastic gradient descent (SGD) approach that is used to reduce the cost by updating all the weights and biases in the neural network by using a back-propagation algorithm.

So far, this process just builds the TensorFlow objects and links them with each other. Now, the process creates a TensorFlow session where the graph is run and imports data. Before the **session.run** function is called, the process reads the data from the vectorized sentences, and every time the temporary buffer that is defined reaches the batch size, which is also defined by us, the code calls TensorFlow and runs the optimizer, which then runs the whole model.

We also have a buffer list for the label because we must provide the sentence with its respective label so that the network is trained correctly. We also run the cost function to know the total cost that is generated for that specific epoch.

The training phase happens several times, which are the number of times that is defined in the *epochs* variable. The code is using 10, although you can change it and check whether you achieve better results. Basically, the training data set is submitted in batches to the TensorFlow model, and the error is calculated and then adjusted by the Adam optimizer. The process repeats for *epoch* times.

After the training loop is done, add to the graph the correct tensor, where it uses the **tf.argmax** function, which is the index position with the maximum value from the network output and the correct label. The **tf.equal** function is 1 if both are the same or 0 if they do not match. Using the correct tensor, the accuracy is calculated as the mean of the correct tensor.

Now, we read the test data set file that is vectorized through the same batch process, this time without the epochs. We submit the testing data to be evaluated against our model by using the **eval** function and passing the data set and its label to the placeholders. When completed, the model is saved by the **saver.save** function.

5.3.5 Using the model

With the trained model saved and ready to use, a function that is called **get_sentiment** is created, which receives a sentence as input and prints whether it is positive or negative (Example 5-6).

Example 5-6 Function that is used to use the model

```
def get_sentiment(input_data):
    tf.reset_default_graph()
    pl = tf.placeholder('float')
    nn_output = ff_neural_net(pl)
    saver = tf.train.Saver()
    with open('word_dict.pickle', 'rb') as f:
        word_dict = pickle.load(f)

    with tf.Session() as sess:
        saver.restore(sess, "model.ckpt")
        words = word_tokenize(input_data.lower())
        lemm_words = [lemm.lemmatize(w) for w in words]
        hot_vector = np.zeros(len(word_dict))

        for word in lemm_words:
            if word.lower() in word_dict:
                index_value = word_dict.index(word.lower())
                hot_vector[index_value] += 1

        hot_vector = np.array(list(hot_vector))
        result = (sess.run(tf.argmax(nn_output.eval(
                                    feed_dict={pl: [hot_vector]}), 1)))
        if result[0] == 0:
            print('Negative:', input_data)
        elif result[0] == 1:
            print('Positive:', input_data)
```

Start the function by defining the placeholder, which receives our sentence, loads our model, and creates a saver object to load our trained model later. We also open the dictionary of words file.

We then create a TensorFlow session and restore the saved model. Restoring is basically reading the values from the variables that are saved in the `model.ckpt` file and assigning them to the variables that are created when running the model with the **`ff_neural_net`** function.

Using `sess.run`, we run our model by passing **`hot_vector`** and then by using the **`tf.argmax`** function, and we get the neural network answer. We know that `[1, 0]` is positive and `[0, 1]` is negative, so if **`argmax`** returns `0`, meaning that the index position 0 is the biggest and it is a positive sentence; if it is 1, then it is a negative sentence. We do an IF to make the return human-readable.

5.3.6 Running the code

To run the code by using IBM PowerAI, source the framework that you want to use, which loads the framework environment with the libraries that are required. To do so, go to `/opt/DL`, where the frameworks installation is, and open the one that you want to use. In our case, we want to use the `tensorflow/bin`. From the directory (`/opt/DL/tensorflow/bin`), run **`source tensorflow-activate`**.

To train your model, you must run the **`training`** function and pass a TensorFlow placeholder to it. To use your model, you must run the **`get_sentiment`** function and provide the sentence that you want to check.

While you train your model, you can look at how your GPUs are being used by running the **`nvidia-smi`** command on your terminal window.

If you want to do some testing or want to make changes to the model, you can experiment with a smaller data set. For example, we created a function that is called **`smaller_dataset_gen`** that can be used to randomly extract rows out of the large data set and create a smaller one. All that you need to do is run the function and provide the original data set, how many rows it has (1600000 rows if you are using the Sentiment140 one), the name of the target data set, and how many rows you want it to have.

The complete code is available at GitHub. For more information, see Appendix A, "Sentiment analysis code" on page 241.

5.4 Word suggestions by using long and short term memory on TensorFlow

To make a word suggestion to speed up a user typing process or even generate text that is based on some other existing text, it important to have different applications where cognitive systems must be the rule and not the exception. This section describes a word suggestion implementation by using a long and short term memory (LSTM) recurrent neural network (RNN) so that you can understand how it works with TensorFlow and take advantage of IBM PowerAI.

This is a straightforward implementation focusing on clarity so that you can understand the details and build from it. Think of this implementation as a head start.

Our example is all about training our model into a data set of phrases; when using it, we can start typing words to create a phrase and receive predictive suggestions from our model on what the next word can be so we do not have to type it all.

To do so, we use an RNN because the order or sequence the words are in matter to make a good suggestion. We implement a specific RNN called LSTM because a simple RNN cannot keep data from *old* iterations; they can remember only *recent* data, LSTM remember what is important and *forget* what is not. For more information, see 1.2.3, “Types of neural networks and their usage” on page 8.

5.4.1 Our data set

We are using a simple and small data set with about 100 created questions that simulate users asking how to do something within their company system, such as how to reset their password, how to get access to the internet, and where to find some information.

The data is already in the correct format, but in a production environment you must obtain this data from databases and apply transformations to convert them to the required format.

5.4.2 Overall structure of the code

The overall structure of the code is about the same as the one that is used by the sentiment analysis in 5.3, “Sentiment analysis by using TensorFlow on IBM PowerAI” on page 123, so the same diagram is used (Figure 5-1 on page 124). The diagram represents how our code is structured.

There is a data preparation phase to create our dictionaries and put the phrases into the correct Python objects so that they are ready to be used by the neural network. Then, in the second phase we define our LSTM neural network model, train it, and save it so that we can get to the third phase where the model can be used.

5.4.3 Data preparation

In this phase, we create our dictionary of words so that we know each different word that occurs in our data set. We are not excluding any words because every word is important for the system.

In this phase, we transform each phrase from the data set into a list of words and save it into a pickle file so that we can process even larger data sets without having to keep all of them in memory.

Example 5-7 gives you a sample of the code.

Example 5-7 The create_word_list function

```
def create_word_list(source_ds):
    words_list = []
    num_lines = 0
    count = 0
    with open(source_ds, 'r', 20000, 'latin-1') as ds:
        with open('ds_to_list.pickle', 'wb', 10000) as ds_list:
            for line in ds:
                words = word_tokenize(line.lower())
                words_list += list(words)
                pickle.dump(words, ds_list)
                num_lines += 1

            word_count = set(words_list)
            word_list_final = {}
            for i in word_count:
                word_list_final[i] = count
                count += 1
            rev_word_dict = dict(zip(word_list_final.values(),
                                     word_list_final.keys()))

    list_size = len(word_list_final)

    print("Word dictionary size: {}".format(list_size))
    with open('word_dict.pickle', 'wb') as wd:
        pickle.dump(word_list_final, wd)

    with open('rev_word_dict.pickle', 'wb') as wd:
        pickle.dump(rev_word_dict, wd)

    print("Word dictionary generated and saved")
    return list_size, num_lines
```

The **`create_word_list`** function receives the data set and goes through it line by line. For each line, the functions transforms it into a list of words by using the **`word_tokenize`** function from the NLTK library, saves it into a file by using pickle, and adds it to a new list that is called `words_list`, which is then stored uniquely by using a set.

We create a dictionary and give each word a number from 0 - **`word_count`** size. Those numbers are used to feed the neural network. We must create a reverse word dictionary to make things easier to map when we have the word's number and must go back to the word. Basically a new dictionary is created by using the first dictionary's values as keys and its keys as values.

After this process, we write both dictionaries to files so they can be used in different modules. The word dictionary size and the original data set size are returned by the function so that they can be used for loop control in different modules.

5.4.4 Model creation

After the dictionary of words is created with its respective number for each word, and the data set is in the format of one list per phrase, with each item of the list being one word of the phrase, we create our model by using the LSTM neural network. The **lstm_rnn** function is where this definition takes place, as shown in Example 5-8.

Example 5-8 The lstm_rnn function

```
def lstm_rnn(tf_placeholder):

    output_weight = tf.Variable(tf.random_normal([num_neurons,
                                             files_details['list']]))
    output_biases = tf.Variable(tf.random_normal([files_details['list']]))

    tf_placeholder = tf.reshape(tf_placeholder, [-1, interval_size])

    tf_placeholder = tf.split(tf_placeholder, interval_size, 1)

    rnn_layers = rnn.MultiRNNCell([rnn.BasicLSTMCell(num_neurons),
                                   rnn.BasicLSTMCell(num_neurons)])

    outputs, states = rnn.static_rnn(rnn_layers, tf_placeholder, dtype=tf.float32)

    return tf.matmul(outputs[-1], output_weight) + output_biases
```

Complete the following steps:

1. Define the weights and biases for the output layer of the neural network. The weights are a matrix with all its weights randomly initialized, and the biases are an array that also is randomly initialized. They both are TensorFlow variables, so the framework knows that they must be updated and adjusted during the training phase.
2. Create a TensorFlow placeholder to receive the data set (in number format) and reshape it to conform to the format that is expected by the **rnn.static_rnn** function.
3. TensorFlow sets up the LSTM neural network by using ready to use functions.
4. Use the **rnn.MultiRNNCell** function to create an RNN with more than one hidden layer. This function receives the neurons for each layer, which in this case are LSTM neurons. The **BasicLSTMCell** function with the number of neurons must be used. The **static_rnn** function binds the whole model together with the placeholder to create the neural network. TensorFlow helps define a complete structure of the neural network.
5. Perform the output layer operations, which are multiplying the second hidden layer output by the output weights and sum of the biases. The last position of the outputs variable is used because we are interested in the last output from the RNN. An RNN generates one output after each iteration, so it generates an array as output.

5.4.5 Training

The training phase is similar to the phase that is applied to the sentiment analysis in 5.3, "Sentiment analysis by using TensorFlow on IBM PowerAI" on page 123. The difference is how and what is provided to the neural network. Example 5-9 goes through the code and shows its details.

Example 5-9 The training function

```
def training(in_placeholder):
    rnn_output = lstm_rnn(in_placeholder)
    saver = tf.train.Saver()

    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
                                             logits=rnn_output, labels=y))
    optimizer = tf.train.RMSPropOptimizer(learning_rate=0.001).minimize(cost)

    correct = tf.equal(tf.argmax(rnn_output, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

    init = tf.global_variables_initializer()

    with tf.Session() as session:
        session.run(init)
        step = 0
        start = 0
        end = interval_size
        acc_total = 0
        loss_total = 0
        counter = 0

        with open('word_dict.pickle', 'rb') as wd:
            word_dict = pickle.load(wd)
            while step < epochs:
                with open('ds_to_list.pickle', 'rb', 20000) as ds:
                    for _ in range(files_details['data set']):
                        line_list = pickle.load(ds)
                        line_size = len(line_list)
                        while end < line_size:
                            sequence = [[word_dict[i]] for i in
                                                  line_list[start:end]]
                            sequence = np.array(sequence)
                            sequence = np.reshape(sequence,
                                             [-1, interval_size, 1])
                            label = line_list[end]
                            label_hot_vector = np.zeros([files_details['list']])
                            label_hot_vector[word_dict[label]] = 1.0
                            label_hot_vector = np.reshape(label_hot_vector,
                                                  [1, -1])

                            start += 1
                            end += 1

                            _, acc, loss, rnn_predictions = session.run(
                                     [optimizer, accuracy, cost, rnn_output],
                                     feed_dict={in_placeholder: sequence,
                                     y: label_hot_vector})
                            counter += 1
                            acc_total += acc
                            loss_total += loss

                print('{}. Loss: {:.4f} and Accuracy: {:.2f}%'.format
                                      (step+1, loss_total / counter,
                                         (100 * acc_total) / counter))
                acc_total = 0
                loss_total = 0
                counter = 0

                start = 0
                end = interval_size

                step += 1
            saver.save(session, 'model.ckpt')
            print('Training completed')
```

1. The **`training`** function begins by concluding the TensorFlow graph creation. It first calculates the error, which is the difference of the correct label and what the model outputs. The error is calculated by using the **`softmax_cross_entropy_with_logits`** function. This generated error is then optimized by the **`RMSPropOptimizer`** function. RMSProp is a gradient descent optimization algorithm that is used to reduce the error and adjust the neural network's weights and biases. You can exchange this algorithm for any other to try it out. In our case, using RMSProp helped us reach high accuracy rates.
2. Still building the TensorFlow graph, we define the correct variable that receives an array (in this case, size 1) with value `1` if the network output is the same as the one that is provided by the label (y placeholder) or `0` if they are different. As with the previous step, the mean is calculated and assigned to the accuracy variable to be used to calculate the general accuracy later during the graph execution phase.
3. We then create a TensorFlow session to start the training process. The dictionary of words file is opened and we start looping through the epochs. The `epoch` is how many times we want to iterate through all our data set to train our model. There are different ways to define that, from establishing a specific number of times to checking the accuracy and run until the accuracy reaches a minimal variation rate.

   We are using a specific number of epochs, 60000, which can reach more than 90% accuracy. Of course, increasing the number of epochs and providing a better data set can improve the accuracy even more.

   Within the epochs loop, we start going through our data set, which is already in a list format, and for each phrase the code picks the first three words as the training data, converts them to the word's respective number, and reshapes it to conform to the neural network input. We also provide the label, which is the fourth word.
4. For the label, we transform the word into a hot vector with the same size as the dictionary of words. We use a hot vector to check which position has the biggest value, which is the strongest suggestion, and the rest of the successive suggestions.

   The process to create the hot vector is straightforward. After we have the array filled with zeros, it is necessary to add only one to the index position referring to the respective number of the word that is found in the dictionary of words (**`word_dict`**).
5. With the training data and label prepared, they are fed into the neural network for training. After all the epochs interactions are processed, the trained model is saved by using the **`save`** method from **`tf.trainer.Saver`**.

5.4.6 Using the model

To use the trained network model, create the **get_word** function, as shown in Example 5-10. This function is responsible for providing a CLI experience that prompts the user to type in at least three words, and based on the input calls the neural network model, gets its return and presents the three best options of words so the user can auto-complete the phrase being written.

Example 5-10 The get_word function

```
def get_word():
        with open('word_dict.pickle', 'rb') as wd:
            word_dict = pickle.load(wd)
        with open('rev_word_dict.pickle', 'rb') as rwd:
            rev_word_dict = pickle.load(rwd)
            in_placeholder = tf.placeholder('float', [None, interval_size, 1])
            rnn_output = lstm_rnn(in_placeholder)
            saver = tf.train.Saver()
            with tf.Session() as sess:
                sess.run(tf.global_variables_initializer())
                saver.restore(sess, 'model.ckpt')
                phrase_check = 'invalid'
                while phrase_check == 'invalid':
                    prompt = 'Type at least {} words: '.format(interval_size)
                    phrase = input(prompt)
                    phrase = phrase.strip()
                    words = word_tokenize(phrase.lower())
                    if len(words) < interval_size:
                        print('We need at least {} words!'.format(interval_size))
                    else:
                        words = words[-interval_size:]
                        phrase_check = 'valid'
                next_word = '-'
                word_dict_size = len(word_dict)
                phrase_to_num = []
                for i in words:
                    if i in word_dict.keys():
                        phrase_to_num.append(word_dict[i])
                    else:
                        phrase_to_num.append(word_dict_size)
                        word_dict_size += 1

                while next_word != '4':
                    phrase_reshape = np.reshape(np.array(phrase_to_num),
                                                [-1, interval_size, 1])
                    rnn_predictions = sess.run(rnn_output,
                                               feed_dict={in_placeholder:
                                                          phrase_reshape})
                    answers = get_top(3, list(rnn_predictions[0]))
                    print('Suggestions are:')
                    num = 1
                    for j in answers:
                        print('{}: {}'.format(num, rev_word_dict[j]))
                        num += 1
                    print('{}: Finish phrase\n'.format(num))
                    prompt = 'Select the number or type a word: '
                    next_word = input(prompt)

                    if next_word in ['1', '2', '3']:
                        phrase = phrase + ' ' +
                                   rev_word_dict[answers[int(next_word)-1]]
                        print('Your phrase so far: {}'.format(phrase))
                        phrase_to_num = phrase_to_num[1:]
                        phrase_to_num.append(answers[int(next_word)-1])
                    elif next_word == '4':
                        print('Final phrase: {}'.format(phrase))
                    else:
                        phrase = phrase + ' ' + next_word
                        print('Your phrase so far: {}'.format(phrase))
                        phrase_to_num = phrase_to_num[1:]
                        if next_word in word_dict.keys():
                            phrase_to_num.append(word_dict[next_word])
                        else:
                            phrase_to_num.append(word_dict_size)
                            word_dict_size += 1
```

The function begins by loading the dictionary of words files into the model, starting a TensorFlow session and restoring the trained model (weights and biases).

Considering that most of the code here is related to the user experience and rules validating the typed information, this section does not go through the details, although it is important to point out how the network receives the input.

We chose to use three words to check against the network, so even though there are more than three words in the phrase, only the last three are fed into the network. We get a hot vector as its return, order it by using the **`get_top`** function, then pick the three highest values and their index positions to match against the reverse word dictionary to get the actual words, and send the output to the user. This process continues on until the user finishes the phrase `Clicking option 4`.

5.4.7 Running the code

The steps to run this code are the same as the ones that are used for the sentiment analysis code. With IBM PowerAI, you simply can load the TensorFlow environment by running the TensorFlow environment by running **`source tensorflow-activate`** from the `/opt/DL/tensorflow/bin` directory, and then run your code as usual by calling **`python <your .py file`**>.

For our case, **`get_word.py`** is the starting point. To train the model, uncomment the lines that are shown in Example 5-11 and run **`python get_word.py`**.

Example 5-11 Lines of code to be uncommented for training

```
x = tf.placeholder('float', [None, interval_size, 1])
training(x)
```

After the model is trained, comment those two lines again and use only the **`get_word`** function.

The complete code is found on GitHub. For more information, see Appendix A, “Sentiment analysis code” on page 241.

5.4.8 Final considerations

The word suggestion code can be improved and implemented in different ways. In this section, we transformed each word into a number, although if more word context is required, this number can be represented as a Word2Vec, which is a complete representation. However, it requires a model to convert the words and more processing capacity.

Although the sample scenario does not implement retraining, it can be an interesting exercise. If a user types in a phrase where more than 70% of the words are not in the dictionary of words, the phrase can be inserted into the data set and the model retrained.

The hot vector representation of the label works, but if you grow it too much, then the number of words in the dictionary requires more processing.

Other parameters can be changed to increase the model accuracy, and a test data set is good way to check the accuracy without using only the data in the training set.

6

Introduction to IBM Spectrum Conductor Deep Learning Impact

IBM Spectrum Conductor Deep Learning Impact (DLI) is built on IBM Spectrum Conductor with Spark, a highly available and resilient multi-tenant distributed framework, which provides Apache Spark and deep learning (DL) applications lifecycle support.

This chapter contains the following topics:

- Benefits of IBM Spectrum Conductor Deep Learning Impact
- Key features of Deep Learning Impact
- DLI deployment
- Introduction to DLI graphic user interface
- Supported deep learning network and training engine in DLI
- Use case: Using a Caffe Cifar-10 network with DLI

6.1 Definitions, acronyms, buzzwords, and abbreviations

Table 6-1 shows the terminologies that are used in this chapter.

Table 6-1 Some terminologies that are used in this chapter

Definitions	Description
IBM PowerAI	Deep Learning Impact.
Inference	Wikipedia: Inference.
IBM Spectrum Conductor with Spark	IBM Spectrum Conductor with Spark enterprise platform.
Deep learning	A DL platform that is based on IBM Spectrum Conductor with Spark.
LMDB	Lightning Memory-Mapped Database, which is used by Caffe.
TFRecords	TFRecords is a commonly used TensorFlow training data format.
Parameter Server (PS)	GitHub Parameter Server.
Graphical processing unit (GPU)	Wikipedia: Graphical processing unit.
NCCL	NVIDIA Collective Communications Library.
Hyperparameters tuning	When you train a neural network, a different hyperparameters setting of the model affects the training result, such as the time cost or the model performance. Hyperparameter optimization or tuning searches a set of optimal hyperparameters for a DL model that has the best performance.
Tree-structured Parzen Estimator Approach (TPE)	*Algorithms for Hyper-Parameter Optimization*
Random search	Search the best hyperparameter setting by randomly sampling settings from all possible candidates.
Bayesian optimization	Bayesian optimization is a methodology for the global optimization of noisy blackbox functions. For more information, see *Practical Bayesian Optimization of Machine Learning Algorithms*.
Underfitting	Underfitting refers to a model that cannot model the training data or generalize new data.
Overfitting	Overfitting refers to a model that models the training data too well.
Gradient explosion	Exploding gradients arise in deep networks when the gradients' associating weights and the net's error become too large.
Saturation	If the input of activation function is too large or too small, the gradients that are computed are so close to zero that learning either becomes slow or stops working.

6.2 Benefits of IBM Spectrum Conductor Deep Learning Impact

On IBM Power platforms, IBM PowerAI provides many DL frameworks and useful technologies for the customer to take advantages of and implement. In many cases, customers want to use DL technology and integrate with their existing applications to bring new value to their business. There are some important elements to ensure a (not one) DL project can be put into production successfully, such as data management, arithmetic and neural network management, and computing resource management.

Figure 6-1 demonstrates a general lifecycle for DL projects.

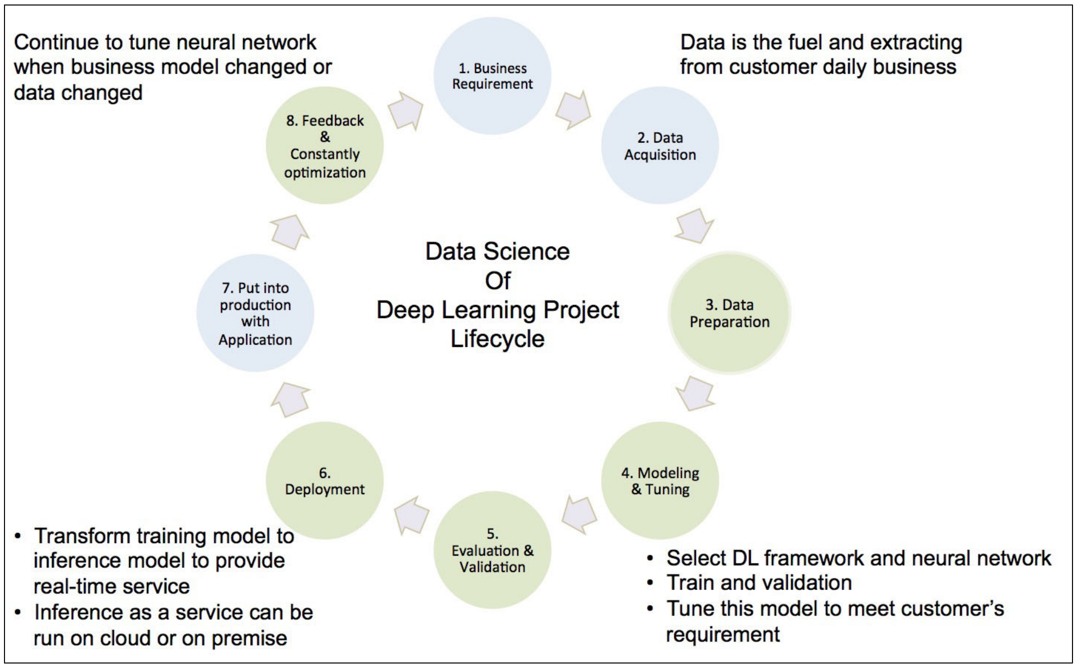

Continue to tune neural network when business model changed or data changed
1. Business Requirement
Data is the fuel and extracting from customer daily business
2. Data Acquisition
3. Data Preparation
4. Modeling & Tuning
5. Evaluation & Validation
6. Deployment
7. Put into production with Application
8. Feedback & Constantly optimization
Data Science Of Deep Learning Project Lifecycle
• Transform training model to inference model to provide real-time service
• Inference as a service can be run on cloud or on premise
• Select DL framework and neural network
• Train and validation
• Tune this model to meet customer's requirement

Figure 6-1 Data science of deep learning project lifecycle

A DL product always begins with the customer's business requirement (step 1). In most cases, data is provided by the customer. If you use a supervised training network, data labeling is needed too (step 2).

After you understand the business requirements and available data, you can implement one or more neural networks by completing the following high-level steps:

1. Reformatting or restructuring data to be in a usable/required format to be used by the model (step 3).
2. Import a DL network to create a training model (step 4).
3. Start a model training task (step 4).
4. Tune a model to improve accuracy (step 5).
5. Generate an inference model based on the trained model (step 6).
6. Integrate an inference model with a customer's application (step 6 and 7).
7. Put the model into production (step 7).
8. Generate an updated inference model to adapt to changing data (step 8).

These steps explain a closed-loop lifecycle for DL projects.

DLI takes effect from step 3 - 8, except for step 7. DLI can provide other benefits:

- Multi-tenant support
- Manages GPU resources dynamically
- Training visualization
- Early stop warning during training
- Hyperparameter searching for model tuning
- Supports multiple-node training with fabric technology, which provides easier integration and better speed ratio
- Provide a RESTful API for a customer's application to call an inference service

6.3 Key features of Deep Learning Impact

This section introduces some key features of DLI.

As a DL model development and management platform, the major targets of DLI are the following ones:

- To improve the accuracy of the neural network model
- To improve the training efficiency while processing a large-scale data set
- To provide real-time response for a user clients inference request after the model is deployed

To achieve these targets, DLI provides parallel data processing, hyperparameter tuning, parallel training with optimized gradient and weight reducing algorithm, training advisor, inference as service functions, and others.

6.3.1 Parallel data set processing

A DLI data set is used for model tuning, training, testing, and validation. Data set management features can help with a user management data set. A DLI data set can create, remove, and list data sets, monitor a data set's status, and view data set details. It can support and manage the following data set types:

- LMDB and TFRecords
- Images for classification
- Images for object detection
- Images vector
- CSV data
- Raw data

When creating a data set, DLI can:

- Randomly shuffle the data
- Split a data set to train data set, test data set, and validate data set
- Resize images
- Convert data to LMDB or TFRecords

6.3.2 Monitoring and Optimization for one training model

The function is used for training process monitoring and issue detection. Training a deep neural network is complex and time-consuming, so it is important to know the current progress and status of a user's training job. Monitoring and Optimization (MAO) provides the following features:

- Monitor the DL training job by capturing logs from instrumented underlying DL frameworks (filtering the logs based on keywords).
- Visualize the training progress by summarizing the training job's monitoring data.
- Advise and optimize the training (enables the user to accept them and have it implement the optimizations on their behalf).

The training visualization feature can visualize run time, iteration, loss, and accuracy metrics as histograms of weights, activations, and gradients of the neural network. From these charts, the user knows whether the training is running smoothly or there is something wrong.

During the training process, MAO can help the user detect the following issues of neural network model design:

- Gradient explosion
- Overflow
- Saturation
- Divergence
- Overfitting
- Underfitting

If DLI detects any of these issues, DLI estimates whether the training process can skip the issue. If not, DLI provides an early stop suggestion and the amendment suggestion to the user. The user can stop the training, adjust the model, and start again. The function helps the user improve the tuning model efficiency.

6.3.3 Hyperparameter optimization and search

Hyperparameters are parameters whose values are set before the start of the model training process. DL models such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) can have 10 - 100 hyperparameters, for example, the learning rate, the regulations, and others. These parameters affect the model training process and the final model performance. Hyperparameters optimization is one of the direct impediments to tuning DL models. There are three algorithms to automatically optimize the hyperparameters: Random search, TPE, and Bayesian optimization based on the Gaussian Process.

Random search

This method is the most popular search method in machine learning (ML) and DL. It easily supports parallel jobs with high efficiency. If a user has the candidate value for the hyperparameters and they are discrete, random search is the best choice.

Tree-structured Parzen Estimator Approach search

This is a tree-based search method and the sampling window follows the Parzen estimator. This is a sequential search model, and during each search iteration, in parallel, it searches the potential optimization point. The TPE search efficiency is between the random search and the Bayesian search.

Bayesian search

This method is a traditional and high accuracy method to search hyperparameters. It uses Gaussian probability density to describe the relationship between hyperparameters and loss value of neural networks, and takes advantage of the Expectation Maximization (EM) algorithm to constantly find the best result. Bayesian search is also a sequential search method, so it can be quickly convergent to the best optimal value.

Our implementation takes advantage of the state-of-art research results and the IBM Spectrum Conductor with Spark platform to efficiently parallel search the best set of hyperparameters that has the best final model performance.

6.3.4 IBM Fabric for distributed training

DLI uses IBM Fabric technology as part of its general distribution DL framework. IBM Fabric is a general distribution DL framework. The implementation provides comprehensive support for distribution DL across multiple GPUs and nodes. It provides compatibility with existing TensorFlow and Caffe models. IBM Fabric supports various parallelization schemes for a gradient descent method, which results in more than an 80% speedup ratio. Furthermore, IBM Fabric can efficiently scale out on IBM Spectrum Conductor with Spark clusters.

IBM Fabric mainly supports the following functions:

- Single Node, Multiple GPUs Training is supported without PS for TensorFlow and Caffe models.
- Multiple Nodes, Multiple GPUs Training is supported by PS for TensorFlow and Caffe models.
- Distribution synchronous and asynchronous training algorithms are supported for multi-nodes, including a synchronous gradient data control algorithm, an asynchronous gradient data control algorithm, a synchronous weight data control algorithm, and an asynchronous weight data control algorithm.
- NCCL is supported for broadcasting and reducing gradient and weight data across multiple GPUs.
- Continue training is supported by saving and restoring the training checkpoint and snapshot files.

6.3.5 IBM Fabric and auto-scaling

In addition to the distributed training engine with IBM Fabric, there is an added auto-scaling function that is available. The distributed training engine with IBM Fabric and auto-scaling function uses a fine-grained control for training. This is an alternative to the coarse-grained strategy, where the resource utilization is capped and no additional GPUs can be added to the training phase. With auto-scaling, the Spark scheduler distributes the task and the session scheduler dynamically issues resources from the resource manager, enabling more GPUs to be added.

In the reclaim case, the scheduler worker finishes the current training iteration, pushes the gradient on PS, and then exits and gracefully shuts down. In the scale-up case, the scheduler dynamic adds more workers to the training job to pick up the pending training iteration.

6.3.6 DLI inference model

This model puts the trained DL model into production, which is the most exciting moment for a data scientist. The DLI inference model function makes this step easy and efficient by just clicking several buttons in the portal. The inference model is also the most important way that a user can really benefit from the DL model. The IBM Spectrum Conductor with Spark DLI serves the data scientist and user by managing the inference model as a service in the IBM Spectrum Conductor with Spark cluster and provides access to the user with an advanced inference engine. Based on the powerful scheduling, elastic scaling, and efficient resource management features from IBM Spectrum Conductor with Spark, the inference request from the user can be serviced in IBM Spectrum Conductor with Spark DLI smoothly and efficiently.

The DLI inference model supports the following functions:

- Convert a trained model into an inference model.
- Create an inference model with pre-trained weight files.
- Infer from an IBM Spectrum Conductor with Spark DLI web portal and REST API.
- Manage an inference model and results from a web portal and REST API.
- Support an inference model preinstallation to reduce the response time and increase resource utilization.
- Provide large throughput.
- Support running in a large dynamic cluster.

6.3.7 Supporting a shared multi-tenant infrastructure

DLI is built on IBM Spectrum Conductor with Spark, which uses the IBM Enterprise Grid Orchestrator (IBM EGO) product to provide resource management and scheduling capabilities for Spark applications within the IBM Spectrum Conductor with Spark cluster. IBM EGO also provides the underlying system infrastructure to enable multiple applications to operate within a shared multi-tenant infrastructure.

Note: For more information, see IBM EGO.

Figure 6-2 shows the work flow to create a multi-tenant environment.

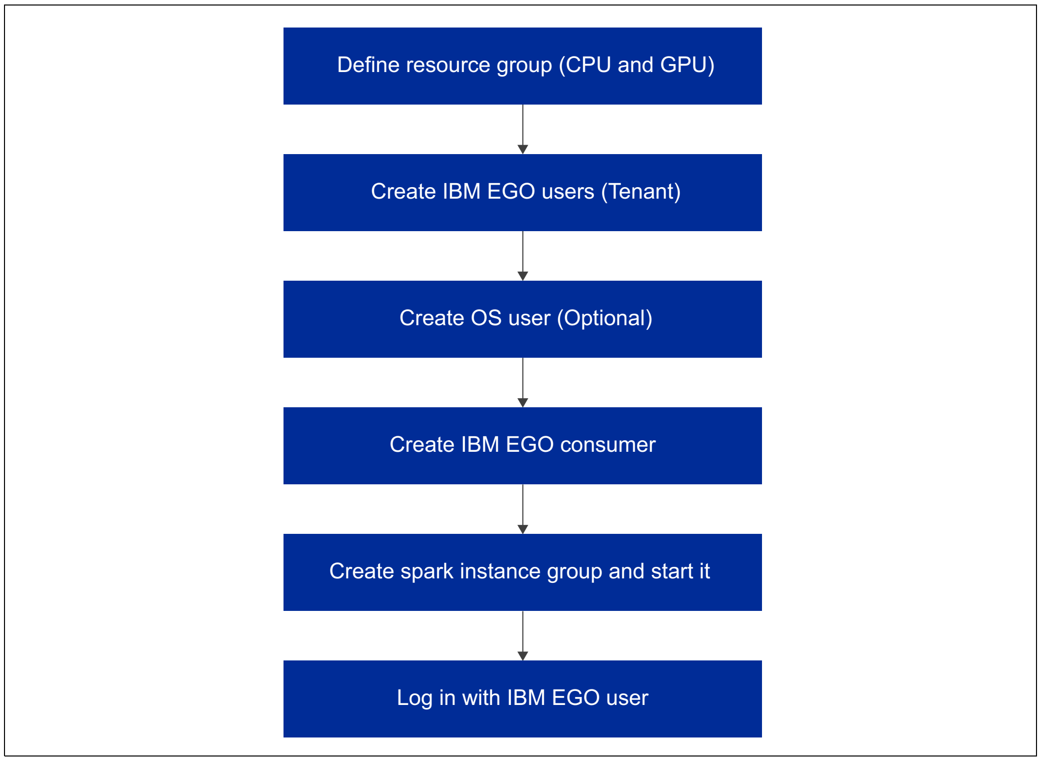
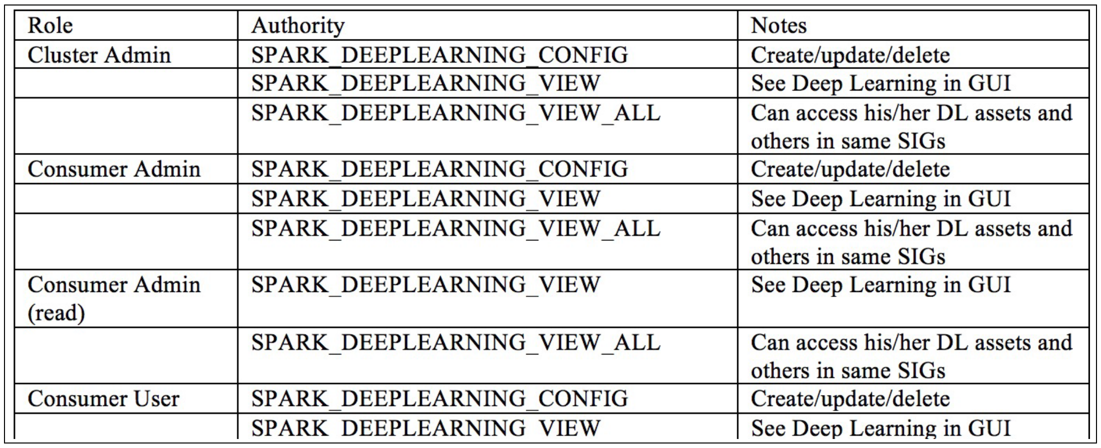

Define resource group (CPU and GPU)
Create IBM EGO users (Tenant)
Create OS user (Optional)
Create IBM EGO consumer
Create spark instance group and start it
Log in with IBM EGO user

Figure 6-2 Creating a multi-tenant environment work flow

During the creation of the IBM EGO user, there are different authorities for one user. Figure 6-3 describes the role and authority in IBM Spectrum Conductor with Spark.



Role	Authority	Notes
Cluster Admin	SPARK_DEEPLEARNING_CONFIG	Create/update/delete
	SPARK_DEEPLEARNING_VIEW	See Deep Learning in GUI
	SPARK_DEEPLEARNING_VIEW_ALL	Can access his/her DL assets and others in same SIGs
Consumer Admin	SPARK_DEEPLEARNING_CONFIG	Create/update/delete
	SPARK_DEEPLEARNING_VIEW	See Deep Learning in GUI
	SPARK_DEEPLEARNING_VIEW_ALL	Can access his/her DL assets and others in same SIGs
Consumer Admin (read)	SPARK_DEEPLEARNING_VIEW	See Deep Learning in GUI
	SPARK_DEEPLEARNING_VIEW_ALL	Can access his/her DL assets and others in same SIGs
Consumer User	SPARK_DEEPLEARNING_CONFIG	Create/update/delete
	SPARK_DEEPLEARNING_VIEW	See Deep Learning in GUI

Figure 6-3 Different role and authority for one IBM EGO user

Within one IBM Spectrum Conductor with Spark cluster, there is a session scheduler role to dispatch resources for different tasks. There are three policies: Priority, FIFO, and Fairshare.

Figure 6-4 on page 151 shows the multi-tenant environment, and resource management for tenants and applications.

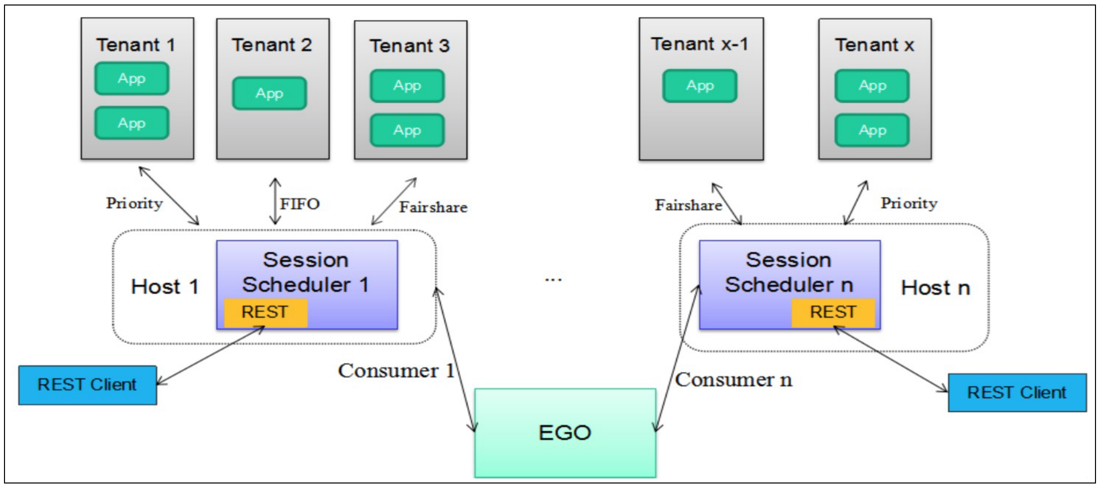


Figure 6-4 Multi-tenant resource management example

Note: To support the IBM Fabric and the auto-scaling function, set Fairshare for the schedule policy.

Multi-tenant example 1

Figure 6-5 is a straightforward scenario. In this scenario, there are two servers with their own CPU and GPU resources.

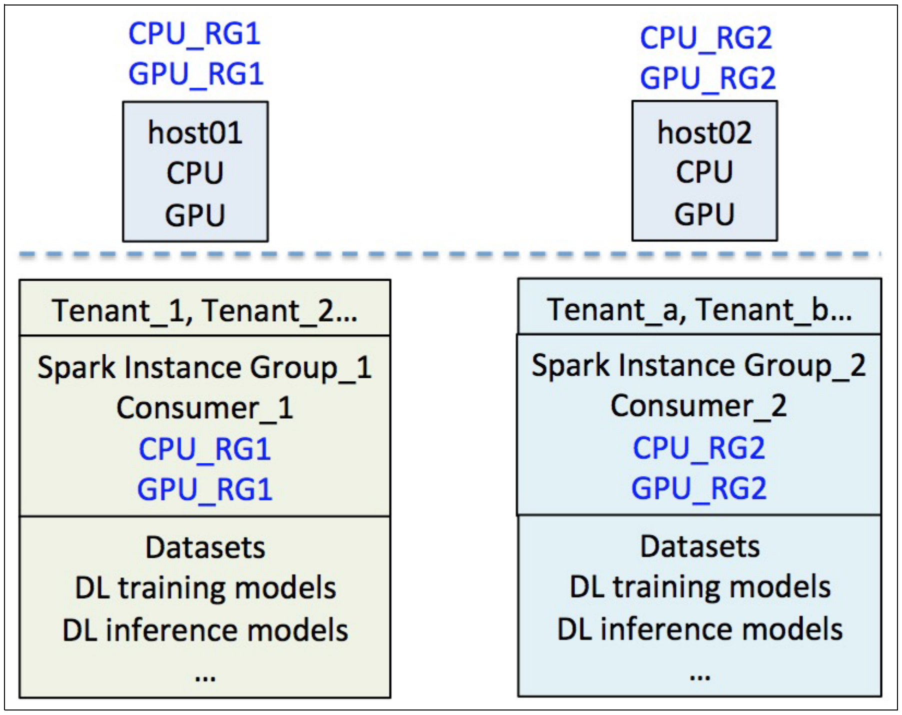


Figure 6-5 Multi-tenant example 1

The scenario has the following characteristics:

- The user creates a resource group for each host. Host1 has CPU_RG1 and GPU_RG1 resource groups and host2 has CPU_RG2 and GPU_RG2 resource groups.
- Consumer_1 includes CPU_RG1 and GPU_RG1 resource groups and tenant group 1 (Tenant_1, Tenant_2, and so on).
- Consumer_2 includes CPU_RG2 and GPU_RG2 resource groups and tenant group 2 (Tenant_a, Tenant_b, and so on).
- The Spark Instance Group (SIG) 1 includes consumer_1 and its resource groups. The SIG 2 includes consumer_2 and its resource groups.

With this kind of configuration, the two tenant groups use separate hardware resources for DL tasks, and can see only their own tasks.

Multi-tenant example 2

Figure 6-6 shows another sample scenario. In this scenario, there are two servers with their own CPU and GPU resources.

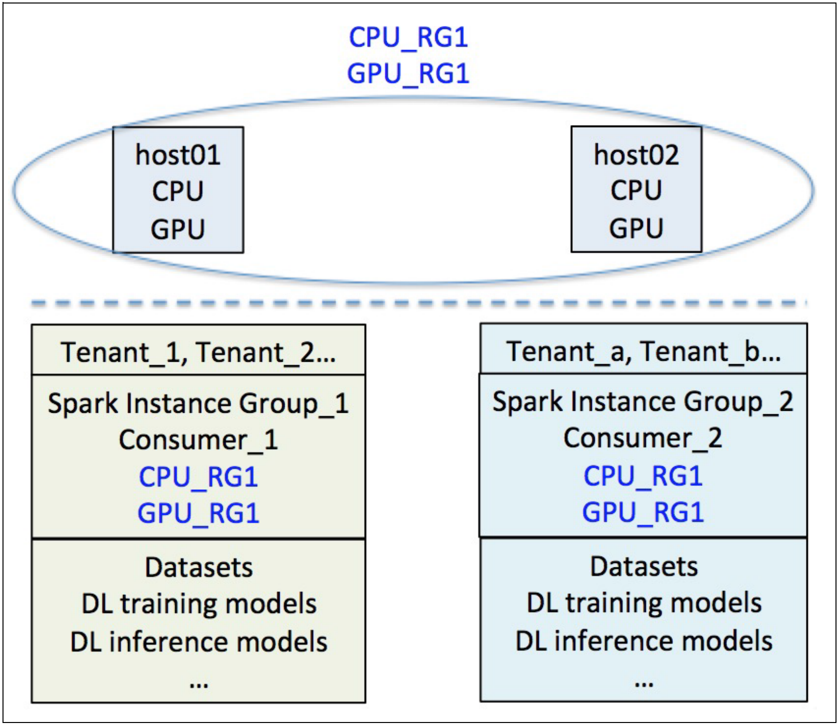


Figure 6-6 Multi-tenant example 2

The scenario has the following characteristics:

- All CPU resources of the two hosts are configured in one resource group: CPU_RG1.
- All GPU resources of the two hosts are configured in one resource group: GPU_RG.
- Consumer_1 includes the two resource groups and tenant group 1 (Tenant_1, Tenant_2, and so on).
- Consumer_2 includes the two resource groups and tenant group 2 (Tenant_a, Tenant_b, and so on).
- SIG 1 includes consumer_1 and the two resource groups.
- SIG 2 includes consumer_2 and the two resource groups.

With this kind of configuration, the two tenant groups share the hardware, and can see only their own DL tasks.

Multi-tenant example 3

Figure 6-7 is a simple example. In this scenario, there are two servers with their own CPU and GPU resources. The scenario has the following characteristics:

- All resources of host1 and host2 are configured into two resource groups: CPU_RG1 and GPU_RG2.
- All resources of host2 and host3 are configured into two resource groups: CPU_RG2 and GPU_RG2.
- Consumer_1 has CPU_RG1 and GPU_RG1 resource groups and tenant group 1 (Tenant_1, Tenant_2, and so on).
- Consumer_2 has CPU_RG2 and GPU_RG2 resource groups and tenant group 2 (Tenant_a, Tenant_b, and so on).
- SIG 1 includes consumer_1 and its resource groups.
- SIG 2 includes consumer_2 and its two resource groups.

With this kind of configuration, the two tenant groups share hardware of hosts2, but have their own hardware. The two tenant groups can see only their own DL tasks.

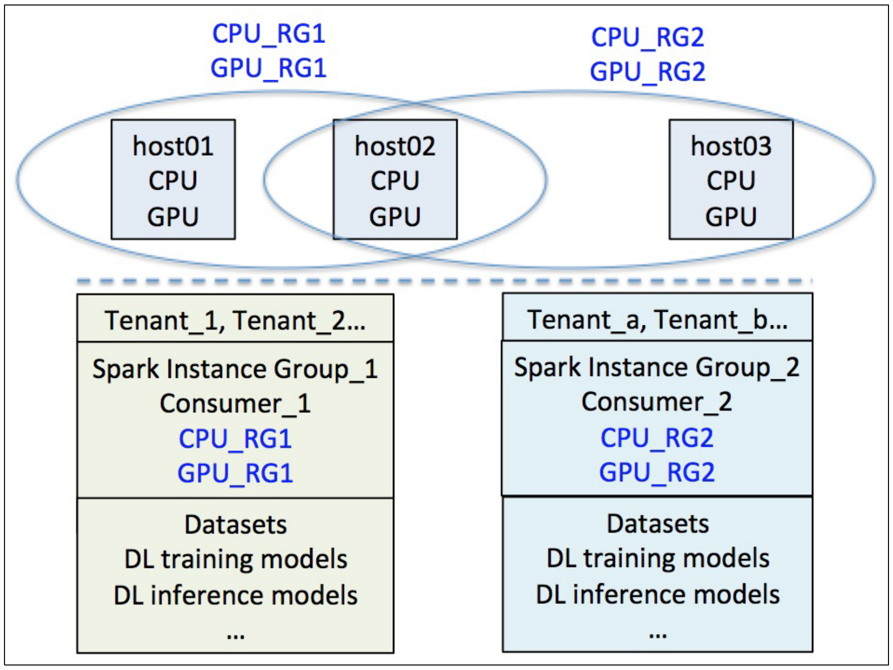


Figure 6-7 Multi-tenant example 3

6.4 DLI deployment

This section introduces how to deploy DLI on IBM PowerAI.

6.4.1 Deployment consideration

This section introduces three deployment modes: single node, multi-nodes without a high availability (HA) function, and multi-nodes with a HA function. In real case scenarios, there are some other methods, although they depend on the customer's requirements.

Regarding this deployment model, this scenario focus on a general architecture. In our environment, the version of software stack is as follows:

- DLI V1.1
- IBM Spectrum Conductor with Spark V2.2.1
- IBM PowerAI V1.5
- RHEL for Power Systems (ppc64le) V7.4
- CUDA V9.0.176
- GPU driver V384.81
- CUDA Deep Neural Network (cuDNN) V7.0.4
- IBM Spectrum MPI V10.1

Note: You must use a fully qualified domain name (FQDN) to access the IBM Spectrum Conductor with Spark GUI. In a DLI solution, you use the FQDN too. Here is an example of a `/etc/hosts` file FQDN entry:

```
172.16.51.96    dli01.dli.com   dli01
```

6.4.2 DLI single-node mode

In this mode, there is only one node. This mode is usually deployed for developing and testing purposes. The node acts as both master and compute roles. Figure 6-8 on page 155 shows the sequence of installation steps.

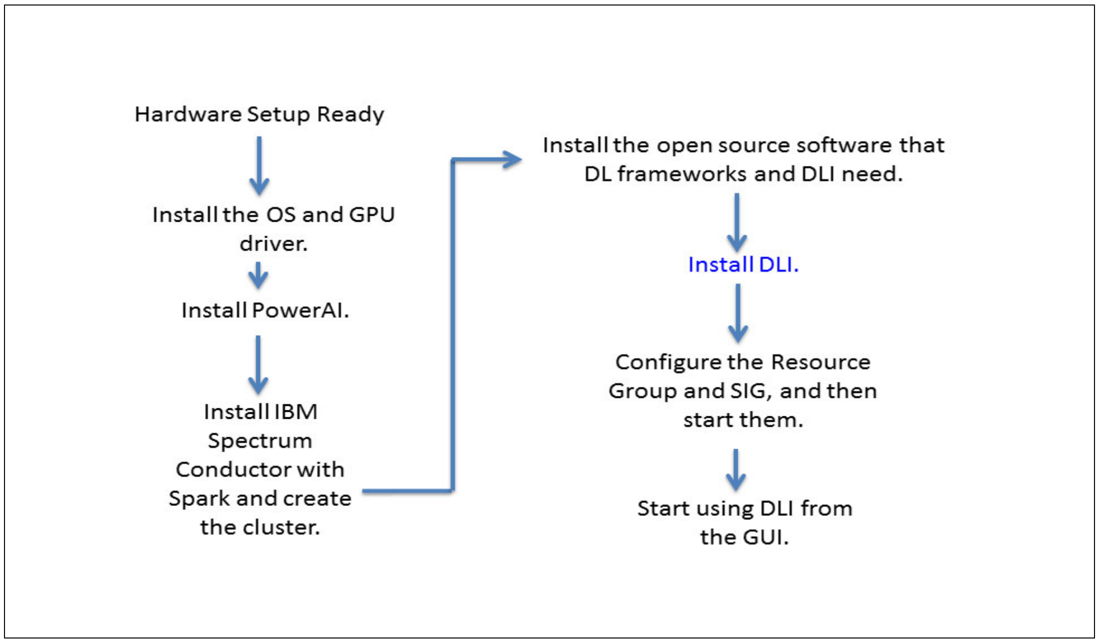

Hardware Setup Ready
Install the OS and GPU driver.
Install PowerAI.
Install IBM Spectrum Conductor with Spark and create the cluster.
Install the open source software that DL frameworks and DLI need.
Install DLI.
Configure the Resource Group and SIG, and then start them.
Start using DLI from the GUI.

Figure 6-8 Installation steps for single node DLI environment

For explanation of the first three steps, see Chapter 4, “Deploying IBM PowerAI” on page 59.

After IBM PowerAI is installed, you can start the installation of IBM Spectrum Conductor with Spark and create a cluster. You also must install the required open source software. To obtain the installation script, see GitHub.

The DLI application itself can be installed next, and the GPU resource and SIGs can be configured. The environment is then ready for DL activities.

6.4.3 DLI cluster without a high availability function

In this mode, there are several nodes in the DLI cluster, and there is always one node that acts as the management node. In this case, this node acts as the master node too, as shown in Figure 6-9.

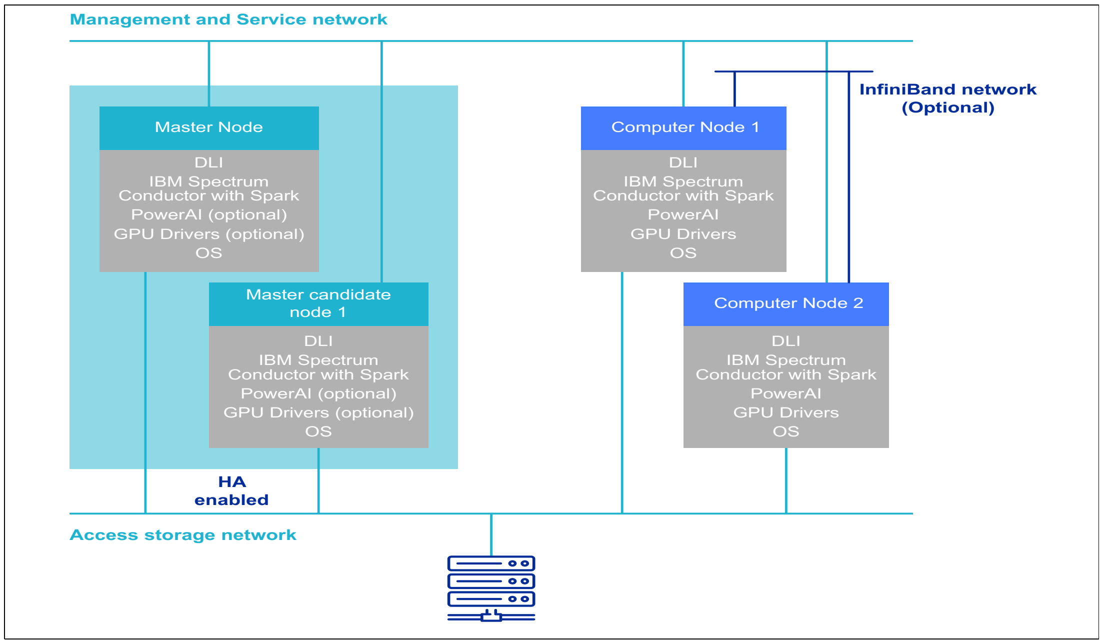


Figure 6-9 Topology of a DLI cluster without the high availability function

Figure 6-10 illustrates the typical installation steps for this configuration of the DLI cluster.

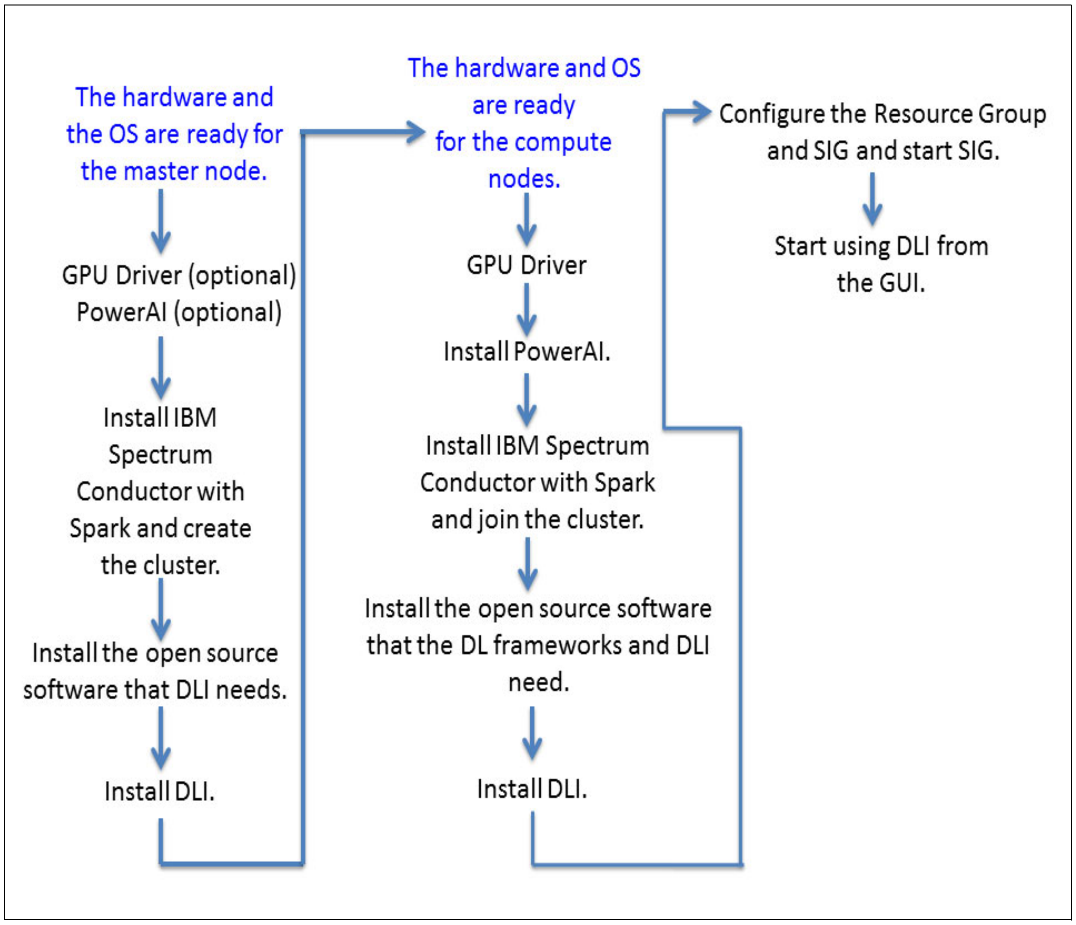

The hardware and the OS are ready for the master node.
GPU Driver (optional) PowerAI (optional)
Install IBM Spectrum Conductor with Spark and create the cluster.
Install the open source software that DLI needs.
Install DLI.
The hardware and OS are ready for the compute nodes.
GPU Driver
Install PowerAI.
Install IBM Spectrum Conductor with Spark and join the cluster.
Install the open source software that the DL frameworks and DLI need.
Install DLI.
Configure the Resource Group and SIG and start SIG.
Start using DLI from the GUI.

Figure 6-10 Installation steps for a DLI cluster without the high availability function enabled

6.4.4 DLI cluster with a high availability function

Figure 6-11 shows one topology where the HA function is enabled. The topology has two management nodes and two compute nodes.

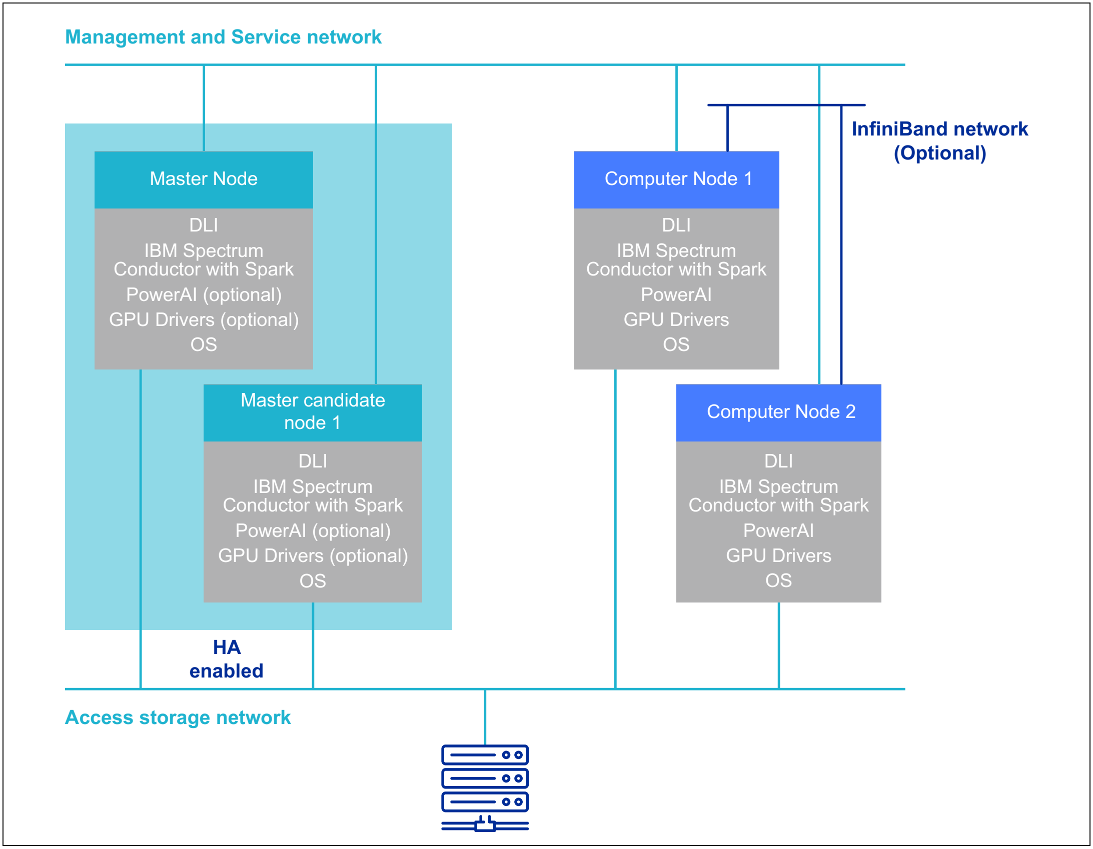


Figure 6-11 Topology of DLI cluster with high availability function enabled

Note: If you want to enable the DLI HA function, do not configure GPU devices in the master nodes because if a workload is running on the master node and this node fails (although all IBM Spectrum Conductor with Spark and DLI services can be restarted on the master candidate node), the workload must be manually restarted because it abnormally ended.

Figure 6-12 on page 159 shows the installation steps for this kind of cluster.

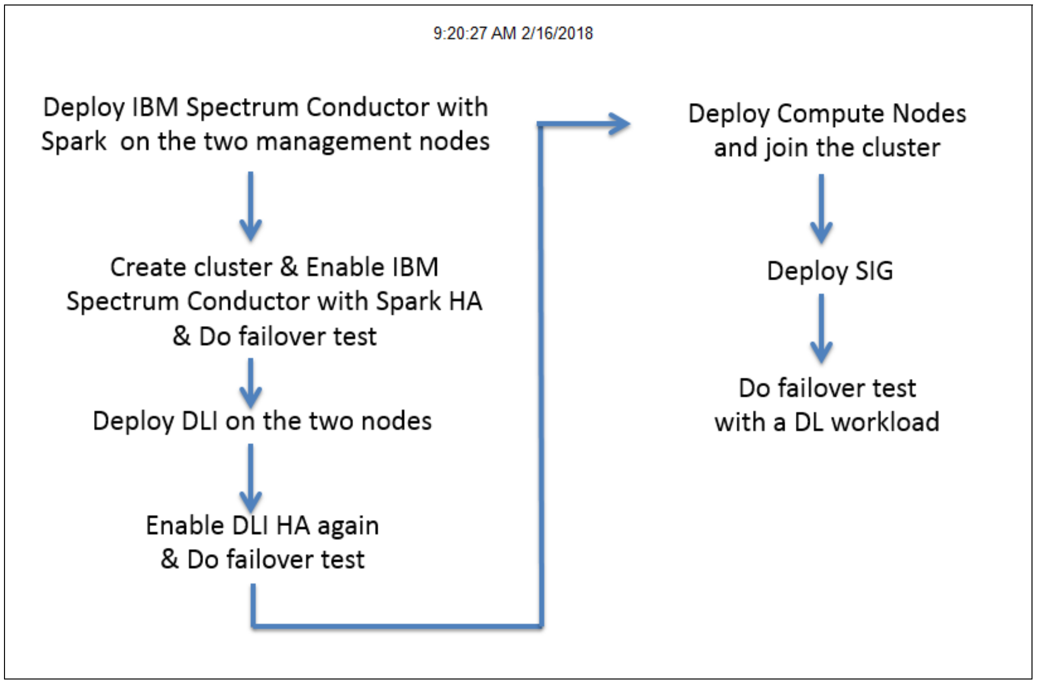

9:20:27 AM 2/16/2018
Deploy IBM Spectrum Conductor with Spark on the two management nodes
Create cluster & Enable IBM Spectrum Conductor with Spark HA & Do failover test
Deploy DLI on the two nodes
Enable DLI HA again & Do failover test
Deploy Compute Nodes and join the cluster
Deploy SIG
Do failover test with a DL workload

Figure 6-12 Installation steps for a DLI cluster with the high availability function enabled

6.4.5 Binary files installation for the high availability enabled cluster

Before you install a HA enabled DLI cluster, you must consider where to place the IBM Spectrum Conductor with Spark and DLI files. The two typical options are locally or in a shared file system.

Note: The shared file system implementation is not described in this section. We assume that you have implemented a shared file system by using IBM Spectrum Scale™ or any file system of your preference.

Binary files in local disks

In this installation, all binary files are installed in a local file system, as shown in Figure 6-13. However, all configuration files and log files are installed in the shared file system.

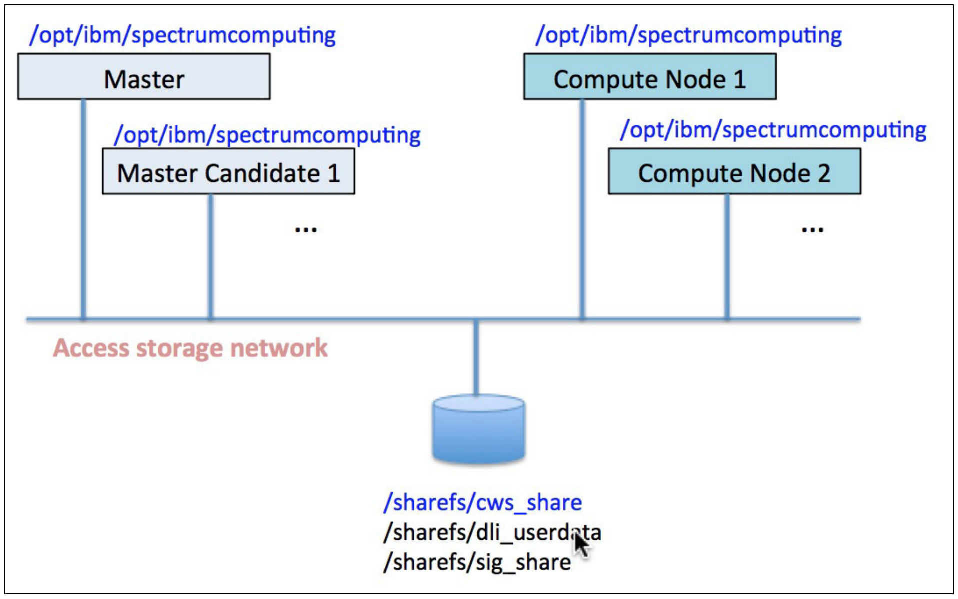


Figure 6-13 An example of file system layout for location installation

Note: The `/opt/ibm/spectrumcomputing` directory is the default directory to place IBM Spectrum Conductor with Spark and DLI binary files. You can change it during the installation.

Binary files in shared file systems

Figure 6-13 illustrates an example installation for a HA enabled DLI cluster. During the installation sequence, you set one option to specify a target installation directory, which is in a shared file system. Then, all binary files and other files are installed in this shared file system, as shown in Figure 6-14 on page 161.

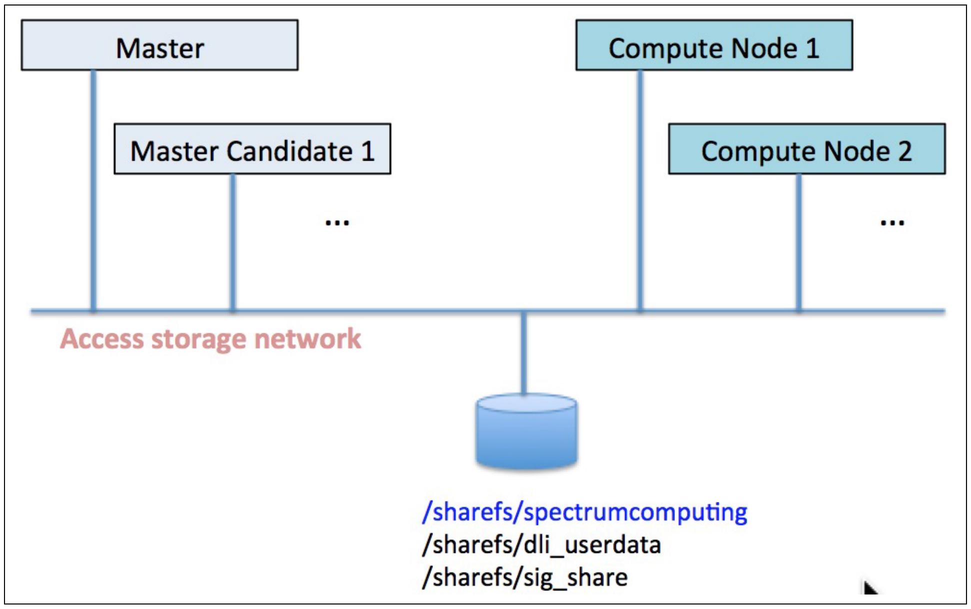


Figure 6-14 Shared file system installation

Note: The `/sharefs/spectrumcomputing` directory is an example shared file system for the IBM Spectrum Conductor with Spark installation.

The `/sharefs/dli_userdata` directory is an example of setting the shared file system during a DLI installation.

THe `/sharefs/sig_share` directory is an example of setting the shared file system during the SIG creation because it is used for the SIG HA function.

6.4.6 A DLI cluster with a high availability function installation guide

This section provides an overview about how to set up a DLI cluster and enable the HA feature.

Topology

In this case, the topology consists of three nodes: Two nodes are configured as management nodes, and one node is configured as the compute node. The two management nodes enable the HA function. The compute node runs the DL workload, as shown in Figure 6-15.

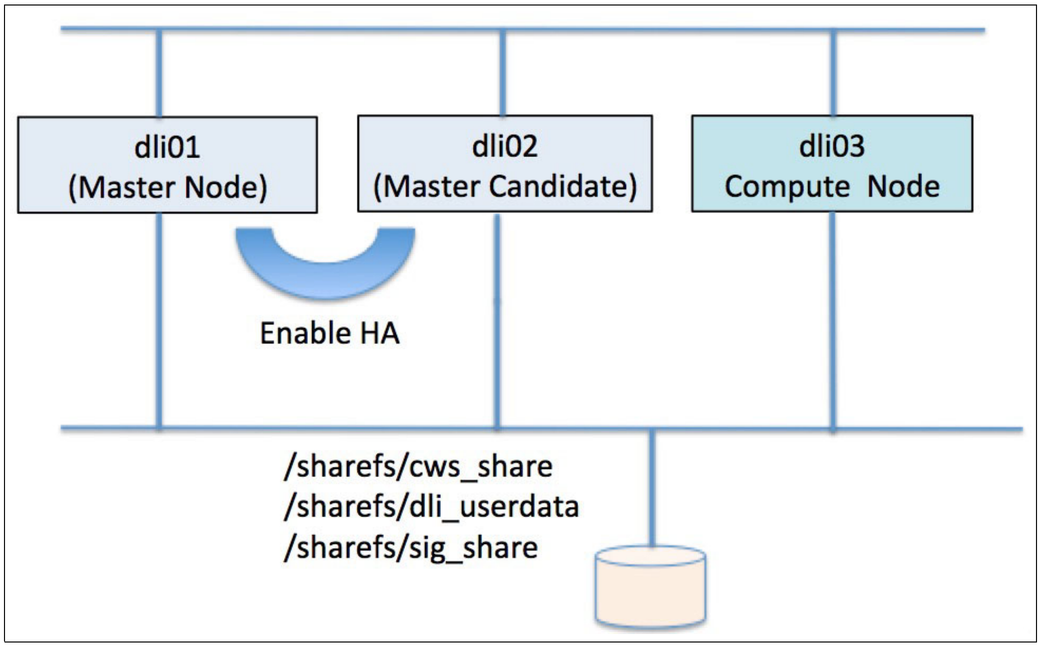


Figure 6-15 Topology of a DLI cluster

Regarding the binary files location, this example uses a local file system with a default value of `/opt/ibm/spectrumcomputing`.

Figure 6-16 on page 163 shows the installation steps for this configuration.

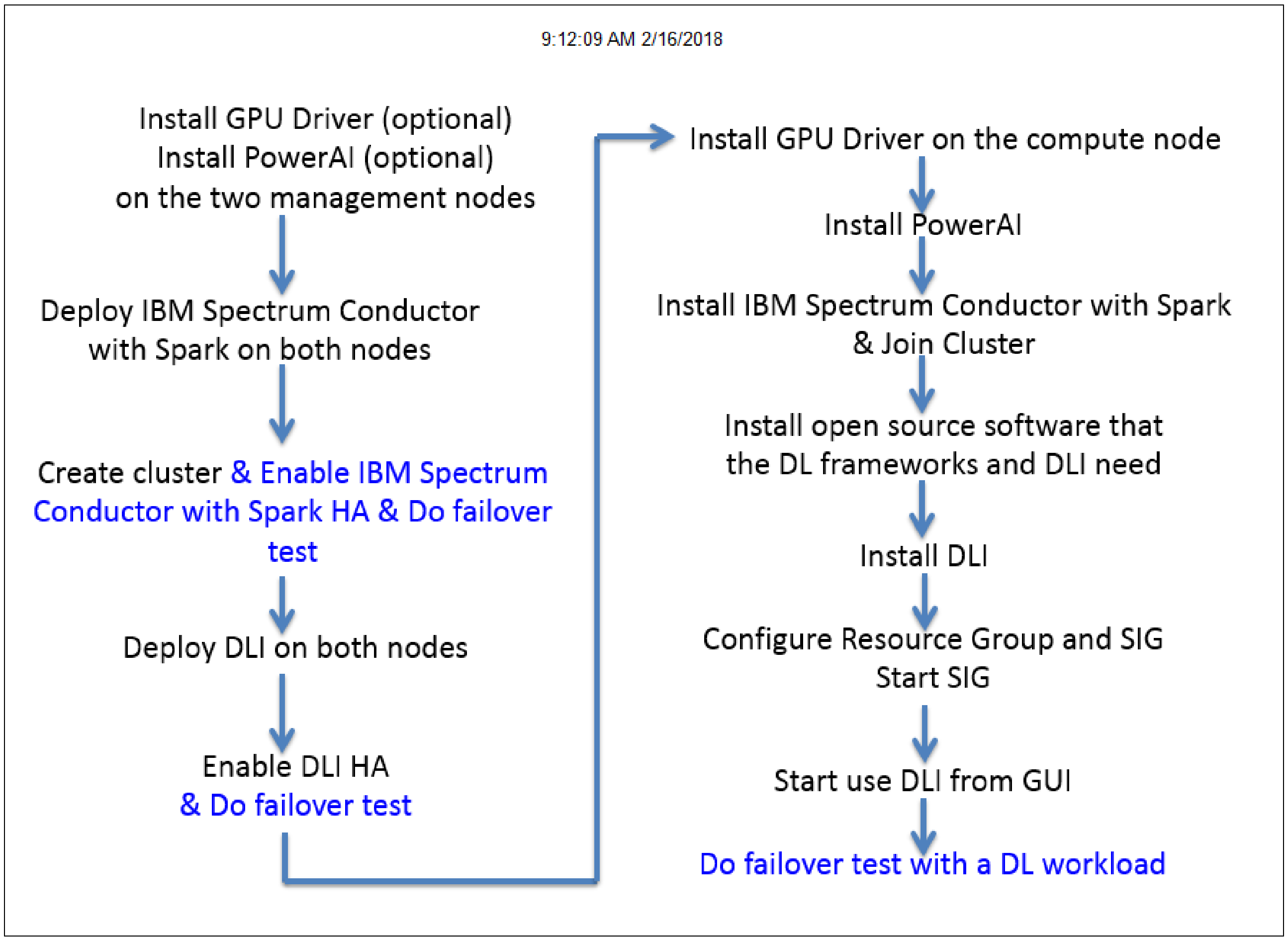

9:12:09 AM 2/16/2018
Install GPU Driver (optional)
Install PowerAI (optional)
on the two management nodes
Deploy IBM Spectrum Conductor with Spark on both nodes
Create cluster & Enable IBM Spectrum Conductor with Spark HA & Do failover test
Deploy DLI on both nodes
Enable DLI HA & Do failover test
Install GPU Driver on the compute node
Install PowerAI
Install IBM Spectrum Conductor with Spark & Join Cluster
Install open source software that the DL frameworks and DLI need
Install DLI
Configure Resource Group and SIG
Start SIG
Start use DLI from GUI
Do failover test with a DL workload

Figure 6-16   Installation steps

It is expected that the required operating system is installed and configured according to the installation guide and that IBM PowerAI itself is installed on all nodes.

Because HA DLI relies on the IBM Spectrum Conductor with Spark HA framework, do any required HA testing for IBM Spectrum Conductor with Spark before starting DLI. The HA tests must be repeated after the subsequent installation of DLI and again with a running DL workload.

Steps to install two management nodes

As shown in Example 6-1, some configuration is required across all the nodes before you begin the installation.

Example 6-1   Environment settings for all nodes

```
root@dli01:~# cat /etc/hosts
...
172.16.51.96    dli01.dli.com dli01
172.16.51.97    dli02.dli.com dli02
172.16.51.98    dli03.dli.com dli03

root@dli01:~# cat /etc/sysctl.conf
...
vm.max_map_count = 262144
```

```
root@dli01:~# useradd egoadmin -m

root@dli01:~# cat /etc/security/limits.conf
egoadmin   soft    nproc     65535
egoadmin   hard    nproc     65535
egoadmin   soft    nofile    65535
egoadmin   hard    nofile    65535
root - nproc 65536
root - nofile 65536
* - memlock unlimited

root@dli01:~# cat /etc/security/limits.d/20-nproc.conf
*          soft    nproc     4096
root       soft    nproc     unlimited
egoadmin   -       nproc     65536
```

Note: Configuring the Network Time Protocol (NTP) is a requirement in a distributed multi-node environment.

In this example environment, we use a simple NFS mount as a shared file system. In a production environment, use IBM Spectrum Scale (or its equivalent) to provide the shared file system.

The detailed sequence of steps that are used to install and configure the two management nodes are documented in Table 6-2.

Table 6-2 Steps for management nodes installation

Operations on dli01 node		**Operations on dli02 node**	
User	**Operation**	**User**	**Operation**

root	Install IBM Spectrum Conductor with Spark:
	`export CLUSTERADMIN=egoadmin`
	`export CLUSTERNAME=dlicluster`
	`export LANG=C`
	`bash -x cws-2.2.1.0_ppc64le.bin --quiet`		

egoadmin	Create the IBM Spectrum Conductor with Spark cluster:
	`source /opt/ibm/spectrumcomputing/profile.platform`
	`egoconfig join $(hostname) -f`		

egoadmin	Set the entitlement: `egoconfig setentitlement entitle_file_cws.dat`		

Operations on dli01 node		Operations on dli02 node	

root	Start the IBM Spectrum Conductor with Spark service and check its status:
	`source /opt/ibm/spectrumcomputing/profile.platform`
	`egosh ego start`
	`sleep 5`
	`egosh user logon -u Admin -x Admin`
	`egosh service list`		

		root	Install IBM Spectrum Conductor with Spark.

		egoadmin	Join the cluster:
			`source /opt/ibm/spectrumcomputing/profile.platform`
			`egoconfig join dli01`

		root	Start the IBM Spectrum Conductor with Spark service and check its status:
			`source /opt/ibm/spectrumcomputing/profile.platform`
			`egosh ego start all`
			`egosh user logon -u Admin -x Admin`
			`egosh service lis`

root	Stop the IBM Spectrum Conductor with Spark service for both nodes:
	`source /opt/ibm/spectrumcomputing/profile.platform`
	`egosh service stop all`
	`egosh ego shutdown all`		

egoadmin	Enable the HA function for IBM Spectrum Conductor with Spark:
	`source /opt/ibm/spectrumcomputing/profile.platform`
	`egoconfig mghost /sharefs/cws_share`		

		egoadmin	Enable the HA function for IBM Spectrum Conductor with Spark:
			`source /opt/ibm/spectrumcomputing/profile.platform`
			`egoconfig mghost /sharefs/cws_share`

root	Start the IBM Spectrum Conductor with Spark cluster services:
	`source /opt/ibm/spectrumcomputing/profile.platform`
	`egosh ego start all`
	`egosh user logon -u Admin -x Admin`
	`egosh service list`		

root	Log in to the IBM Spectrum Conductor with Spark GUI and update the Master candidate list in the Master and Failover window. Add dli02 to the Master candidate list and click Apply, which restarts the cluster service automatically. “How to log in to the IBM Spectrum Computing Management Console” on page 177 shows how to log in to IBM Spectrum Conductor with Spark. “Configuring the Master and Failover list” on page 181 shows how to configure the failover candidate node list.		
root	Run **`‘init 0’`** to simulate that dli01 crashed.		

Operations on dli01 node		Operations on dli02 node	

		root	As expected, all the master node’s services are taken over by the dli02 node within 3 minutes:
			`egosh service list`
			`egosh resource list`
			`egosh resource -m`

root	Start the dli01 virtual machine (VM) and start the IBM Spectrum Conductor with Spark service:
	`source /opt/ibm/spectrumcomputing/profile.platform`
	`egosh ego start all`
	`egosh user logon -u Admin -x Admin`
	`egosh service lis`
	Check the cluster status.		

root	Install the open source software that is required by the neural network frameworks.		
		root	Install the open source software that is required by the neural network frameworks.

root	Install DLI:
	`cd /bin`
	`rm sh`
	`ln -s bash sh`
	`source /opt/ibm/spectrumcomputing/profile.platform`
	`export CLUSTERADMIN=egoadmin`
	`export CLUSTERNAME=dlicluster`
	`export DLI_CONDA_HOME=/opt/anaconda2`
	`export DLI_SHARED_FS="/sharefs/dli_userdata"`
	`LANG=C bash dli-1.1.0.0_ppc64le.bin --quiet`		

		root	Install DLI.

root	Stop the IBM Spectrum Conductor with Spark service for both nodes:
	`source /opt/ibm/spectrumcomputing/profile.platform`
	`egosh service stop all`
	`egosh ego shutdown all`		

egoadmin	Add the DLI entitlement: `egoconfig setentitlement entitle_file_dli.dat`		

egoadmin	Enable the HA function:
	`source /opt/ibm/spectrumcomputing/profile.platform`
	`egoconfig mghost /sharefs/cws_share`		

Operations on dli01 node		Operations on dli02 node	

		egoadmin	Enable the HA function:
			`source`
			`/opt/ibm/spectrumcomputing/profile.pl`
			`atform`
			`egoconfig mghost /sharefs/cws_share`

root	Refresh the environment variable and start the IBM Spectrum Conductor with Spark service for both nodes:
	`source /opt/ibm/spectrumcomputing/profile.platform`
	`egosh ego start all`		

		root	Run **`'init 0'`** to simulate that dli02 crashed.
root	Check the cluster's status and the DLI services.		
root			Start dli02 and start the service.

Steps to install the compute node

After the installation and testing of the two management nodes is done, install DLI on the compute node, as shown in Table 6-3.

Table 6-3 Steps for compute node installation

User	Operation
root	Install the operating system, GPU driver. and IBM PowerAI.
root	Install IBM Spectrum Conductor with Spark.
egoadmin	Join the existing cluster.
root	Start the IBM Spectrum Conductor with Spark service.
root	Install the required open source software for DL frameworks and DLI.
root	Install DLI.

root	Check the cluster status:
	`root@dli03:~# egosh resource list`
	`NAME     status       mem    swp    tmp   ut    it    pg   r1m  r15s  r15m ls`
	`dli02.d* ok           56G   167M    36G   1%     0   0.0   0.3   0.1   0.2   1`
	`dli03.d* ok           51G   167M    40G   9%     5   0.0   2.0   4.0   0.6   1`
	`dli01.d* ok           46G   167M    36G   1%     5   0.0   0.6   0.9   0.8   1`
	
	`root@dli03:~# egosh resource list -m`
	`IBM EGO current master host name : dli01.dli.com`
	`Candidate master(s)          : dli01.dli.com dli02.dli.com`

Enable IBM Spectrum Conductor with Spark to monitor GPU resources

At the time of writing, a script must be run to enable IBM Spectrum Conductor with Spark to monitor GPU resources. Example 6-2 shows an example usage of the current command.

Example 6-2 How to enable IBM Spectrum Conductor with Spark to monitor and manage GPU resources

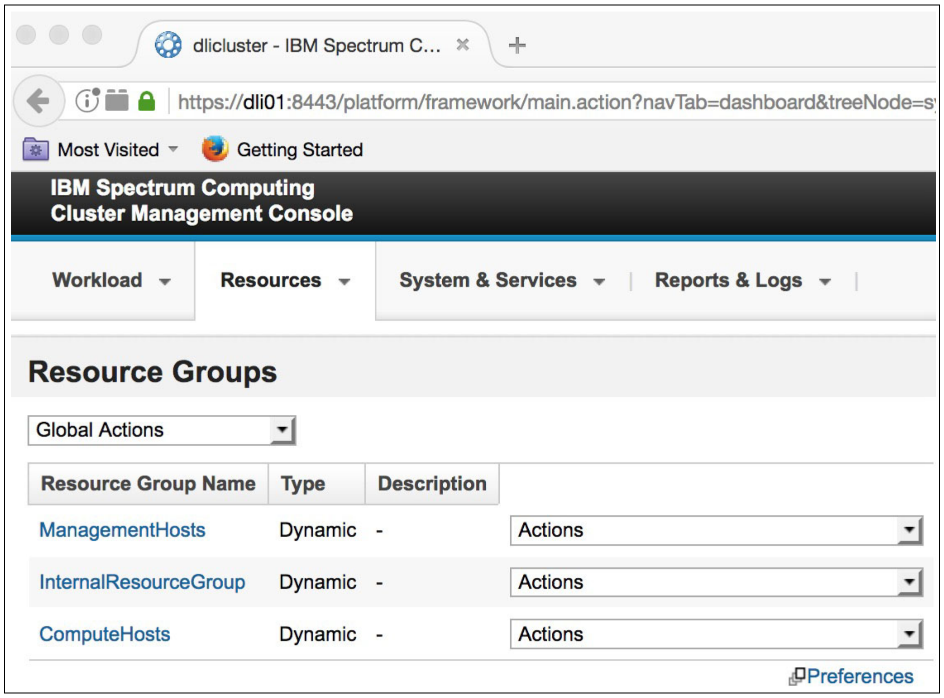
```
#source /opt/ibm/spectrumcomputing/profile.platform
#/opt/ibm/spectrumcomputing/conductorspark/2.2.1/etc/gpuconfig.sh enable --quiet
-u Admin -x Admin
```

To log in to the IBM Spectrum Computing Cluster Management Console, see “How to log in to the IBM Spectrum Computing Management Console” on page 177.

Creating resource groups

To create the resource groups, complete the following steps:

1. Click **Resources** →**Resource Planning (Slots)** →**Resource Groups**. Figure 6-17 shows the default resource group configuration.


dlicluster - IBM Spectrum C...
https://dli01:8443/platform/framework/main.action?navTab=dashboard&treeNode=s
Most Visited
Getting Started
IBM Spectrum Computing
Cluster Management Console
Workload
Resources
System & Services
Reports & Logs
Resource Groups
Global Actions

Resource Group Name	Type	Description	
ManagementHosts	Dynamic	-	Actions
InternalResourceGroup	Dynamic	-	Actions
ComputeHosts	Dynamic	-	Actions

Figure 6-17 Resource group configuration window

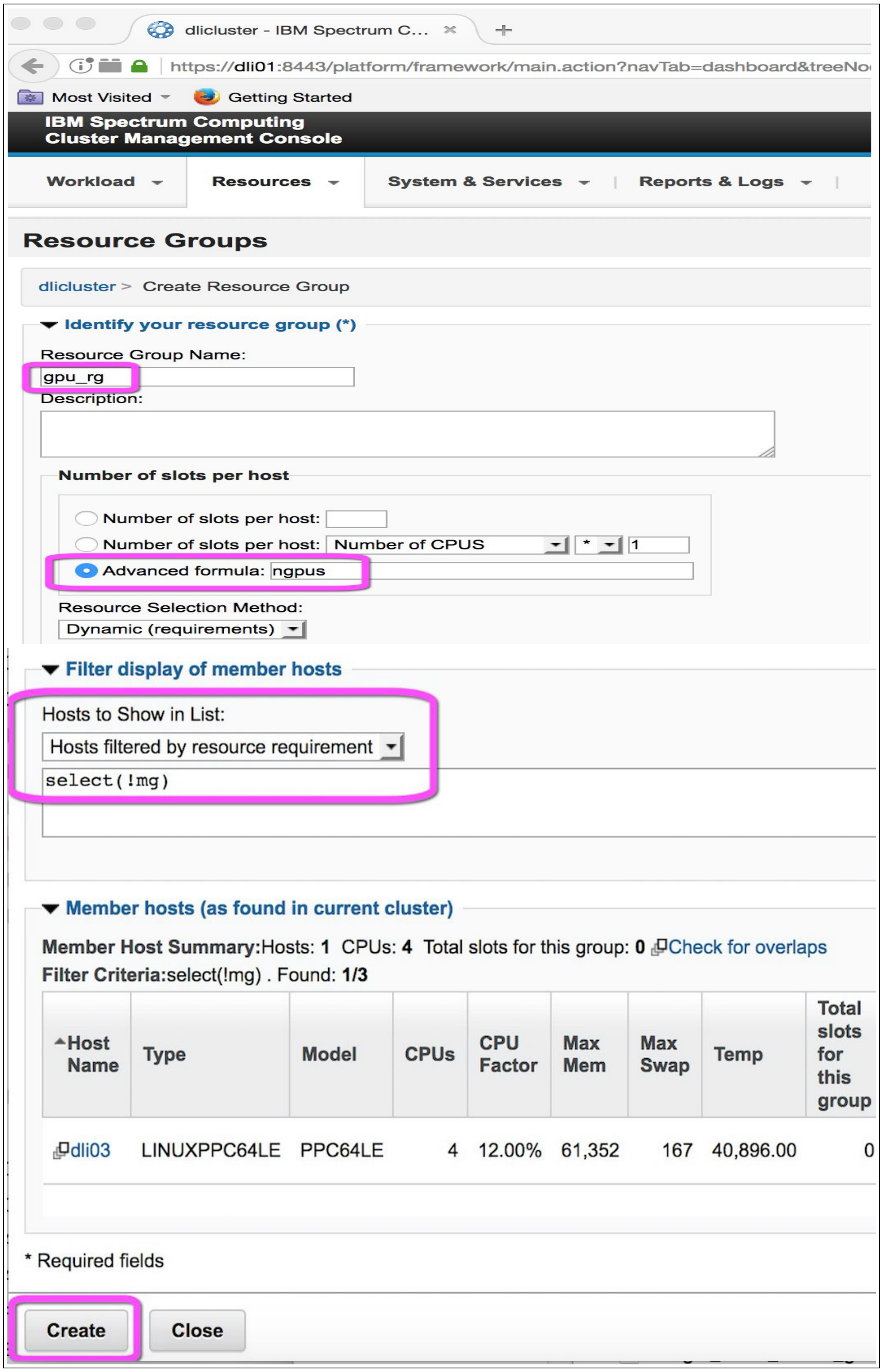
Alternatively, the same information can be retrieved by running a command, as shown in Example 6-3.

Example 6-3 How to view a resource group from the command line

```
root@dli01:~# egosh resourcegroup view ComputeHosts
---------------------------------------------------------------------------
Name          : ComputeHosts
Description   :
Type          : Dynamic
ResReq        : select(!mg)
SlotExpr      :
NHOSTS        : 1

SLOTS         FREE          ALLOCATED
4             4             0

Resource List : dli03.dli.com

root@dli01:~# egosh resourcegroup view ManagementHosts
---------------------------------------------------------------------------
Name          : ManagementHosts
Description   :
```

```
Type          : Dynamic
ResReq        : select(mg)
SlotExpr      :
NHOSTS        : 2

SLOTS         FREE          ALLOCATED
64            43            21

Resource List : dli02.dli.com dli01.dli.com
```

2. From the drop-down list box, select **Create a Resource Group** and set the parameters for this new resource group, as shown in Figure 6-18 on page 170.


dlicluster - IBM Spectrum C...
https://dli01:8443/platform/framework/main.action?navTab=dashboard&treeNo
Most Visited
Getting Started
IBM Spectrum Computing
Cluster Management Console
Workload
Resources
System & Services
Reports & Logs
Create
Close

Resource Groups

dlicluster > Create Resource Group

Identify your resource group (*)

Resource Group Name: gpu_rg

Description:

Number of slots per host

- Number of slots per host:
- Number of slots per host: Number of CPUS * 1
- [✓ Advanced formula: ngpus]

Resource Selection Method: Dynamic (requirements)

Filter display of member hosts

Hosts to Show in List: Hosts filtered by resource requirement

```
select(!mg)
```

Member hosts (as found in current cluster)

Member Host Summary: Hosts: **1** CPUs: **4** Total slots for this group: **0** Check for overlaps

Filter Criteria:select(!mg) . Found: **1/3**

Host Name	Type	Model	CPUs	CPU Factor	Max Mem	Max Swap	Temp	Total slots for this group
dli03	LINUXPPC64LE	PPC64LE	4	12.00%	61,352	167	40,896.00	0

* Required fields

Figure 6-18 Setting the parameters for the GPU resource group

Note: The `ngpus` value sets the slot number to the physical GPU adapter number. If there is a requirement to share a GPU adapter for multiple tasks, the field can be defined with a suitable multiple, for example, `ngpus*2`.

The same resource group can also be created from the command line, as shown Example 6-4.

Example 6-4 Viewing the GPU resource group from the command line

```
root@dli01:~# egosh resourcegroup list
NAME                  HOSTS         SLOTS         FREE          ALLOCATED
gpu_rg                1             1             1             0
ComputeHosts          1             4             4             0
InternalResourceGroup 3             30            21            9
ManagementHosts       2             64            43            21
```

Creating the Spark Instance Group

After the GPU resource group is created, create the SIG by completing the following steps:

1. Select **Workload** →**Spark** →**Spark Instance Groups**, as shown in Figure 6-19.

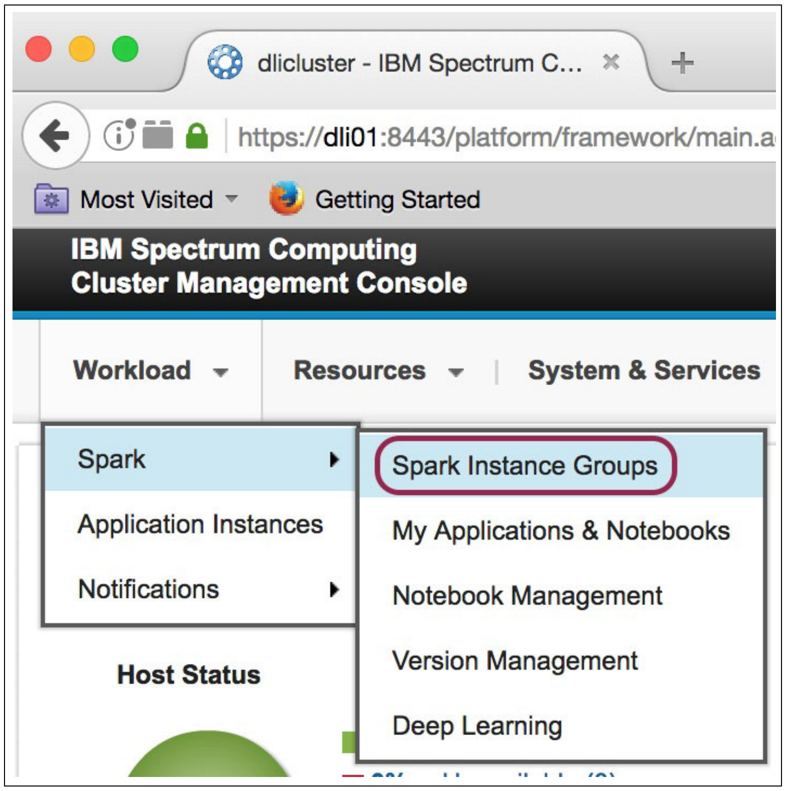


Figure 6-19 Creating Spark Instance Group: 1

2. Click **New** →**Template**, as shown in Figure 6-20.

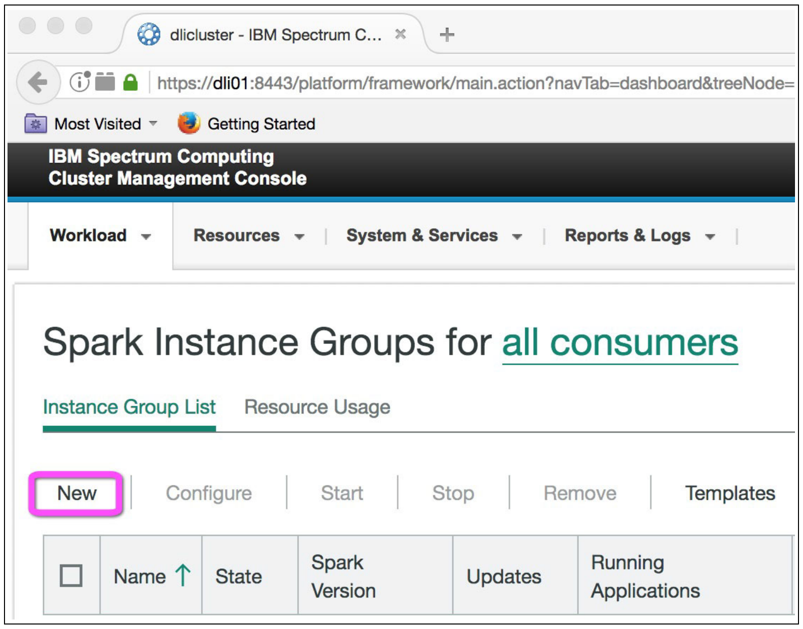


Figure 6-20 Creating Spark Instance Group: 2

3. In the template window, there is one existing template that is named dli-sig-template. Click **Use**, as shown in Figure 6-21.

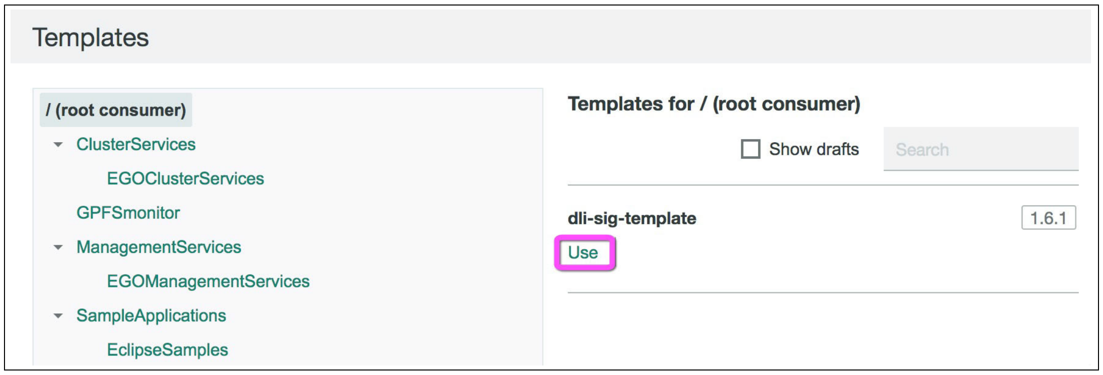


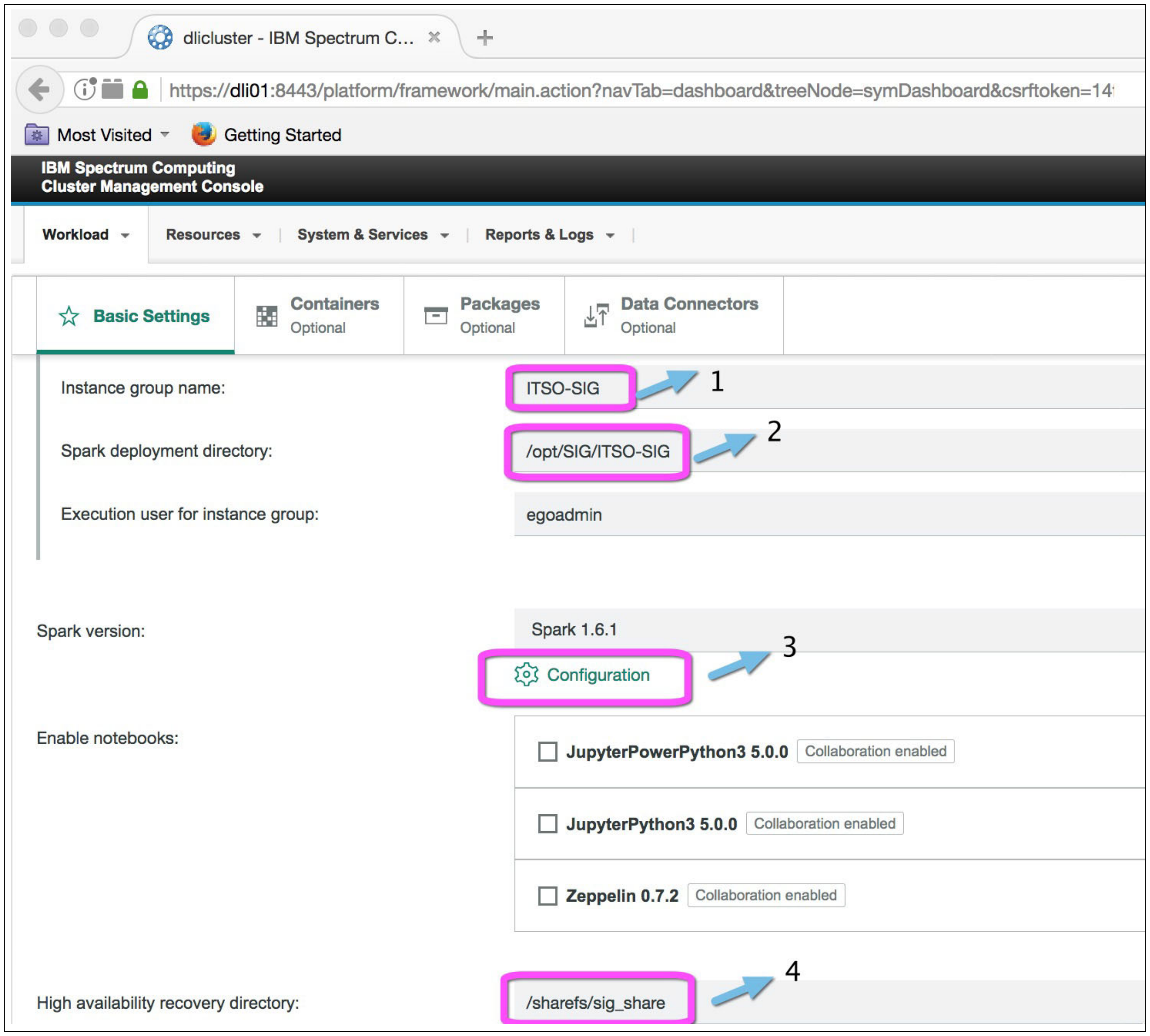
Figure 6-21 Creating Spark Instance Group: 3

4. Complete the required parameters of this new SIG, as shown in Table 6-4.

Table 6-4 Parameter settings for the SIG

Parameter	**Value and comments**
Instance Group Name	`ITSO-SIG`
Spark deployment directory	`/opt/SIG/ITSO-SIG`
`JAVA_HOME`	`/usr/lib/jvm/java-1.8.0/jre` Click **Configuration** under Spark version to set.
HA recovery directory	`/sharefs/sig_share`
Spark executors (GPU slots)	`gpu_rg`

Figure 6-22, Figure 6-23 on page 174, and Figure 6-24 on page 175 show the configuration windows for these SIG parameters.



Instance group name: ITSO-SIG

Spark deployment directory: /opt/SIG/ITSO-SIG

Execution user for instance group: egoadmin

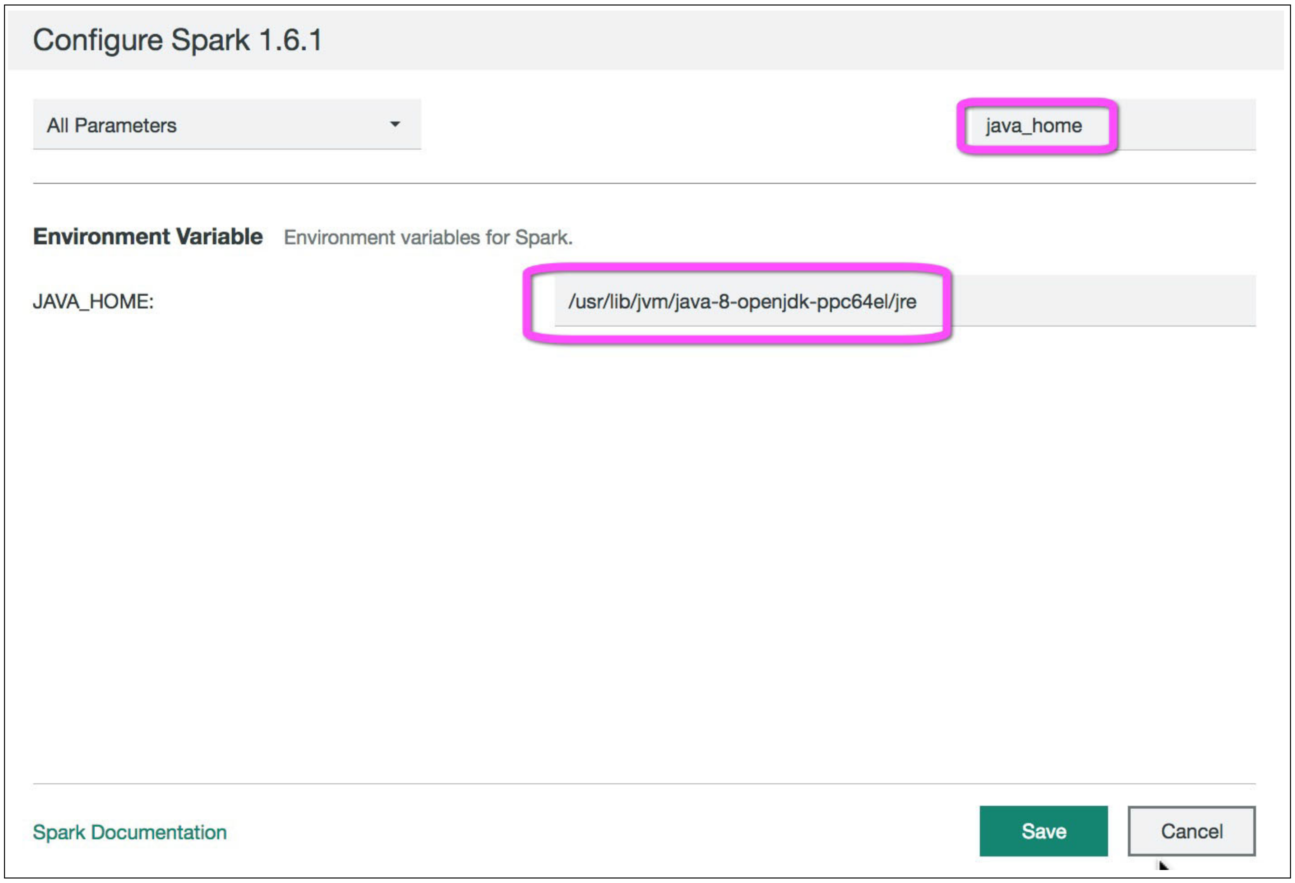
Spark version: Spark 1.6.1

Enable notebooks:
- ☐ JupyterPowerPython3 5.0.0
- ☐ JupyterPython3 5.0.0
- ☐ Zeppelin 0.7.2

High availability recovery directory: /sharefs/sig_share

Figure 6-22 Creating Spark Instance Group: 4

Note: If a user wants to enable the notebook tool for the SIG environment, select one of the three choices that are shown in Figure 6-22.


Configure Spark 1.6.1

All Parameters

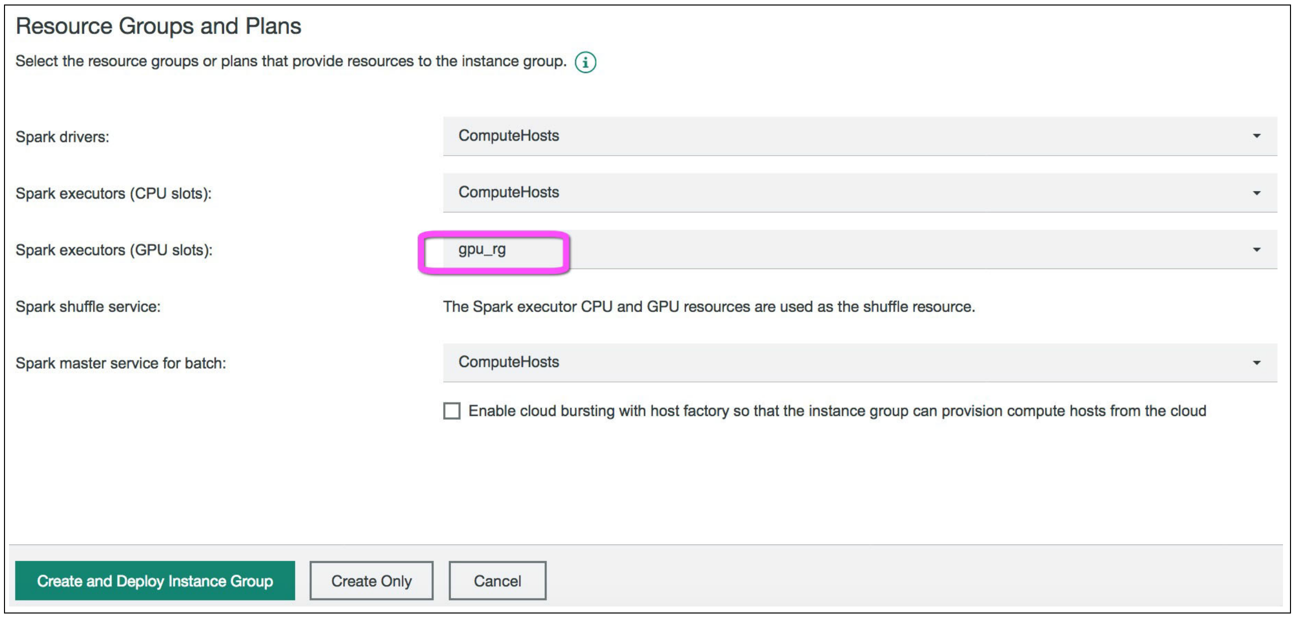
java_home

Environment Variable Environment variables for Spark.

JAVA_HOME: /usr/lib/jvm/java-8-openjdk-ppc64el/jre

Figure 6-23 Set JAVA_HOME variable in the Spark configuration

Note: The required value of `JAVA_HOME` depends on the actual environment on which the installation is being installed.


Resource Groups and Plans

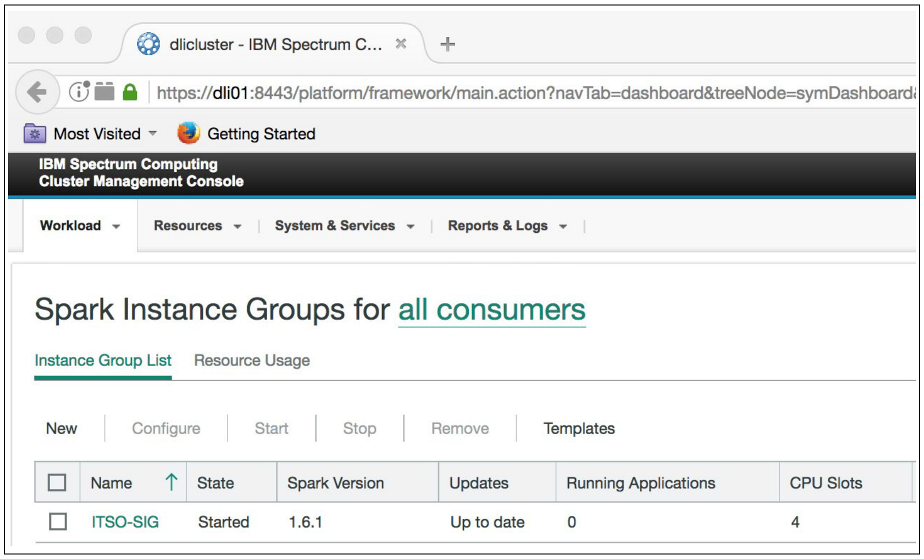
Select the resource groups or plans that provide resources to the instance group.

Spark drivers:	ComputeHosts
Spark executors (CPU slots):	ComputeHosts
Spark executors (GPU slots):	gpu_rg
Spark shuffle service:	The Spark executor CPU and GPU resources are used as the shuffle resource.
Spark master service for batch:	ComputeHosts

☐ Enable cloud bursting with host factory so that the instance group can provision compute hosts from the cloud

Figure 6-24 Assigning GPU resources to SIG

5. Click **Create and Deploy Instance Group** and continue to start this SIG service. When this SIG is successfully created, you see the status as Started, as shown in Figure 6-25.



Spark Instance Groups for all consumers

	Name	State	Spark Version	Updates	Running Applications	CPU Slots
☐	ITSO-SIG	Started	1.6.1	Up to date	0	4

Figure 6-25 Checking the SIG status

The status of existing SIG services can be queried from the command line, as shown in Example 6-5.

Example 6-5 Checking the SIG services from the command line

```
root@dli01:/opt/SIG# egosh service list
SERVICE  STATE    ALLOC CONSUMER RGROUP RESOURCE SLOTS SEQ_NO INST_STATE ACTI
elk-ela* STARTED  257   /Managem*Manage*dli01.d* 1     1      RUN        641
elk-man* STARTED  279   /Managem*Manage*dli01.d* 1     1      RUN        637
purger   STARTED  242   /Managem*Manage*dli01.d* 1     1      RUN        642
WEBGUI   STARTED  254   /Managem*Manage*dli01.d* 1     1      RUN        643
GPFSmon* DEFINED        /GPFSmon*Manage*
```

```
elk-ela* STARTED  258   /Managem*Manage*dli01.d* 1      1       RUN         624
                                 dli02.d* 1      2       RUN         654
plc      STARTED  243   /Managem*Manage*dli01.d* 1      1       RUN         644
elk-shi* STARTED  283   /Cluster*Intern*dli01.d* 1      3       RUN         657
                                 dli02.d* 1      1       RUN         655
                                 dli03.d* 1      2       RUN         656
elk-ind* STARTED  281   /Managem*Manage*dli01.d* 1      1       RUN         640
                                 dli02.d* 1      2       RUN         652
derbydb  DEFINED        /Managem*Manage*
elk-ela* STARTED  259   /Managem*Manage*dli01.d* 1      1       RUN         623
                                 dli02.d* 1      2       RUN         653
ITSO-SI* STARTED  307   /ITSO-SI*        dli03.d* 1      1       RUN         700
ITSO-SI* STARTED  308   /ITSO-SI*Comput*dli03.d* 3      1       RUN         701
                                                 2       RUN         702
                                                 3       RUN         703
mongod   STARTED  244   /Managem*Manage*dli01.d* 1      1       RUN         630
dlmao-o* STARTED  245   /Managem*Manage*dli01.d* 1      1       RUN         631
redis    STARTED  246   /Managem*Manage*dli01.d* 1      1       RUN         632
dlpd     STARTED  247   /Managem*Manage*dli01.d* 1      1       RUN         633
dlmao-m* STARTED  261   /Managem*Manage*dli01.d* 1      1       RUN         634
REST     STARTED  248   /Managem*Manage*dli01.d* 1      1       RUN         635
HostFac* DEFINED        /Managem*Manage*
RS       STARTED  249   /Managem*Manage*dli01.d* 1      1       RUN         629
ascd     STARTED  250   /Managem*Manage*dli01.d* 1      1       RUN         636
ExecPro* STARTED  251   /Cluster*Intern*dli01.d* 1      1       RUN         620
                                 dli02.d* 1      2       RUN         651
                                 dli03.d* 1      3       RUN         562
Service* STARTED  252   /Managem*Manage*dli01.d* 1      1       RUN         645
WebServ* STARTED  255   /Managem*Manage*dli01.d* 1      1       RUN         646
SparkCl* STARTED  280   /Cluster*Intern*dli01.d* 1      2       RUN         639
                                 dli02.d* 1      3       RUN         650
                                 dli03.d* 1      1       RUN         638

The two service related with SIG is ITSO-SIG-sparkss and ITSO-SIG-sparkss.
```

6.5 Master node crashed when a workload is running

After the SIG is started, start a DL workload through the DLI GUI, such as importing a data set or a training model.

For more information about how to use the DLI for DL projects, see 6.8, "Use case: Using a Caffe Cifar-10 network with DLI" on page 212.

In this case, assume that a workload is running in a compute node, then the master node (dli01) crashes, and the workload keeps running. IBM Spectrum Conductor with Spark and DLI services are taken over by dli02, where you can continue to monitor and manage the workloads. The access website changes to `https://dli02.dli.com:8443`.

Note: At the time of writing, DLI HA does not provide a floating service IP.

Figure 6-26 illustrates the DLI HA takeover process.

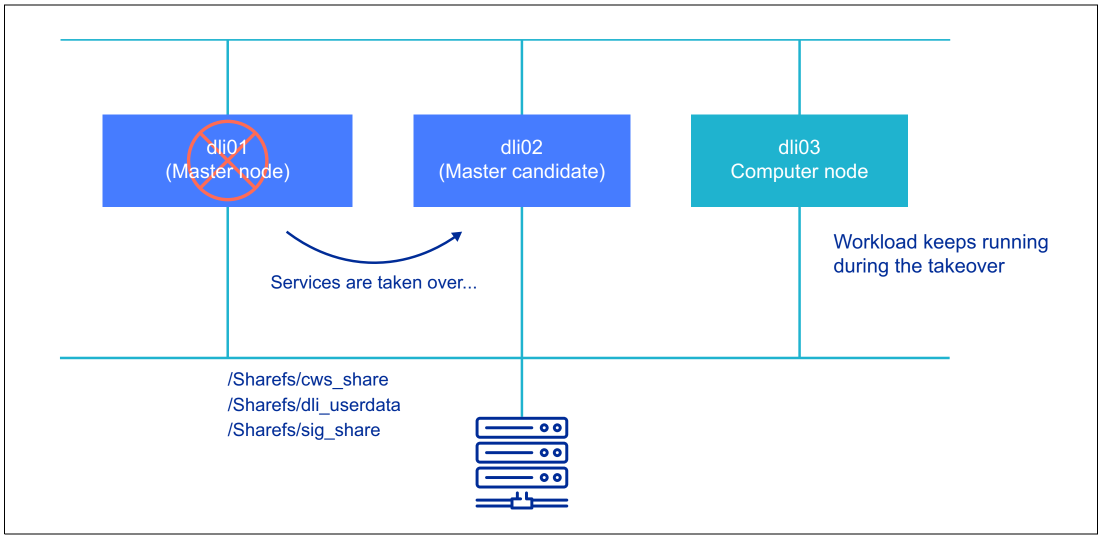


Figure 6-26 Failover scenario illustration

How to log in to the IBM Spectrum Computing Management Console

This section describes how to log in into the IBM Spectrum Computing Management Console.

Editing /etc/hosts on the notebook

You use an FQDN to access DLI, so you must add the IP and host name resolution into your hosts file. If you are using a Windows notebook, the file is in the `C:\Windows\System32\drivers\etc` directory. If you are using a Mac notebook, the file is in the `/etc` directory. In either case, add one line to the `/etc/hosts` file, as shown in Example 6-6.

Example 6-6 The /etc/hosts file configuration in the client

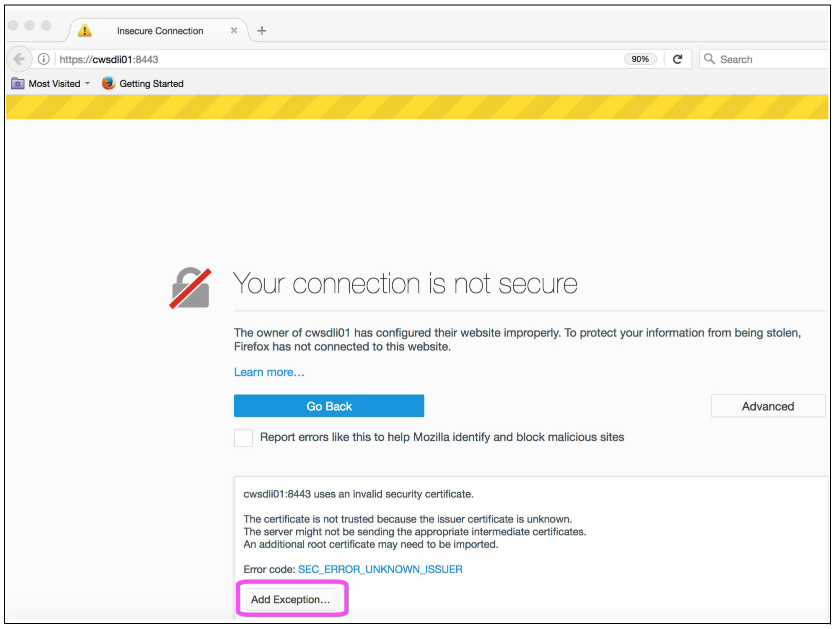
```
172.16.51.96                    dli01.dli.com
172.16.51.97                    dli02.dli.com
172.16.51.98                    dli03.dli.com
```

Getting the certification to access the DLI

There are two methods to get the certification to access DLI:

- Add an exception manually.

  You must add an exception to six ports: 5000, 50001, 8443, 8543, 8643, and 9243. For example, if you use Firefox browser, when you enter `https://dli01.dli.com:8443` and press Enter, the browser shows that it is an invalid security certification. Click **Add exception**, as shown in Figure 6-27. Then, a window opens and prompts you to confirm the security exception, as shown in Figure 6-28 on page 179.



The owner of cwsdli01 has configured their website improperly. To protect your information from being stolen, Firefox has not connected to this website.

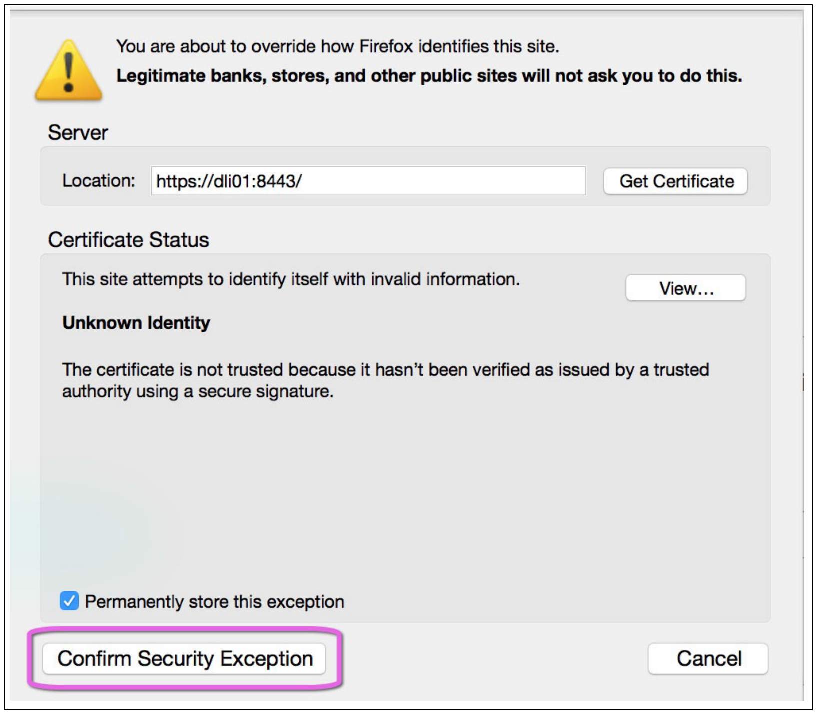
Report errors like this to help Mozilla identify and block malicious sites

cwsdli01:8443 uses an invalid security certificate.

The certificate is not trusted because the issuer certificate is unknown.
The server might not be sending the appropriate intermediate certificates.
An additional root certificate may need to be imported.

Error code: SEC_ERROR_UNKNOWN_ISSUER

Figure 6-27 Add exception window: 1


You are about to override how Firefox identifies this site.
Legitimate banks, stores, and other public sites will not ask you to do this.

Server

Location: https://dli01:8443/

Certificate Status

This site attempts to identify itself with invalid information.

Unknown Identity

The certificate is not trusted because it hasn't been verified as issued by a trusted authority using a secure signature.

☑ Permanently store this exception

Figure 6-28 Add exception window: 2

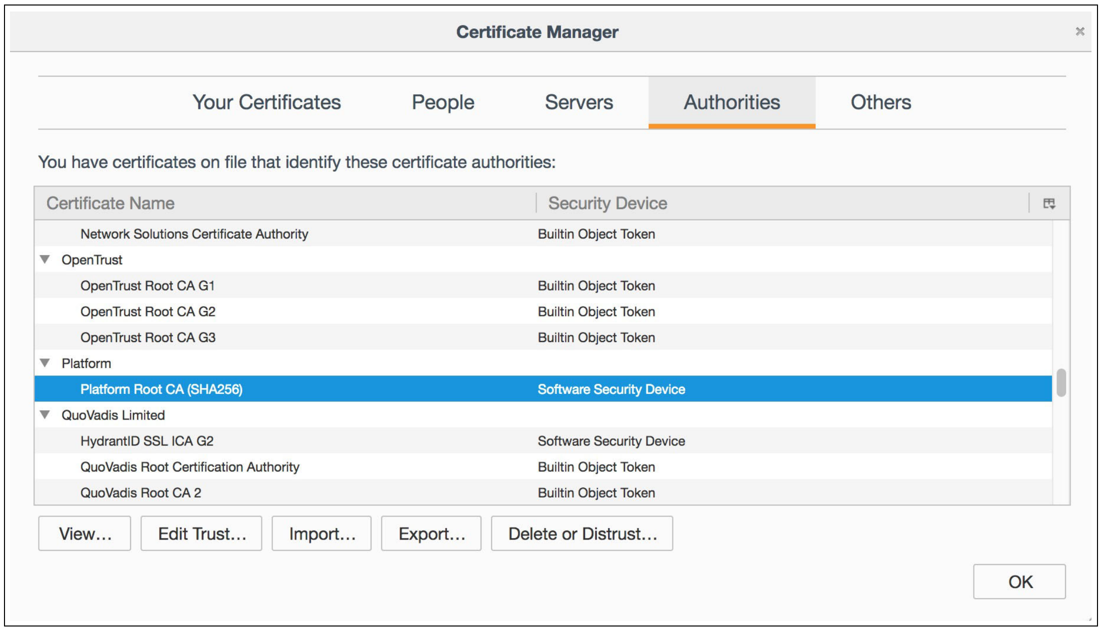
- Get the certification file and import it into the browser

  Proceed to download the certificate from the management host from `/opt/ibm/spectrumcomputing/security/cacert.pem`.

Then, open Firefox and complete the following steps:

1. Click **Tools** →**Options** →**Advanced** →**Certificates** →**View**.
2. Click the **Authorities** tab and click **Import**.
3. Browse to the location where you saved the certificate file and select it.

4. When you are prompted to trust a new CA, ensure that you select **Trust this CA to identify websites** and click **OK**. You can view the certification, as shown in Figure 6-29.



You have certificates on file that identify these certificate authorities:

Certificate Name	Security Device
Network Solutions Certificate Authority	Builtin Object Token
▼ OpenTrust	
OpenTrust Root CA G1	Builtin Object Token
OpenTrust Root CA G2	Builtin Object Token
OpenTrust Root CA G3	Builtin Object Token
▼ Platform	
Platform Root CA (SHA256)	Software Security Device
▼ QuoVadis Limited	
HydrantID SSL ICA G2	Software Security Device
QuoVadis Root Certification Authority	Builtin Object Token
QuoVadis Root CA 2	Builtin Object Token

Figure 6-29 Viewing the certification file

5. Restart the browser.

Logging in to IBM Spectrum Computing Cluster Management Console

Figure 6-30 shows the login window of the IBM Spectrum Computing Cluster Management Console. The default user name and password is Admin.

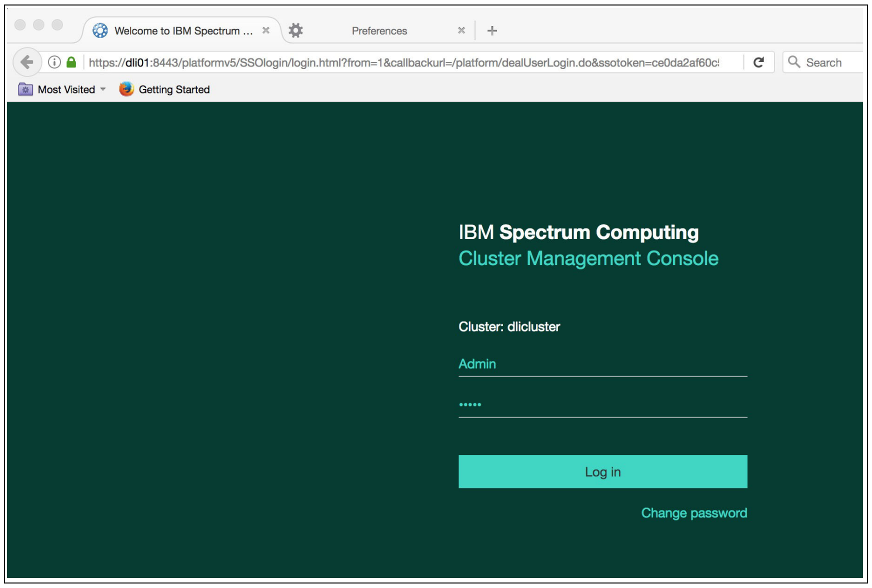

Welcome to IBM Spectrum …
Preferences
https://dli01:8443/platformv5/SSOlogin/login.html?from=1&callbackurl=/platform/dealUserLogin.do&ssotoken=ce0da2af60c
Search
Most Visited
Getting Started
IBM Spectrum Computing
Cluster Management Console
Cluster: dlicluster
Admin
•••••
Log in
Change password

Figure 6-30 Login window for Spectrum Computing Cluster Management Console

Click **Login** and you can see the cluster console main window.

Configuring the Master and Failover list

After you log in to the IBM Spectrum Computing Cluster Management Console, complete the following steps:

1. Click **System & Services** →**Cluster** →**Master and Failover**, as shown in Figure 6-31.

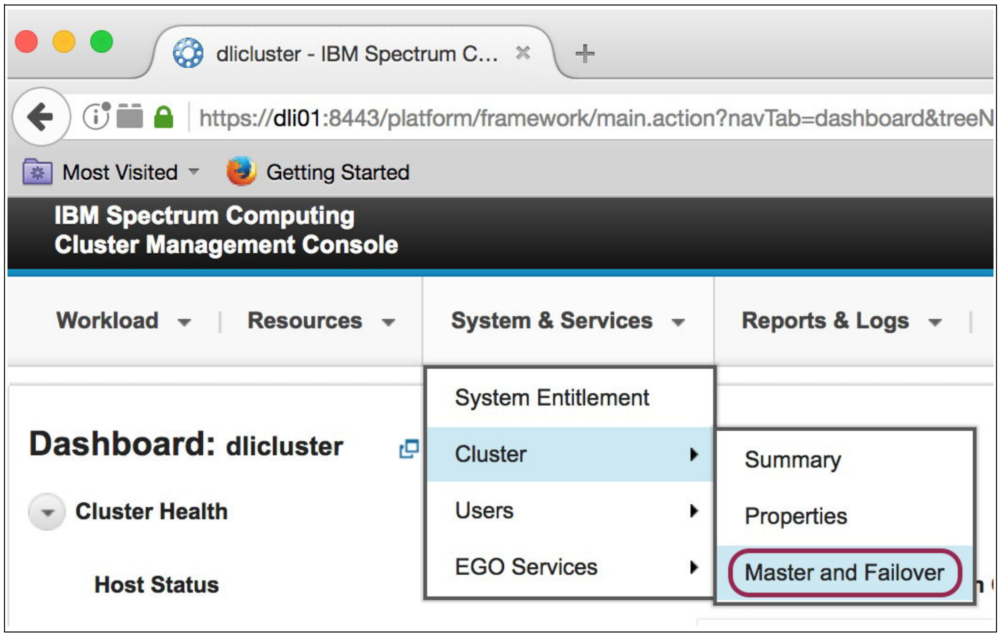


Figure 6-31 Master and Failover configuration for IBM Spectrum Conductor with Spark: 1

2. You can see that the dli02 node is in the available host list. Select it and add it to the Master candidates list, as shown in Figure 6-32.

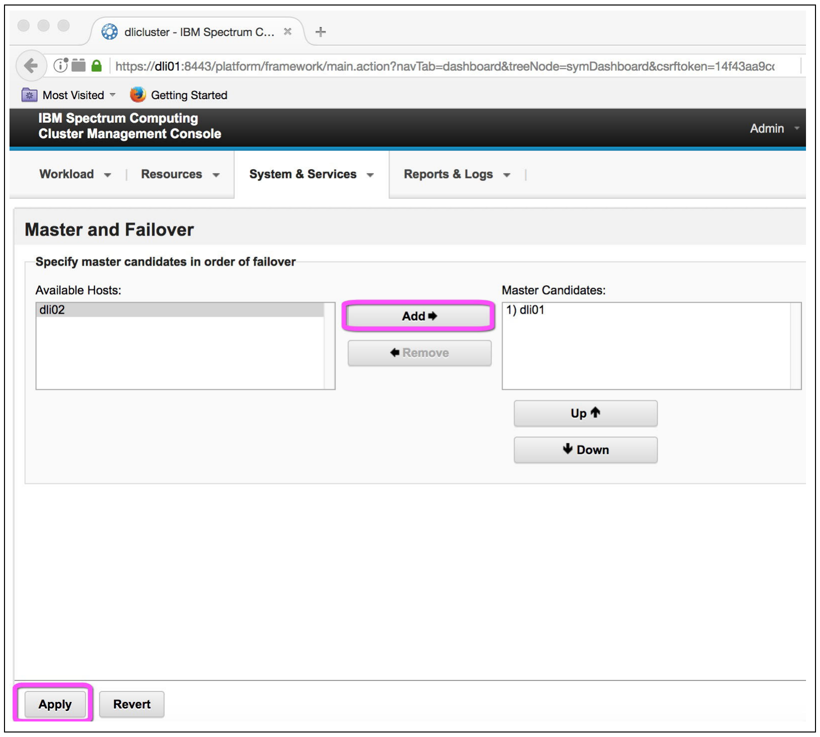

dlicluster - IBM Spectrum C...
https://dli01:8443/platform/framework/main.action?navTab=dashboard&treeNode=symDashboard&csrftoken=14f43aa9cc
Most Visited
Getting Started
IBM Spectrum Computing Cluster Management Console
Admin
Workload
Resources
System & Services
Reports & Logs
Master and Failover
Specify master candidates in order of failover
Available Hosts:
dli02
Add
Remove
Master Candidates:
1) dli01
Up
Down
Apply
Revert

Figure 6-32 Master and Failover configuration for IBM Spectrum Conductor with Spark: 2

3. Click **Apply**. Restart the cluster by clicking **OK**, as shown in Figure 6-33.

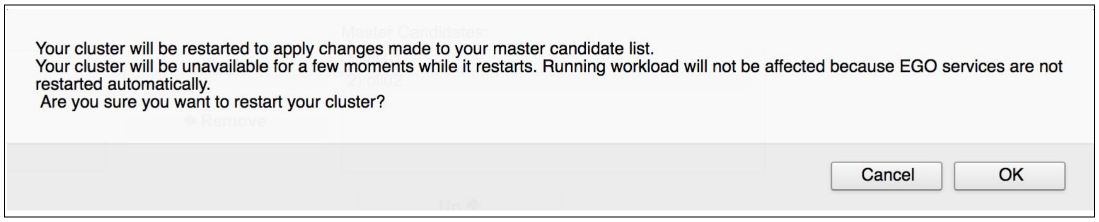

Your cluster will be restarted to apply changes made to your master candidate list.
Your cluster will be unavailable for a few moments while it restarts. Running workload will not be affected because EGO services are not restarted automatically.
Are you sure you want to restart your cluster?

Figure 6-33 Master and Failover configuration for IBM Spectrum Conductor with Spark: 3

6.6 Introduction to DLI graphic user interface

This section describes how to use DLI functions. The environment is one IBM Power Systems VM with four POWER8 CPU cores and one NVIDIA P100 GPU. The `/etc/hosts` file is shown in Example 6-7.

Example 6-7 Contents of /etc/hosts

```
172.16.51.100 cwsdli01.dli.com   cwsdli01
```

The client is one Mac notebook.

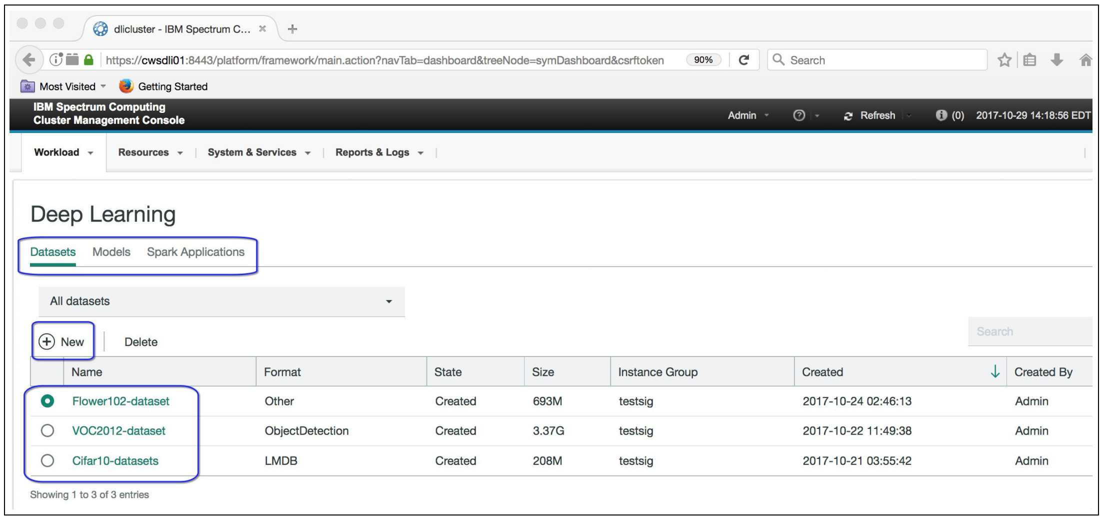
The IBM Spectrum Conductor with Spark software is installed in the default directory `/opt/ibm/spectrumcomputing`, and DLI's user data is installed in the `/scratch/dli_userdata` directory.

6.6.1 Data set management

To start managing your data sets, complete the following steps:

1. When you log in to IBM Spectrum Conductor with Spark, select **Workload** →**Spark** → **Deep Learning**. The main window of DLI opens, as shown in Figure 6-34.

   There are three menus: Data sets, Models, and Spark Applications.



Name	Format	State	Size	Instance Group	Created	Created By
Flower102-dataset	Other	Created	693M	testsig	2017-10-24 02:46:13	Admin
VOC2012-dataset	ObjectDetection	Created	3.37G	testsig	2017-10-22 11:49:38	Admin
Cifar10-datasets	LMDB	Created	208M	testsig	2017-10-21 03:55:42	Admin

Showing 1 to 3 of 3 entries

Figure 6-34 DLI main window

Data set preparation is the first step to train or tuning a model in DLI. The data set management function helps you to create, import, and remove the data sets.

When you start a data set task, there is a Spark task that is created and running on IBM Spectrum Conductor with Spark SIG. DLI keeps monitoring the task status until it completes. The data set management tool calls the DL framework (Caffe, TensorFlow, or others) APIs to create the data sets.

2. Click **New** →**New data set** and a window opens, as shown in Figure 6-35. This window shows the supported data set type in the current DLI version.

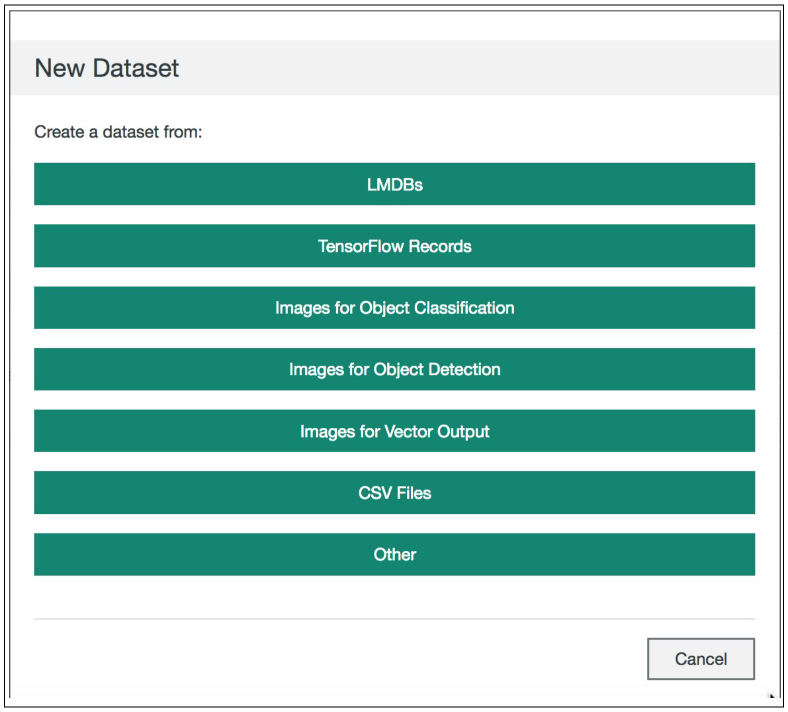
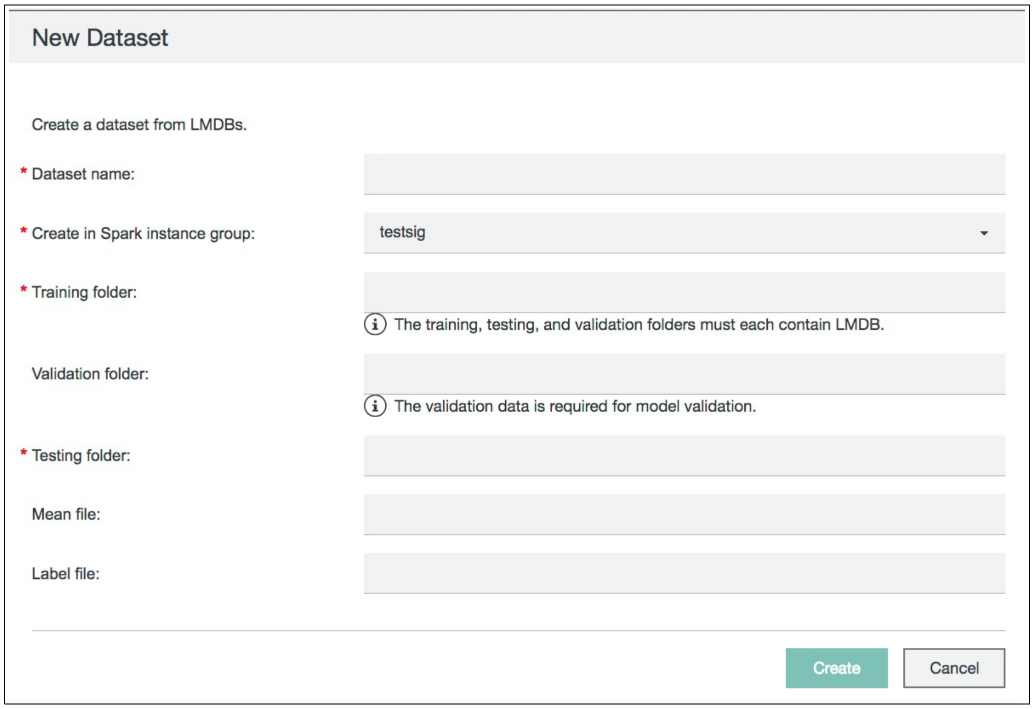


Figure 6-35 Selecting the data type for data set importing

3. The data set management function supports LMDB, TFRecords, CSV, Image for classification, Image for object detection, Image for Vector Output, CSV format, and Other. Other means DLI imports this data directly and does not perform any operations such as shuffle or resize, which requires you to prepare the data set before importing it.

For example, click LMDB. The New data set window opens, where you enter parameters. Regarding the detailed requirement for each item, there is one pop-up window to introduce the message, as shown in Figure 6-36.



New Dataset

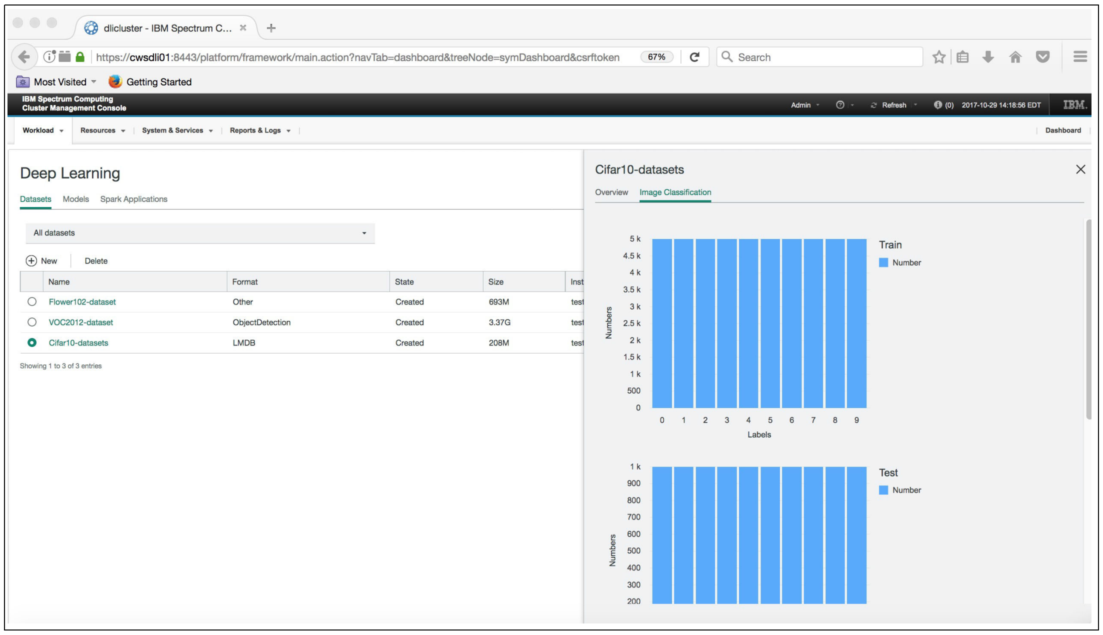
Create a dataset from LMDBs.

* Dataset name:	
* Create in Spark instance group:	testsig
* Training folder:	The training, testing, and validation folders must each contain LMDB.
Validation folder:	The validation data is required for model validation.
* Testing folder:	
Mean file:	
Label file:	

Create Cancel

Figure 6-36 Importing the LMDB data set window

4. After entering all the parameters, click **Create** and DLI imports the data into its working directory, which is in `${DLI_SHARED_FS}/datasets`. After this operation completes, you can view the result. Click the data set name and a window opens that shows the details (Figure 6-37).



https://cwsdli01:8443/platform/framework/main.action?navTab=dashboard&treeNode=symDashboard&csrftoken

Deep Learning

Datasets Models Spark Applications

All datasets

	Name	Format	State	Size	Inst
○	Flower102-dataset	Other	Created	693M	test
○	VOC2012-dataset	ObjectDetection	Created	3.37G	test
●	Cifar10-datasets	LMDB	Created	208M	test

Showing 1 to 3 of 3 entries

Figure 6-37 Viewing the result of the data set import

Note: The message in the window can be different for different data set types.

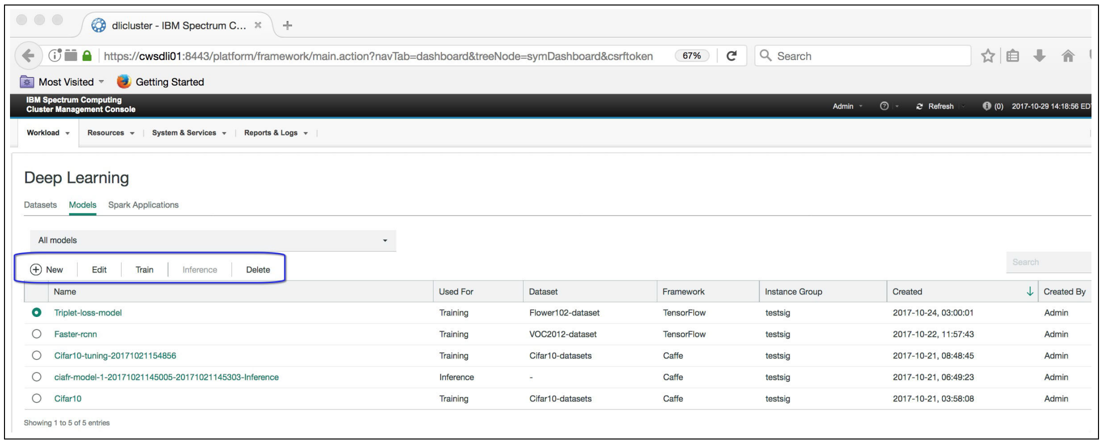
6.6.2 Model management

This section introduces the DL model management. To start, in the Select Models menu, you can see five submenus: New, Edit, Train, Inference, and Delete.

Model template and model creation

To begin learning about the model templates and model creation, complete the following steps:

1. Click **New** to create a model template or model, as shown in Figure 6-38.

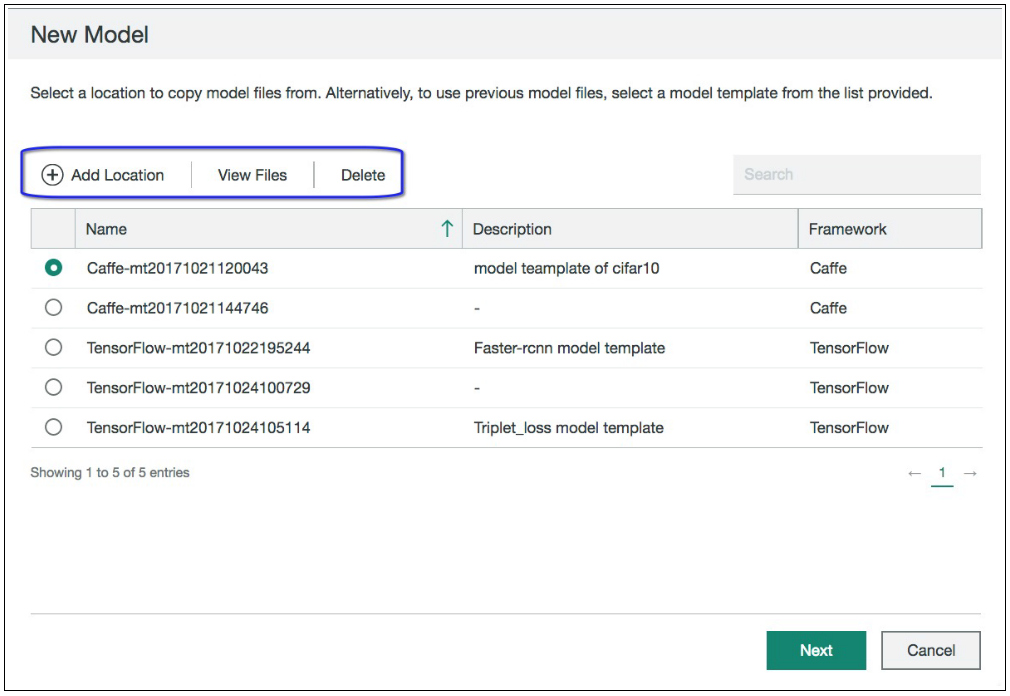

dlicluster - IBM Spectrum C...
https://cwsdli01:8443/platform/framework/main.action?navTab=dashboard&treeNode=symDashboard&csrftoken
67%
Search
Most Visited
Getting Started
IBM Spectrum Computing Cluster Management Console
Admin
Refresh
(0) 2017-10-29 14:18:56 ED
Workload
Resources
System & Services
Reports & Logs
Deep Learning
Datasets
Models
Spark Applications
All models
New
Edit
Train
Inference
Delete
Search

	Name	Used For	Dataset	Framework	Instance Group	Created	Created By
●	Triplet-loss-model	Training	Flower102-dataset	TensorFlow	testsig	2017-10-24, 03:00:01	Admin
○	Faster-rcnn	Training	VOC2012-dataset	TensorFlow	testsig	2017-10-22, 11:57:43	Admin
○	Cifar10-tuning-20171021154856	Training	Cifar10-datasets	Caffe	testsig	2017-10-21, 08:48:45	Admin
○	ciafr-model-1-20171021145005-20171021145303-Inference	Inference	-	Caffe	testsig	2017-10-21, 06:49:23	Admin
○	Cifar10	Training	Cifar10-datasets	Caffe	testsig	2017-10-21, 03:58:08	Admin

Showing 1 to 5 of 5 entries

Figure 6-38 Creating a model template

Figure 6-39 displays the existing model template. You can add, view, or delete a model template. If you want to create a DL model, create a model template first or use an existing one.


New Model

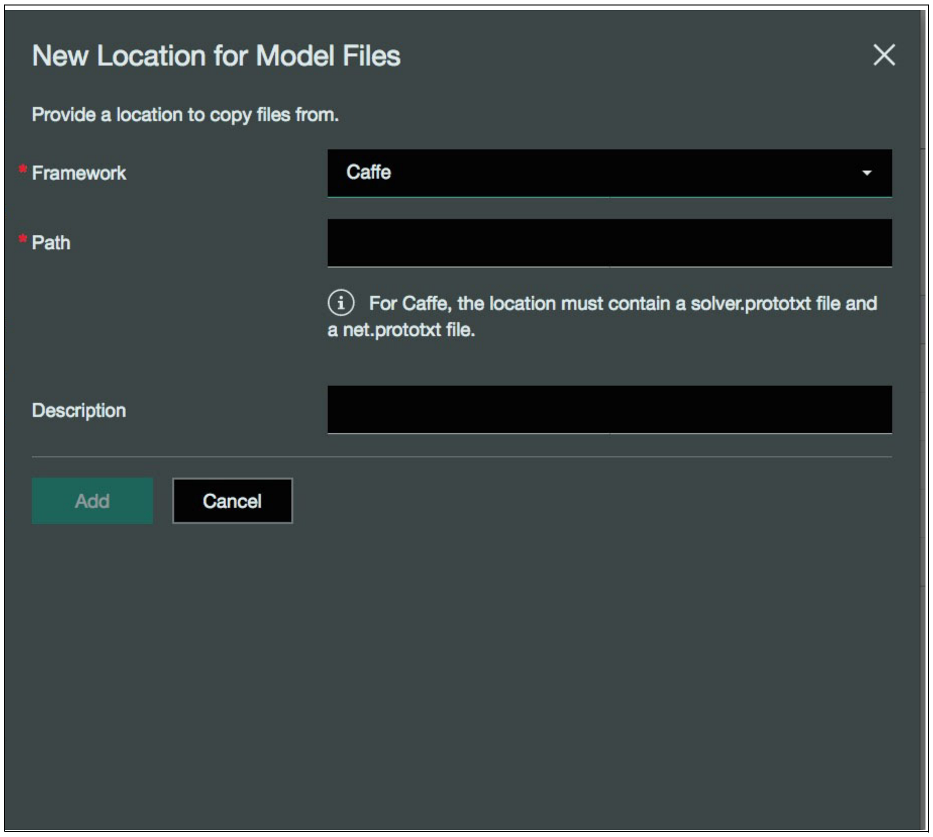
Select a location to copy model files from. Alternatively, to use previous model files, select a model template from the list provided.

	Name	Description	Framework
●	Caffe-mt20171021120043	model teamplate of cifar10	Caffe
○	Caffe-mt20171021144746	-	Caffe
○	TensorFlow-mt20171022195244	Faster-rcnn model template	TensorFlow
○	TensorFlow-mt20171024100729	-	TensorFlow
○	TensorFlow-mt20171024105114	Triplet_loss model template	TensorFlow

Showing 1 to 5 of 5 entries

Figure 6-39 Adding a model template

2. To add a model template, click **Add Location.** Complete the path and description, and click **Add** to create it. For the framework parameter, at the time of writing, you can choose Caffe or TensorFlow. The path points to the folder of model template files that you want to create, as shown in Figure 6-40 on page 187.


New Location for Model Files

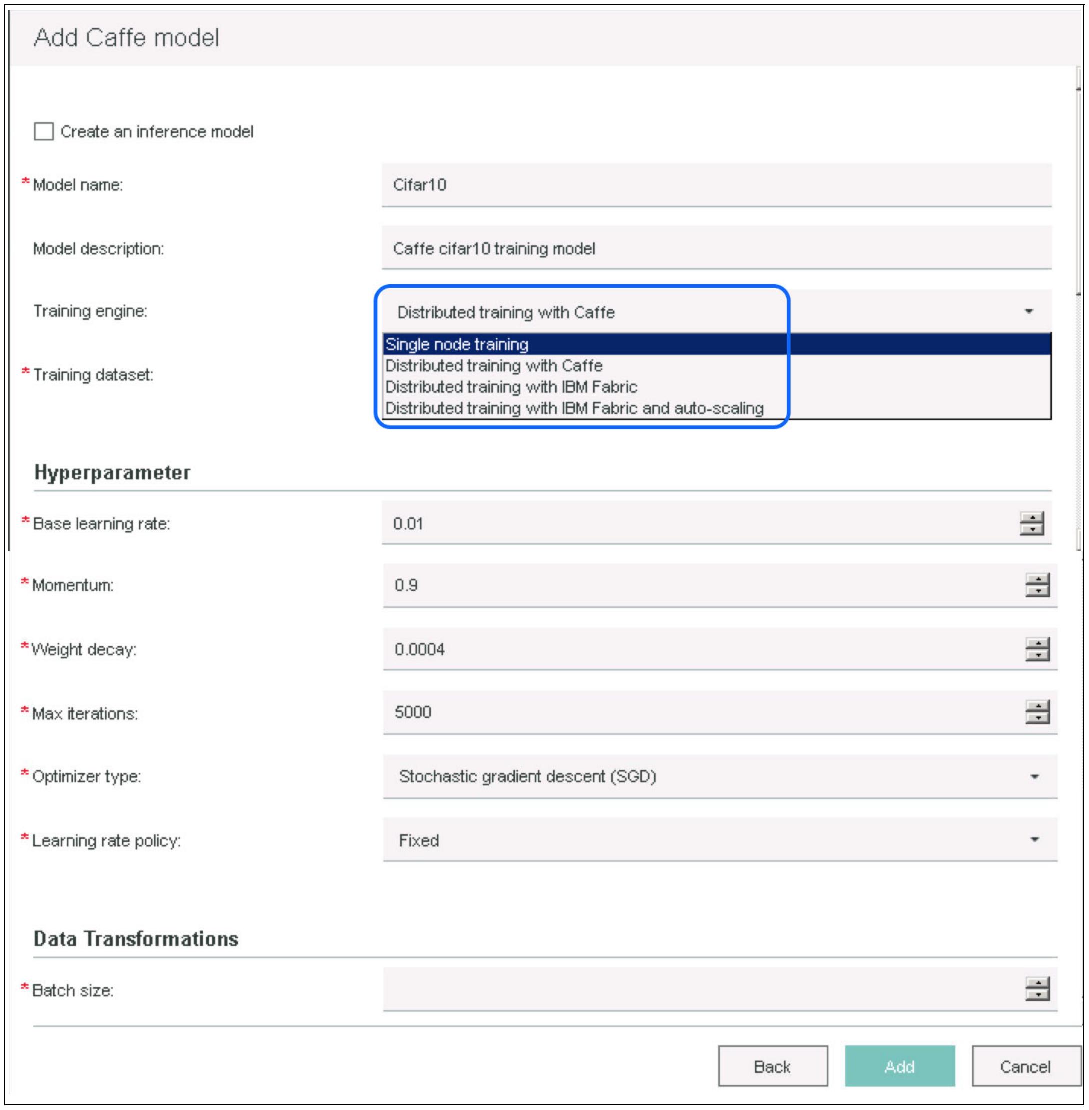
Provide a location to copy files from.

* Framework	Caffe
* Path	
	For Caffe, the location must contain a solver.prototxt file and a net.prototxt file.
Description	

Figure 6-40 Adding the new location for model files

Note: You can also edit or remove the model template.

3. Select a model template and click **Next** to create a model. Figure 6-41 shows an example of creating a Caffe model.


Add Caffe model

☐ Create an inference model

* Model name:	Cifar10
Model description:	Caffe cifar10 training model
Training engine:	Distributed training with Caffe
* Training dataset:	

Single node training
Distributed training with Caffe
Distributed training with IBM Fabric
Distributed training with IBM Fabric and auto-scaling

Hyperparameter

* Base learning rate:	0.01
* Momentum:	0.9
* Weight decay:	0.0004
* Max iterations:	5000
* Optimizer type:	Stochastic gradient descent (SGD)
* Learning rate policy:	Fixed

Data Transformations

* Batch size:	

Figure 6-41 Creating a Caffe model

If you want to create a training mode, the Create an inference model must be cleared.

There is one attribution, which is named Training engine, for the Caffe model, with four options:

– Single node training
– Distributed training with Caffe
– Distributed training with IBM Fabric
– Distributed training with IBM Fabric and auto-scaling

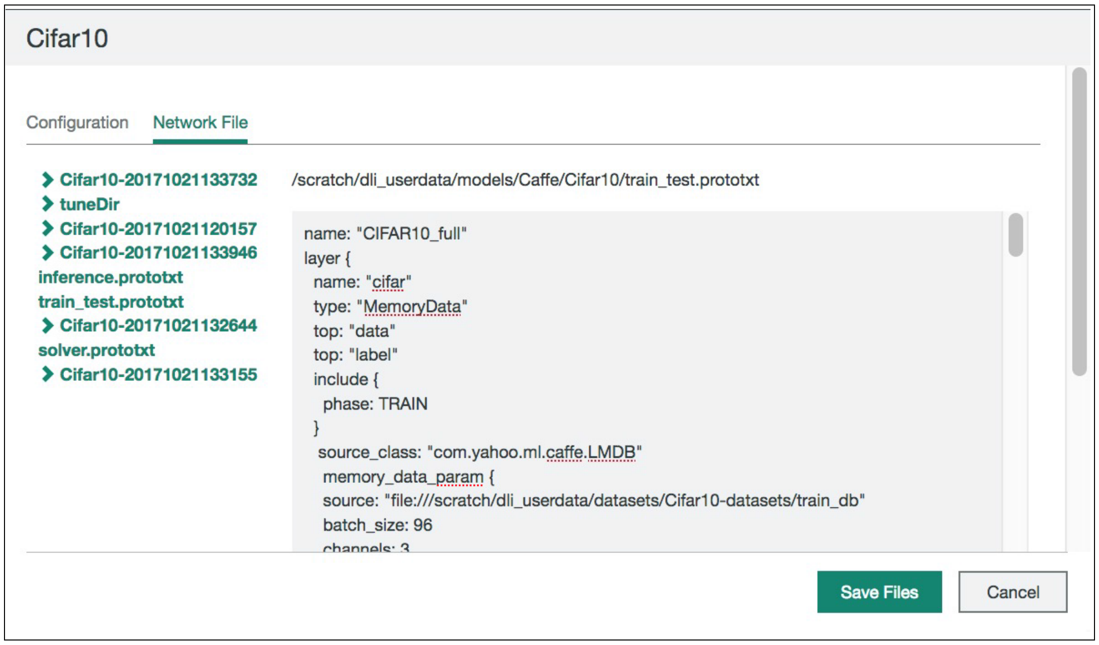
For the TensorFlow model, there are also four options:

– Single node training
– Distributed training with Caffe
– Distributed training with IBM Fabric
– Distributed training with IBM Fabric and auto-scaling

For more information about these options, see 6.7, “Supported deep learning network and training engine in DLI” on page 203.

Note: To create an inference mode with an existing model weight file, select the **Create an inference mode** check box and complete the weight file path.

4. After completing all the required parameters, click **Add** to create it. You can also edit it. There are two parts that can be edited: parameter configuration and network files, as shown in Figure 6-42.



/scratch/dli_userdata/models/Caffe/Cifar10/train_test.prototxt

```
name: "CIFAR10_full"
layer {
  name: "cifar"
  type: "MemoryData"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  source_class: "com.yahoo.ml.caffe.LMDB"
   memory_data_param {
   source: "file:///scratch/dli_userdata/datasets/Cifar10-datasets/train_db"
   batch_size: 96
   channels: 3
```

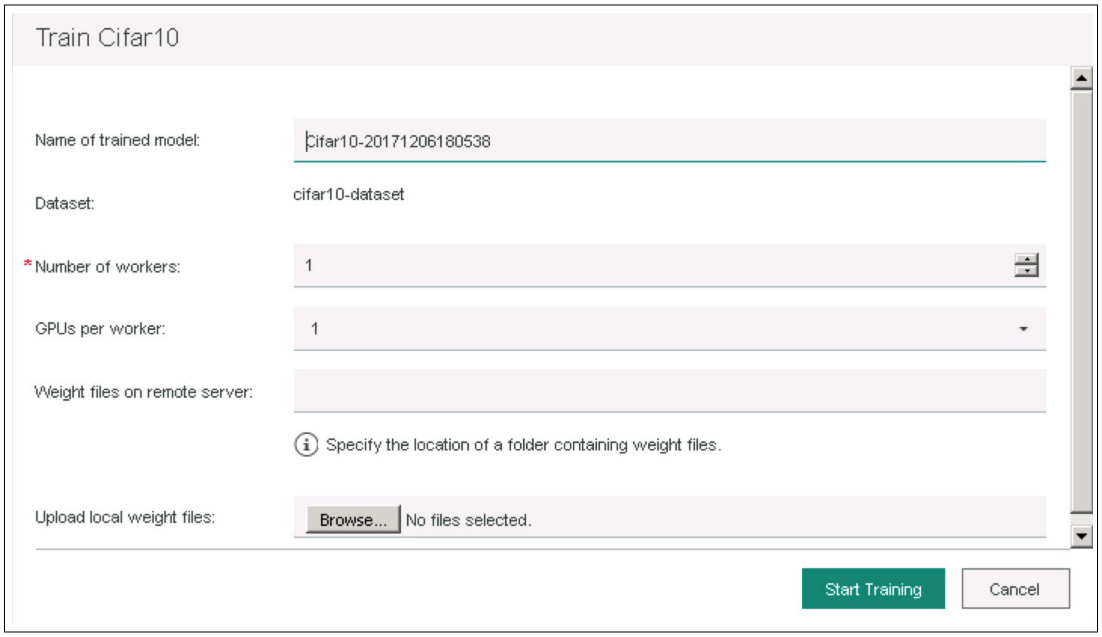
Figure 6-42 Editing the model’s network files in the DLI GUI

Model training management

To train your model, complete the following steps:

1. To start a training workload, select the model and click **Train**. A window opens and prompts you to complete parameters, such as worker number, GPUs per worker and others. For example:
   - Number of works means how many workers to start.
   - GPUs per worker means how many GPU to use for each worker.
   - Weight files on remote server is the folder path of the weight file on the remote server, which is used for continuous training.
   - Upload local weight files is the path of the weight file for uploading, which is also used for continuous training.

2. Click **Start Training** to start this job, as shown in Figure 6-43.


Train Cifar10

Name of trained model: Cifar10-20171206180538

Dataset: cifar10-dataset

* Number of workers: 1

GPUs per worker: 1

Weight files on remote server:

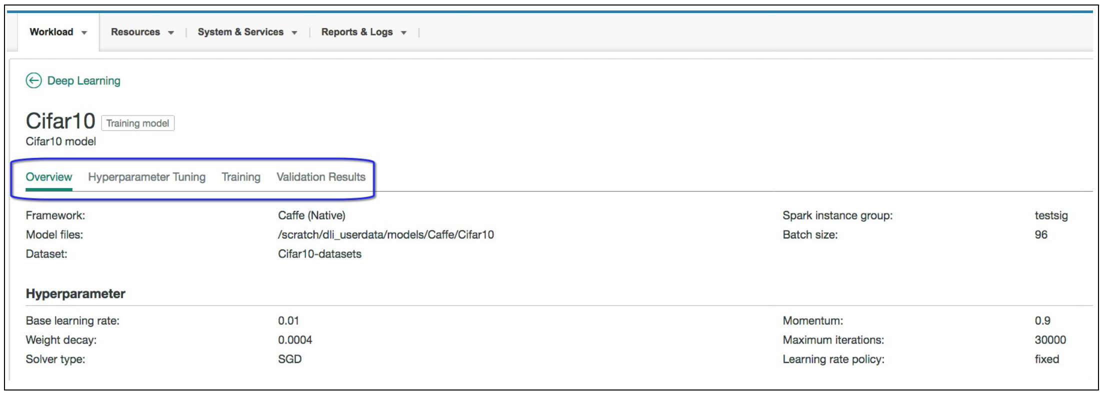
Specify the location of a folder containing weight files.

Upload local weight files: Browse... No files selected.

Figure 6-43 Start Training window

Note: When you select the **Distributed training with IBM Fabric and auto-scaling** option for the training model, there is a Max number of workers parameter. For more information about this parameter, see "Distributed training with IBM Fabric and auto-scaling" on page 206.

3. Click a model name and a new window opens (Figure 6-44). There are four menus: Overview, Hyperparameter Tuning, Training, and Validation results.



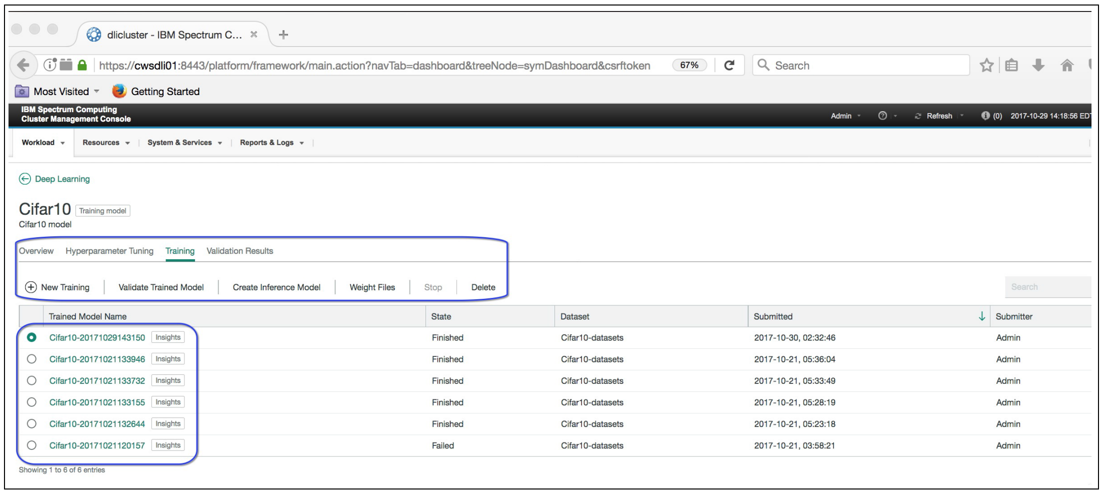
Framework: Caffe (Native)
Model files: /scratch/dli_userdata/models/Caffe/Cifar10
Dataset: Cifar10-datasets
Spark instance group: testsig
Batch size: 96

Hyperparameter

Base learning rate: 0.01
Weight decay: 0.0004
Solver type: SGD
Momentum: 0.9
Maximum iterations: 30000
Learning rate policy: fixed

Figure 6-44 Menus for one training or trained model

4. Click **Training** to open the training and trained model lists, as shown in Figure 6-45 on page 191.


https://cwsdli01:8443/platform/framework/main.action?navTab=dashboard&treeNode=symDashboard&csrftoken

IBM Spectrum Computing Cluster Management Console

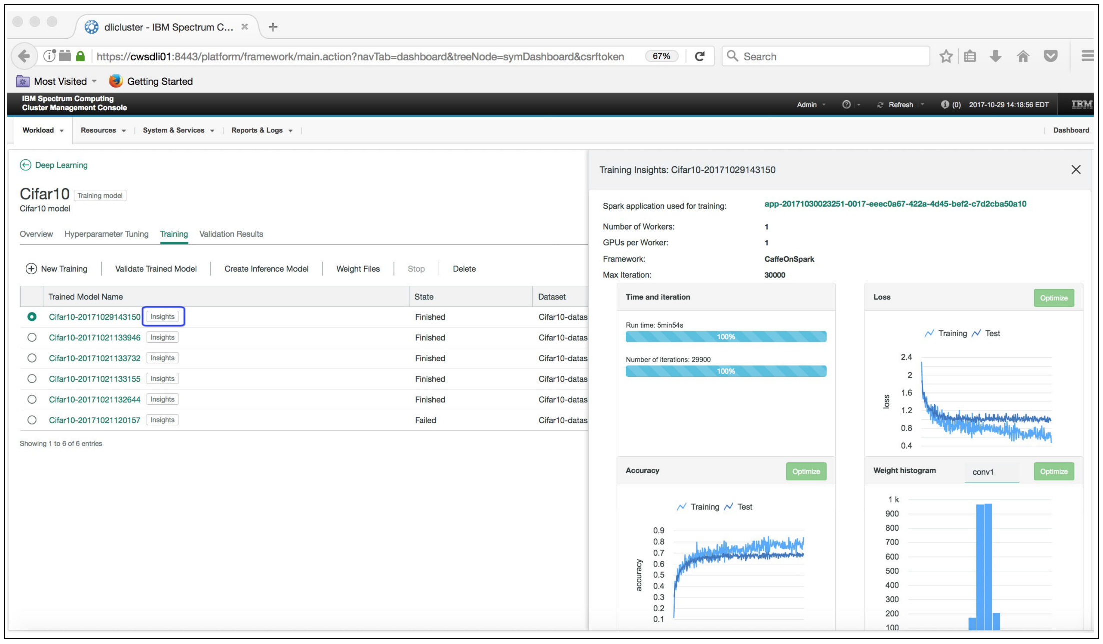
Cifar10 Training model
Cifar10 model

Trained Model Name	State	Dataset	Submitted	Submitter
Cifar10-20171029143150 Insights	Finished	Cifar10-datasets	2017-10-30, 02:32:46	Admin
Cifar10-20171021133946 Insights	Finished	Cifar10-datasets	2017-10-21, 05:36:04	Admin
Cifar10-20171021133732 Insights	Finished	Cifar10-datasets	2017-10-21, 05:33:49	Admin
Cifar10-20171021133155 Insights	Finished	Cifar10-datasets	2017-10-21, 05:28:19	Admin
Cifar10-20171021132644 Insights	Finished	Cifar10-datasets	2017-10-21, 05:23:18	Admin
Cifar10-20171021120157 Insights	Failed	Cifar10-datasets	2017-10-21, 03:58:21	Admin

Showing 1 to 6 of 6 entries

Figure 6-45 Listing all the trained or training jobs of one model

5. Select one training or trained model and click **Insights**. The MAO window opens, as shown in Figure 6-46.


Training Insights: Cifar10-20171029143150

Spark application used for training: app-20171030023251-0017-eeec0a67-422a-4d45-bef2-c7d2cba50a10

Number of Workers: 1

GPUs per Worker: 1

Framework: CaffeOnSpark

Max Iteration: 30000

Run time: 5min54s

Number of iterations: 29900

Figure 6-46 Monitoring and Optimization window for one training or trained model's job

The window includes the following messages:

- Elapsed time and iterations
- Training progress estimation
- Learning curves for loss in train and test net
- Learning curves for accuracy in train and test net
- Optimization suggestion
- Early stop warning
- Weight histogram for each interested layer
- Gradient histogram for each interested layer
- Activation histogram for each interested layer

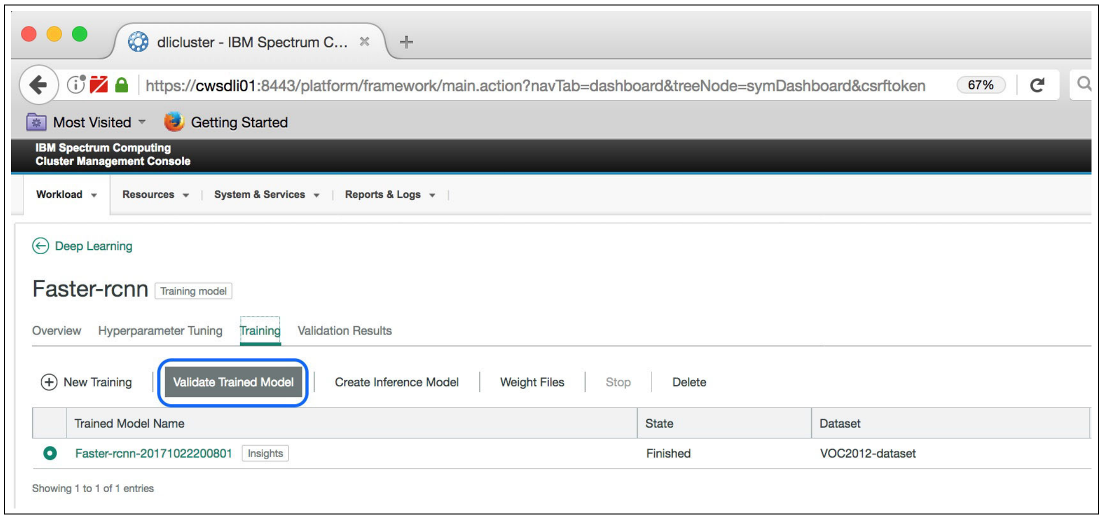
For the early stop warning, the following issues are detected and suggested during the training process:

- Gradient explosion
- Overflow
- Saturation
- Divergence
- Over fitting
- Under fitting

Model validation management

To validate the model, complete the following steps:

1. When a training workload is finished, select it and click **Validate Trained Model** to view the accuracy of the validation data set, as shown in Figure 6-47.

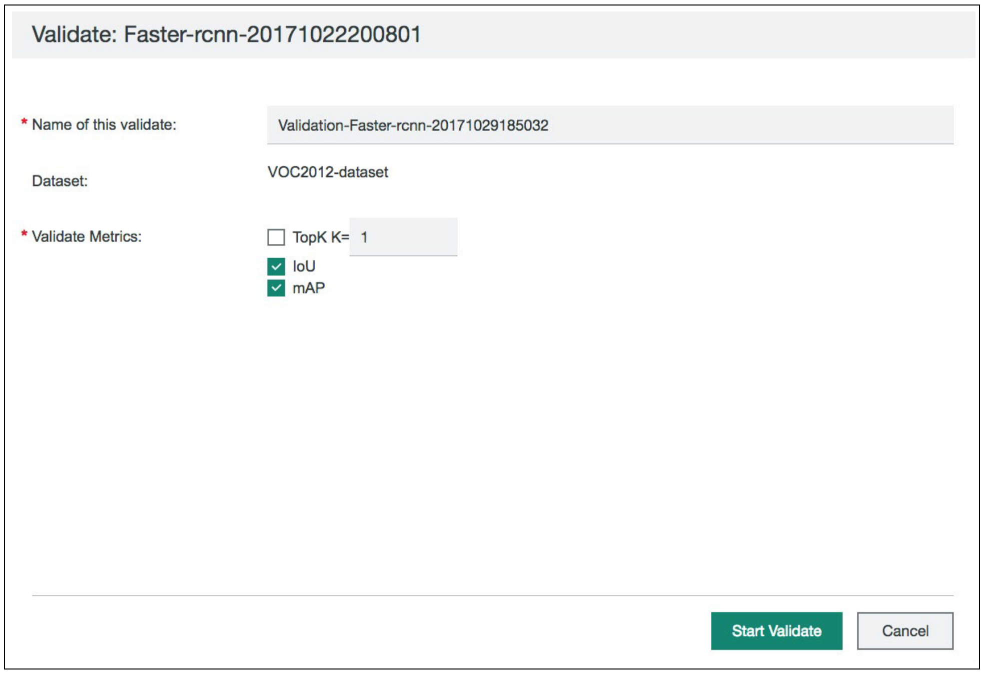


Trained Model Name	State	Dataset
Faster-rcnn-20171022200801 Insights	Finished	VOC2012-dataset

Showing 1 to 1 of 1 entries

Figure 6-47 Validating a trained model

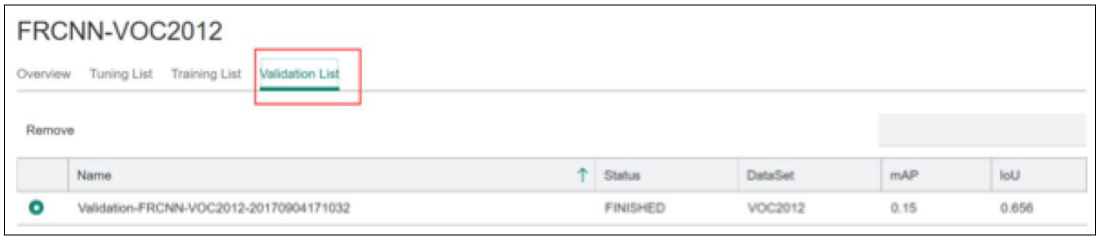
For the validation metrics, Top K is used for image classification, and others are used for object detection, as shown in Figure 6-48 on page 193.


Validate: Faster-rcnn-20171022200801

* Name of this validate:	Validation-Faster-rcnn-20171029185032
Dataset:	VOC2012-dataset
* Validate Metrics:	☐ TopK K= 1 ✓ IoU ✓ mAP

Figure 6-48 Selecting the validation method for one trained model

2. When validation is complete, select the trained model and click **Validation Results**. You get the validation result, as shown in Figure 6-49.


FRCNN-VOC2012

	Name	Status	DataSet	mAP	IoU
●	Validation-FRCNN-VOC2012-20170904171032	FINISHED	VOC2012	0.15	0.656

Figure 6-49 Viewing the validation result

Model hyperparameter tuning management

Hyperparameter tuning is a key feature in DLI. There are three definitions of it:

- Parameters tuning

  For a neural network to train, different parameters that are input to the model affect the training result, such as time cost or the accuracy. Parameters tuning helps to find relatively good parameters compared to random input manually.

- Tuning task

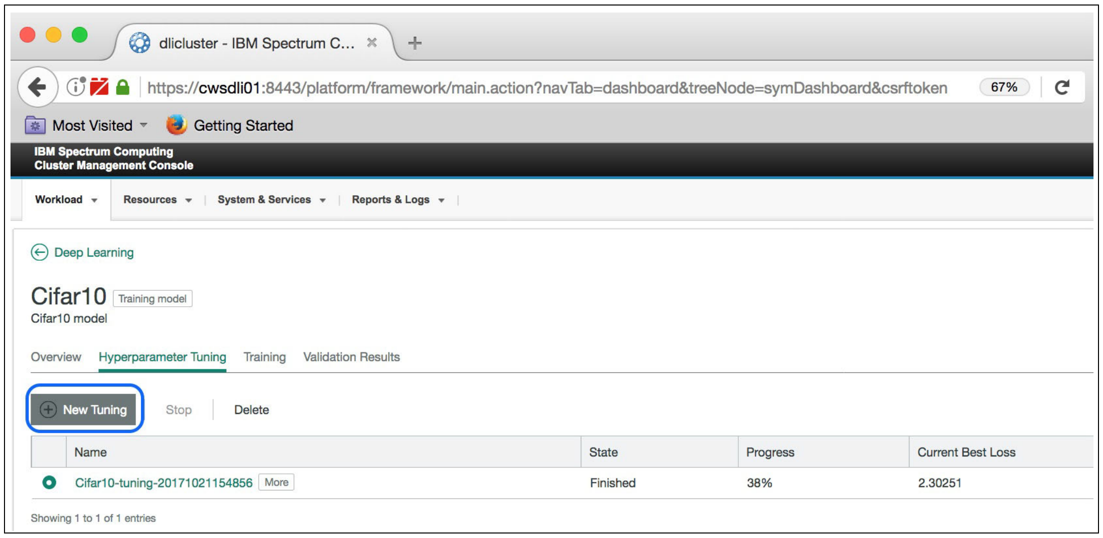
  One of the tasks that you start from UI to tune the hyperparameters for a model.

- Tuning job

  One tuning task gets many tuning jobs that run with a set of hyperparameters.

To tune the model hyperparameters, complete the following steps:

1. To run a hyperparameter search task, select a training model, click its name, and select **Hyperparameter Tuning** →**New Tuning** to start the configuration, as shown in Figure 6-50.



Name	State	Progress	Current Best Loss
Cifar10-tuning-20171021154856	Finished	38%	2.30251

Figure 6-50 Starting a hyperparameter tuning task

In the windows, you can set the parameters and policies for this tuning task. For the search type, DLI supports the policies Random Search, TPE, and Bayesian. The following section explains each one of them:

– Random Search

  Random Search algorithms generate random pairs of combinations by using the input hyperparameters range in the UI, and finds the best one of the tuning jobs results with these random combinations of hyperparameters.

– TPE

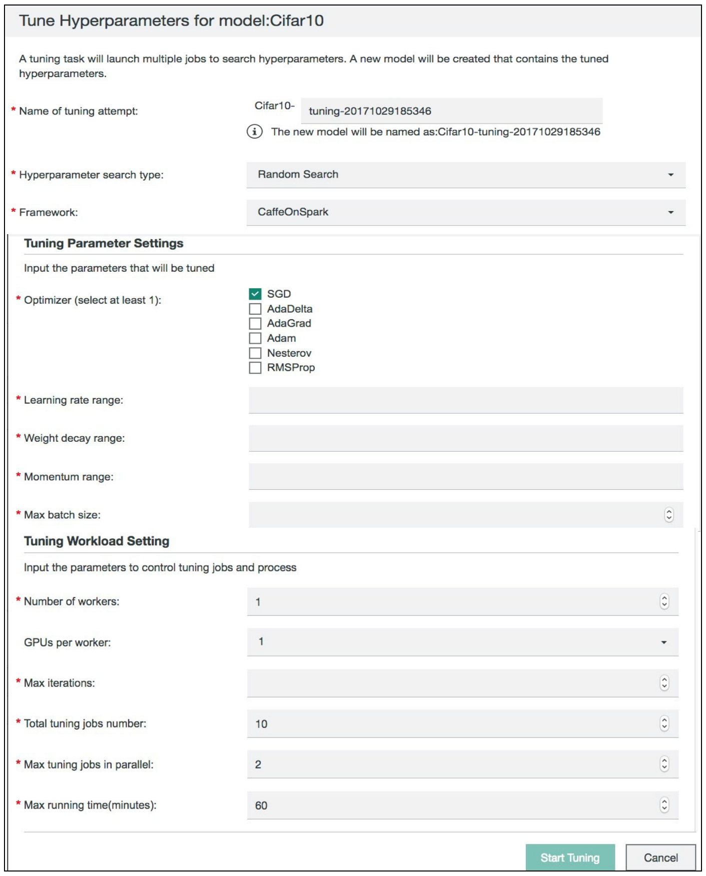
  Automatically chooses a group of better parameters for objective function optimization in ML by evaluating some partial tuning (training) jobs. In principle, it uses a Parzen window to distinguish the selected samples on each dimension of the parameters with a specified random number generator, then applies a Gaussian mixture model with the loss value to derive a new random number generator for new samples until finding some good parameters combination for the training session.

– Bayesian

  Runs partial tuning and training jobs with random chosen parameters in batches. With each batch, it calculates the possibility of *good parameters* corresponding to loss value and uses this possibility as a distribution that comes from theoretical deduction to estimate the Gaussian density of the good parameters. Eventually, it finds a good parameter combination by changing the distribution.

For more information about these policies, see 6.3.3, “Hyperparameter optimization and search” on page 147.

For learning rate, weight decay, and momentum parameters, you can use a range, such as 0.1 - 0.2. If you select Random Search as the search type, it also supports discrete point values, such as 0.1, 0.2, 0.3. For more information, see Figure 6-51 on page 195.



Tune Hyperparameters for model:Cifar10

A tuning task will launch multiple jobs to search hyperparameters. A new model will be created that contains the tuned hyperparameters.

* Name of tuning attempt:	Cifar10- tuning-20171029185346
	The new model will be named as:Cifar10-tuning-20171029185346
* Hyperparameter search type:	Random Search
* Framework:	CaffeOnSpark

Tuning Parameter Settings

Input the parameters that will be tuned

* Optimizer (select at least 1):	✓ SGD ☐ AdaDelta ☐ AdaGrad ☐ Adam ☐ Nesterov ☐ RMSProp
* Learning rate range:	
* Weight decay range:	
* Momentum range:	
* Max batch size:	

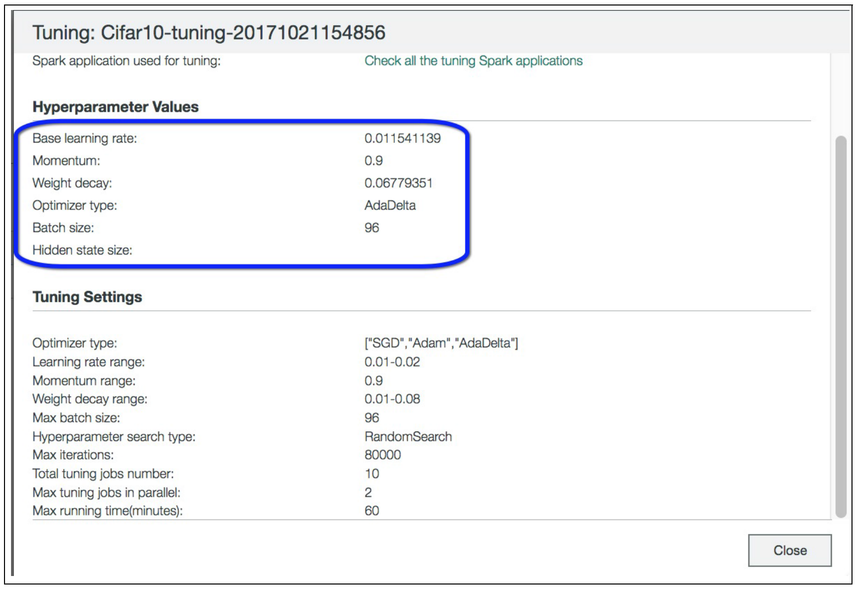
Tuning Workload Setting

Input the parameters to control tuning jobs and process

* Number of workers:	1
GPUs per worker:	1
* Max iterations:	
* Total tuning jobs number:	10
* Max tuning jobs in parallel:	2
* Max running time(minutes):	60

Figure 6-51 Setting parameters for one tuning task

2. After you click **Start Tuning**, the tuning jobs run in the background. When they complete, click **Tuning Task** to check the results, as shown in Figure 6-52.



Tuning: Cifar10-tuning-20171021154856

Spark application used for tuning: Check all the tuning Spark applications

Hyperparameter Values

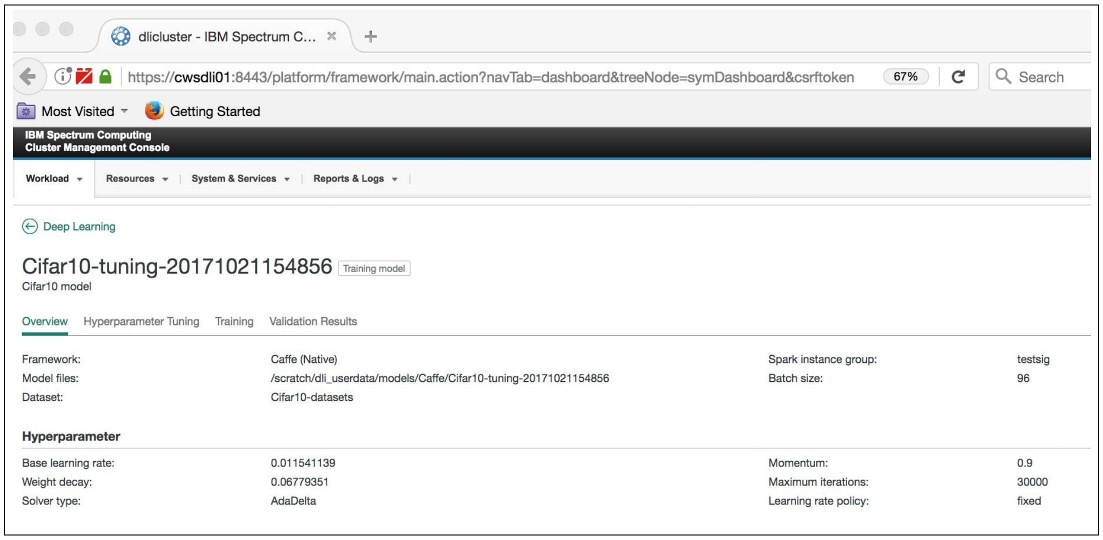
Base learning rate:	0.011541139
Momentum:	0.9
Weight decay:	0.06779351
Optimizer type:	AdaDelta
Batch size:	96
Hidden state size:	

Tuning Settings

Optimizer type:	["SGD","Adam","AdaDelta"]
Learning rate range:	0.01-0.02
Momentum range:	0.9
Weight decay range:	0.01-0.08
Max batch size:	96
Hyperparameter search type:	RandomSearch
Max iterations:	80000
Total tuning jobs number:	10
Max tuning jobs in parallel:	2
Max running time(minutes):	60

Figure 6-52 Viewing the tuning task's result

DLI also creates a new training model with the tuned hyperparameters based on the original model that you selected to tune, as shown in Figure 6-53. The new model is named `{original model name}-tuning-{timestamp}`. You can use this model to train again to get a new trained model, and then do validation to check the accuracy.


dlicluster - IBM Spectrum C...
https://cwsdli01:8443/platform/framework/main.action?navTab=dashboard&treeNode=symDashboard&csrftoken
67%
Search
Most Visited
Getting Started
IBM Spectrum Computing
Cluster Management Console
Workload
Resources
System & Services
Reports & Logs
Deep Learning

Cifar10-tuning-20171021154856 Training model
Cifar10 model

Overview | Hyperparameter Tuning | Training | Validation Results

Framework:	Caffe (Native)	Spark instance group:	testsig
Model files:	/scratch/dli_userdata/models/Caffe/Cifar10-tuning-20171021154856	Batch size:	96
Dataset:	Cifar10-datasets		

Hyperparameter

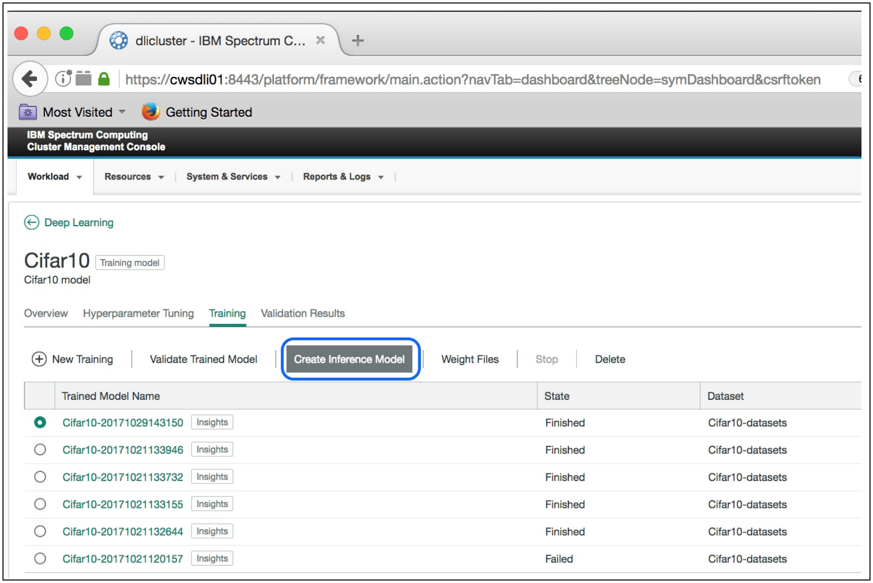
Base learning rate:	0.011541139	Momentum:	0.9
Weight decay:	0.06779351	Maximum iterations:	30000
Solver type:	AdaDelta	Learning rate policy:	fixed

Figure 6-53 Viewing the parameters of a new model that is created by the tuning task

Model inference management

When one trained model satisfies the accuracy requirement, you can transform it in to an inference model by completing the following steps:

1. Select a trained model and click **Create Inference Model**, as shown in Figure 6-54.

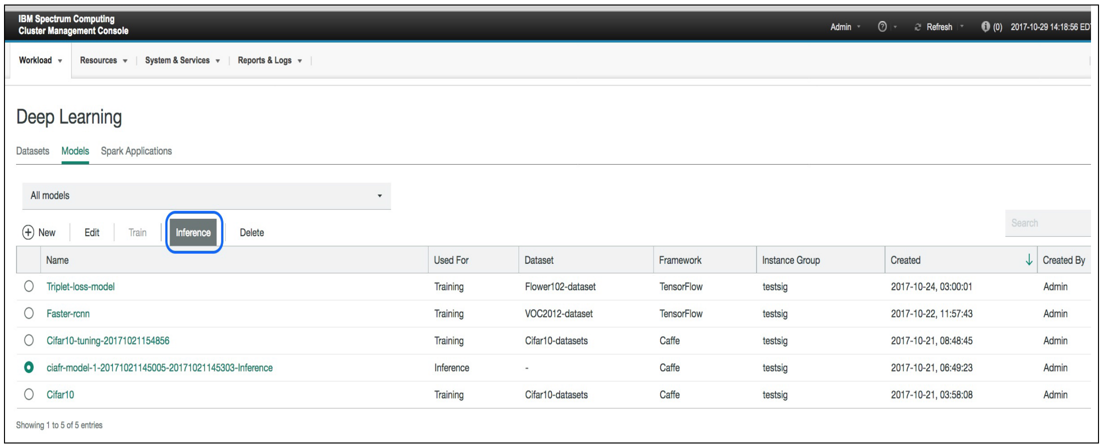


	Trained Model Name	State	Dataset
●	Cifar10-20171029143150 Insights	Finished	Cifar10-datasets
○	Cifar10-20171021133946 Insights	Finished	Cifar10-datasets
○	Cifar10-20171021133732 Insights	Finished	Cifar10-datasets
○	Cifar10-20171021133155 Insights	Finished	Cifar10-datasets
○	Cifar10-20171021132644 Insights	Finished	Cifar10-datasets
○	Cifar10-20171021120157 Insights	Failed	Cifar10-datasets

Figure 6-54 Generating an inference model from an existing trained model

Then, DLI creates a new inference model that is named `{original trained model name}-{timestamp}-Inference`. You can find it in the model list.

2. With this inference model, you can do a prediction operation. Select an inference model and click **Inference**, as shown in Figure 6-55 on page 197.

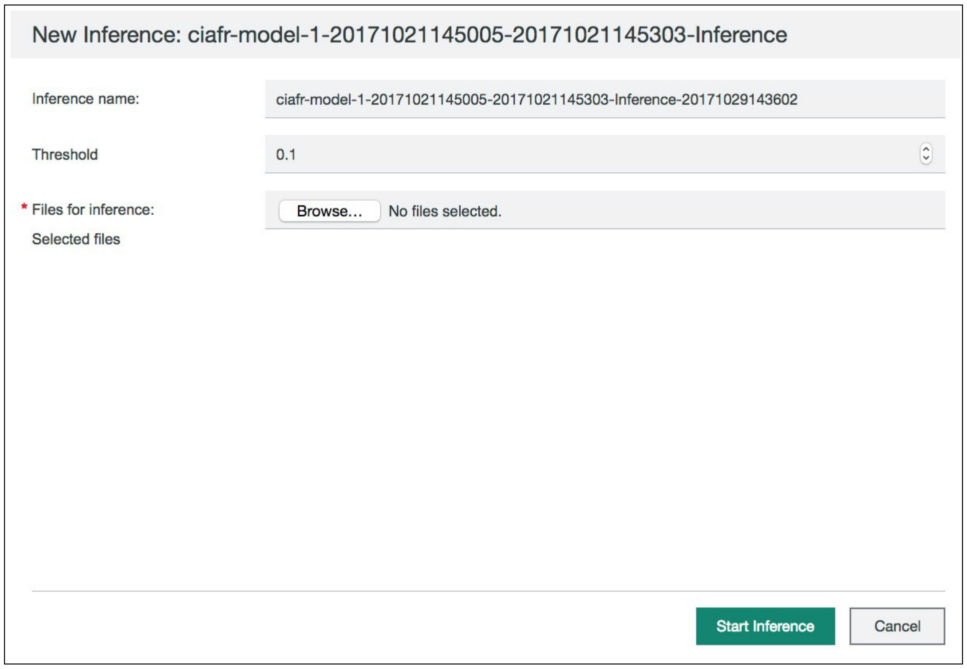


	Name	Used For	Dataset	Framework	Instance Group	Created	Created By
○	Triplet-loss-model	Training	Flower102-dataset	TensorFlow	testsig	2017-10-24, 03:00:01	Admin
○	Faster-rcnn	Training	VOC2012-dataset	TensorFlow	testsig	2017-10-22, 11:57:43	Admin
○	Cifar10-tuning-20171021154856	Training	Cifar10-datasets	Caffe	testsig	2017-10-21, 08:48:45	Admin
●	ciafr-model-1-20171021145005-20171021145303-Inference	Inference	-	Caffe	testsig	2017-10-21, 06:49:23	Admin
○	Cifar10	Training	Cifar10-datasets	Caffe	testsig	2017-10-21, 03:58:08	Admin

Showing 1 to 5 of 5 entries

Figure 6-55   Checking an inference list for an inference model

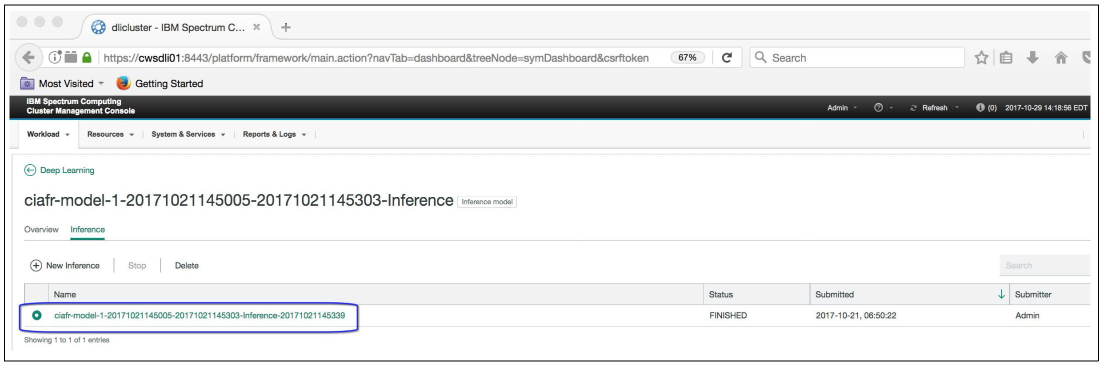
3. Complete the threshold item and select the files on which you want to do prediction. Then, click **Start Inference**, as shown in Figure 6-56.


New Inference: ciafr-model-1-20171021145005-20171021145303-Inference

Inference name:	ciafr-model-1-20171021145005-20171021145303-Inference-20171029143602
Threshold	0.1
* Files for inference:	Browse… No files selected.
Selected files	

Figure 6-56   Setting a parameter for an inference job

4. Click the inference model name and select **Inference**, which displays all the inference tasks, as shown in Figure 6-57.


dlicluster - IBM Spectrum C...
https://cwsdli01:8443/platform/framework/main.action?navTab=dashboard&treeNode=symDashboard&csrftoken
67%
Search
Most Visited
Getting Started
IBM Spectrum Computing Cluster Management Console
Admin
Refresh
(0) 2017-10-29 14:18:56 EDT
Workload
Resources
System & Services
Reports & Logs
Deep Learning

ciafr-model-1-20171021145005-20171021145303-Inference Inference model

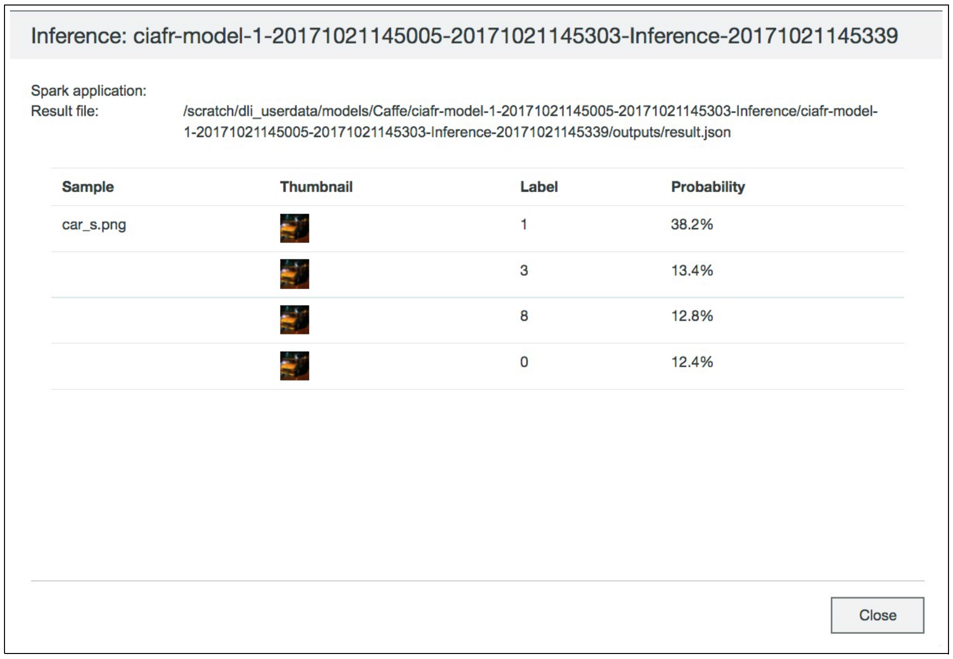
Overview
Inference
New Inference
Stop
Delete
Search

Name	Status	Submitted	Submitter
ciafr-model-1-20171021145005-20171021145303-Inference-20171021145339	FINISHED	2017-10-21, 06:50:22	Admin

Showing 1 to 1 of 1 entries

Figure 6-57 Viewing the inference result: 1

5. Select an inference task and click the name for the result to show. The result is saved in JSON file format in the target directory, as shown in Figure 6-58. The window also shows the target directory name.



Inference: ciafr-model-1-20171021145005-20171021145303-Inference-20171021145339

Spark application:
Result file: /scratch/dli_userdata/models/Caffe/ciafr-model-1-20171021145005-20171021145303-Inference/ciafr-model-1-20171021145005-20171021145303-Inference-20171021145339/outputs/result.json

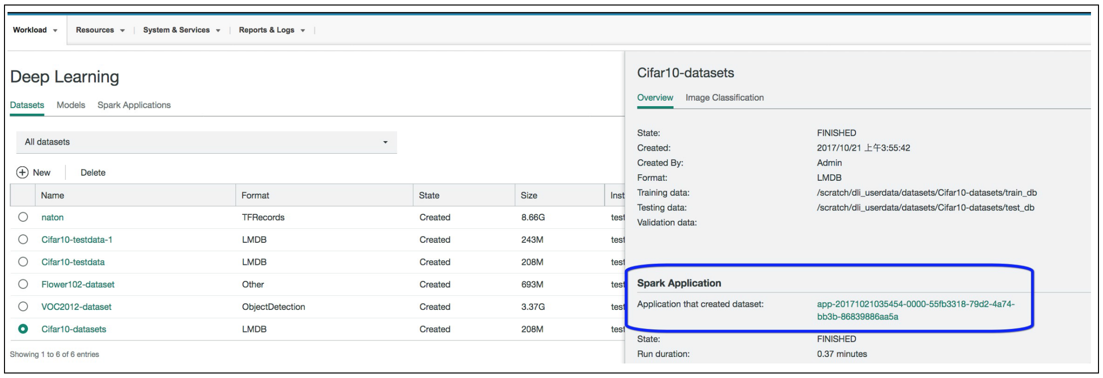
Sample	Thumbnail	Label	Probability
car_s.png		1	38.2%
		3	13.4%
		8	12.8%
		0	12.4%

Figure 6-58 Viewing the inference result: 2

6.6.3 Deep learning activity monitor and debug management

There are many kinds of workloads for DL activities, such as data set importing, model training, model validation, model tuning, and inference model prediction. DLI saves all the logs for debugging or other purposes.

For data set importing, there is a link in the overview window when the user selects a data set and click its name, as shown in Figure 6-59 on page 199.



Deep Learning

Datasets | Models | Spark Applications

All datasets

New | Delete

	Name	Format	State	Size	Inst
○	naton	TFRecords	Created	8.66G	test
○	Cifar10-testdata-1	LMDB	Created	243M	test
○	Cifar10-testdata	LMDB	Created	208M	test
○	Flower102-dataset	Other	Created	693M	test
○	VOC2012-dataset	ObjectDetection	Created	3.37G	test
●	Cifar10-datasets	LMDB	Created	208M	test

Showing 1 to 6 of 6 entries

Cifar10-datasets

Overview | Image Classification

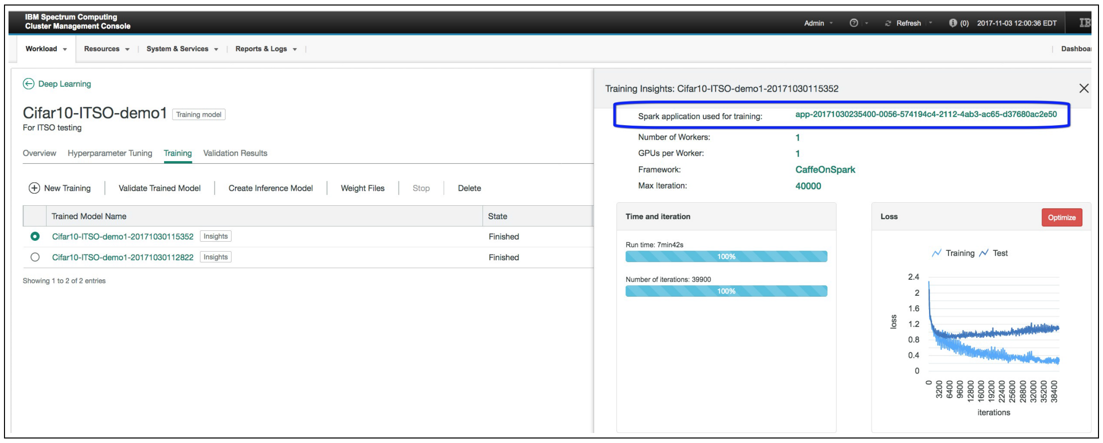
State:	FINISHED
Created:	2017/10/21 上午3:55:42
Created By:	Admin
Format:	LMDB
Training data:	/scratch/dli_userdata/datasets/Cifar10-datasets/train_db
Testing data:	/scratch/dli_userdata/datasets/Cifar10-datasets/test_db
Validation data:	

Spark Application

Application that created dataset:	app-20171021035454-0000-55fb3318-79d2-4a74-bb3b-86839886aa5a
State:	FINISHED
Run duration:	0.37 minutes

Figure 6-59   Finding logs for a data set importing job

For the model training job, the link is in the MAO dashboard, as shown in Figure 6-60.



Cifar10-ITSO-demo1 Training model

For ITSO testing

Overview | Hyperparameter Tuning | Training | Validation Results

New Training | Validate Trained Model | Create Inference Model | Weight Files | Stop | Delete

	Trained Model Name	State
●	Cifar10-ITSO-demo1-20171030115352 Insights	Finished
○	Cifar10-ITSO-demo1-20171030112822 Insights	Finished

Showing 1 to 2 of 2 entries

Training Insights: Cifar10-ITSO-demo1-20171030115352

Spark application used for training:	app-20171030235400-0056-574194c4-2112-4ab3-ac65-d37680ac2e50
Number of Workers:	1
GPUs per Worker:	1
Framework:	CaffeOnSpark
Max Iteration:	40000

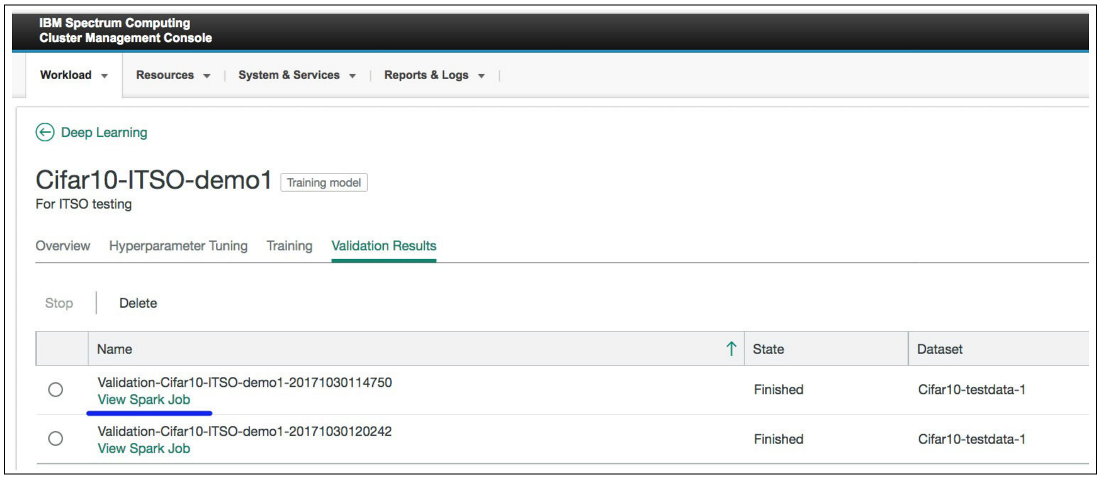
Time and iteration

Run time: 7min42s — 100%

Number of iterations: 39900 — 100%

Figure 6-60   Finding logs for a training model job

For the model validation job, the link is under the validation job's name, as shown in Figure 6-61.



Cifar10-ITSO-demo1 Training model

For ITSO testing

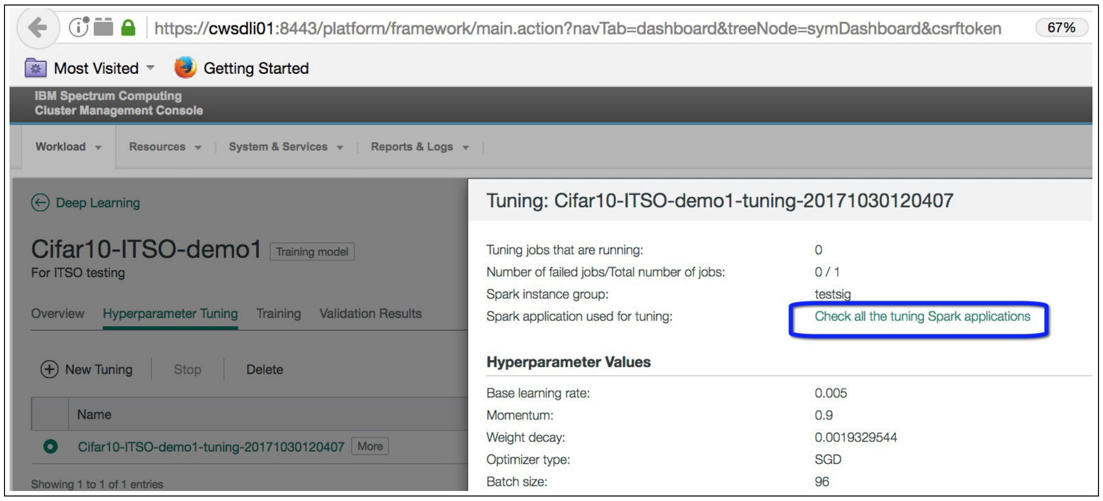
Overview | Hyperparameter Tuning | Training | Validation Results

Stop | Delete

	Name	State	Dataset
○	Validation-Cifar10-ITSO-demo1-20171030114750 View Spark Job	Finished	Cifar10-testdata-1
○	Validation-Cifar10-ITSO-demo1-20171030120242 View Spark Job	Finished	Cifar10-testdata-1

Figure 6-61   Finding logs for a validation job

For the hyperparameter tuning task, the link is in the result window, as shown in Figure 6-62.



Tuning: Cifar10-ITSO-demo1-tuning-20171030120407

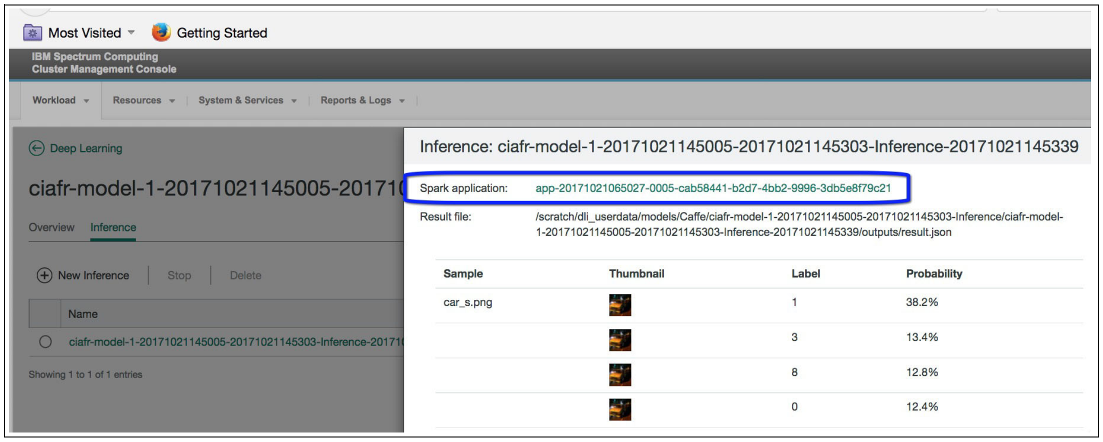
Tuning jobs that are running:	0
Number of failed jobs/Total number of jobs:	0 / 1
Spark instance group:	testsig
Spark application used for tuning:	Check all the tuning Spark applications

Hyperparameter Values

Base learning rate:	0.005
Momentum:	0.9
Weight decay:	0.0019329544
Optimizer type:	SGD
Batch size:	96

Figure 6-62 Finding logs for a hyperparameter tuning task

For the model inference job, the link is in the window that shows the prediction result, as shown in Figure 6-63.



Inference: ciafr-model-1-20171021145005-20171021145303-Inference-20171021145339

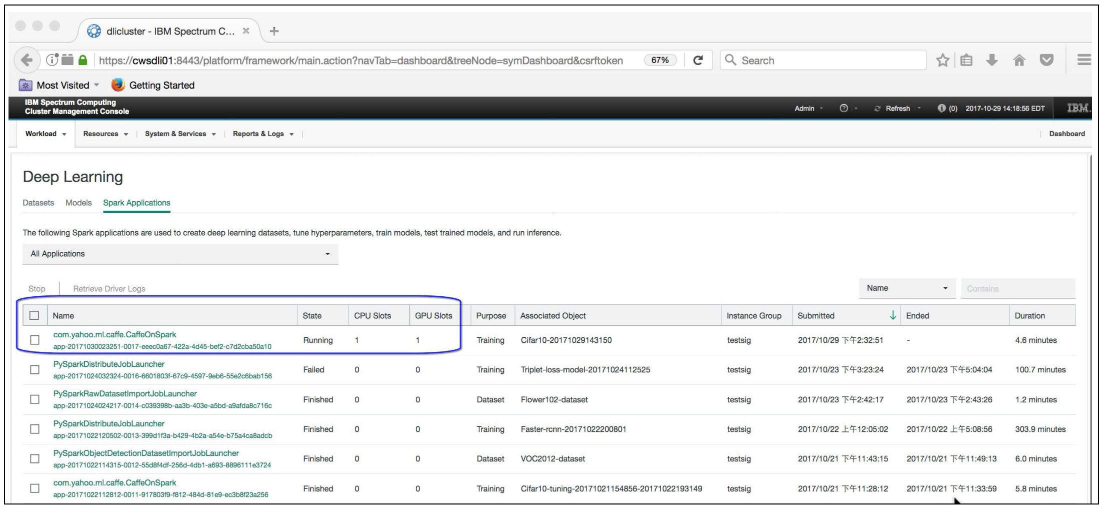
Spark application: app-20171021065027-0005-cab58441-b2d7-4bb2-9996-3db5e8f79c21

Result file: /scratch/dli_userdata/models/Caffe/ciafr-model-1-20171021145005-20171021145303-Inference/ciafr-model-1-20171021145005-20171021145303-Inference-20171021145339/outputs/result.json

Sample	Thumbnail	Label	Probability
car_s.png		1	38.2%
		3	13.4%
		8	12.8%
		0	12.4%

Figure 6-63 Finding logs for an inference job

You can also select the **Spark Application** menu in the Deep Learning dashboard to get debug information, as shown in Figure 6-64 on page 201.



Deep Learning

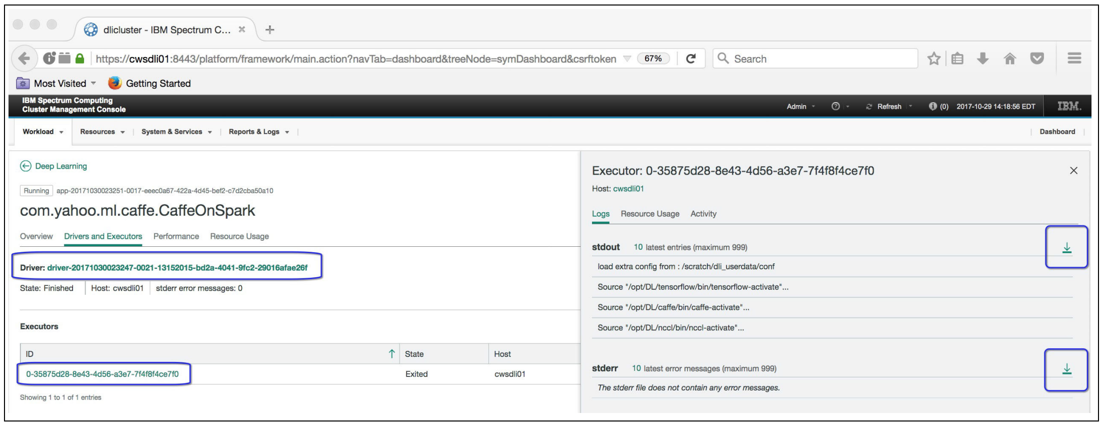
Datasets | Models | Spark Applications

The following Spark applications are used to create deep learning datasets, tune hyperparameters, train models, test trained models, and run inference.

Name	State	CPU Slots	GPU Slots	Purpose	Associated Object	Instance Group	Submitted	Ended	Duration
com.yahoo.ml.caffe.CaffeOnSpark app-20171030023251-0017-eeec0a67-422a-4d45-bef2-c7d2cba50a10	Running	1	1	Training	Cifar10-20171029143150	testsig	2017/10/29 下午2:32:51	-	4.6 minutes
PySparkDistributeJobLauncher app-20171024032324-0016-6601803f-67c9-4597-9eb6-55e2c6bab156	Failed	0	0	Training	Triplet-loss-model-20171024112525	testsig	2017/10/23 下午3:23:24	2017/10/23 下午5:04:04	100.7 minutes
PySparkRawDatasetImportJobLauncher app-20171024024217-0014-c039398b-aa3b-403e-a5bd-a9afda8c716c	Finished	0	0	Dataset	Flower102-dataset	testsig	2017/10/23 下午2:42:17	2017/10/23 下午2:43:26	1.2 minutes
PySparkDistributeJobLauncher app-20171022120502-0013-399d1f3a-b429-4b2a-a54e-b75a4ca8adcb	Finished	0	0	Training	Faster-rcnn-20171022200801	testsig	2017/10/22 上午12:05:02	2017/10/22 上午5:08:56	303.9 minutes
PySparkObjectDetectionDatasetImportJobLauncher app-20171022114315-0012-55d8f4df-256d-4db1-a693-8896111e3724	Finished	0	0	Dataset	VOC2012-dataset	testsig	2017/10/21 下午11:43:15	2017/10/21 下午11:49:13	6.0 minutes
com.yahoo.ml.caffe.CaffeOnSpark app-20171022112812-0011-917803f9-f812-484d-81e9-ec3b8f23a256	Finished	0	0	Training	Cifar10-tuning-20171021154856-20171022193149	testsig	2017/10/21 下午11:28:12	2017/10/21 下午11:33:59	5.8 minutes

Figure 6-64 Finding the log files from the Deep Learning dashboard

For each Spark application job, there are Overview, Drivers and Executors, Performance, and Resource Usage menus. The Drivers and Executors window provides log files to download, as shown in Figure 6-65.



Running app-20171030023251-0017-eeec0a67-422a-4d45-bef2-c7d2cba50a10

com.yahoo.ml.caffe.CaffeOnSpark

Overview | Drivers and Executors | Performance | Resource Usage

Driver: driver-20171030023247-0021-13152015-bd2a-4041-9fc2-29016afae26f

State: Finished | Host: cwsdli01 | stderr error messages: 0

Executors

ID	State	Host
0-35875d28-8e43-4d56-a3e7-7f4f8f4ce7f0	Exited	cwsdli01

Showing 1 to 1 of 1 entries

Executor: 0-35875d28-8e43-4d56-a3e7-7f4f8f4ce7f0

Host: cwsdli01

Logs | Resource Usage | Activity

stdout 10 latest entries (maximum 999)

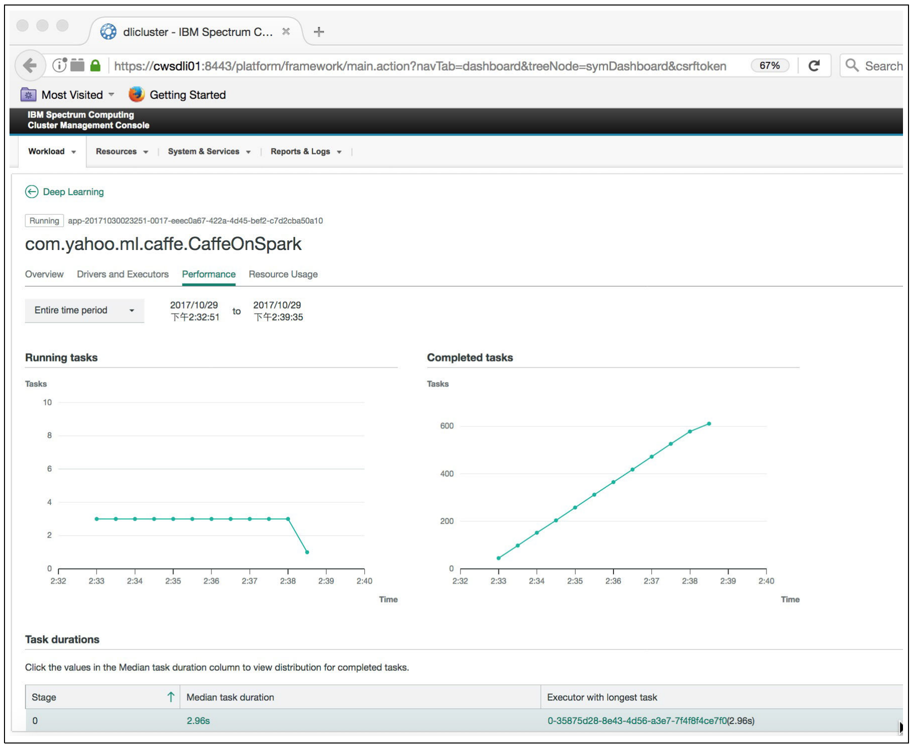
```
load extra config from : /scratch/dli_userdata/conf
Source "/opt/DL/tensorflow/bin/tensorflow-activate"...
Source "/opt/DL/caffe/bin/caffe-activate"...
Source "/opt/DL/nccl/bin/nccl-activate"...
```

stderr 10 latest error messages (maximum 999)

The stderr file does not contain any error messages.

Figure 6-65 One example about how to get different logs for a deep learning activity

The Performance window provides running and completed task information, as shown in Figure 6-66.



app-20171030023251-0017-eeec0a67-422a-4d45-bef2-c7d2cba50a10

com.yahoo.ml.caffe.CaffeOnSpark

2017/10/29 下午2:32:51 to 2017/10/29 下午2:39:35

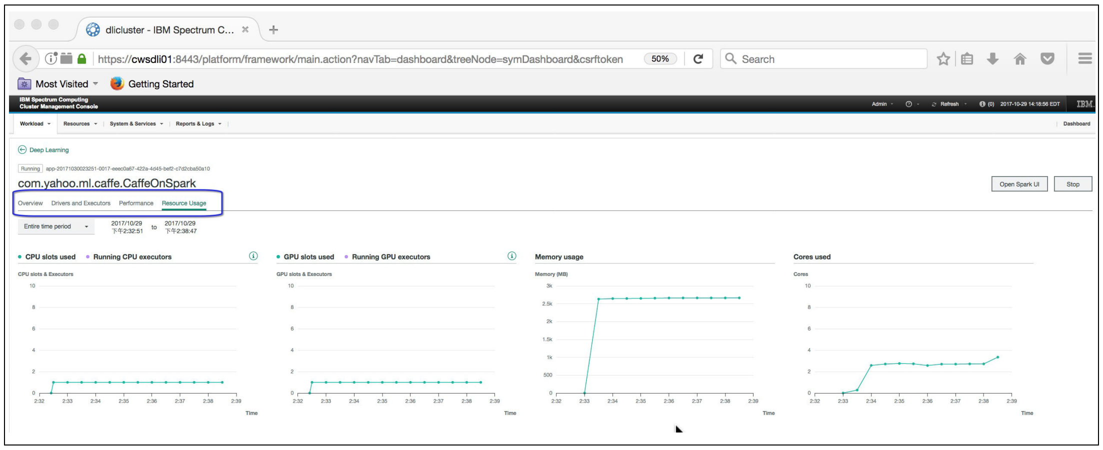
Task durations

Click the values in the Median task duration column to view distribution for completed tasks.

Stage	Median task duration	Executor with longest task
0	2.96s	0-35875d28-8e43-4d56-a3e7-7f4f8f4ce7f0(2.96s)

Figure 6-66 Performance data display window for a job

The Resource Usage window provides CPU, GPU, and physical memory usage while the job is running, as shown in Figure 6-67.



app-20171030023251-0017-eeec0a67-422a-4d45-bef2-c7d2cba50a10

com.yahoo.ml.caffe.CaffeOnSpark

2017/10/29 下午2:32:51 to 2017/10/29 下午2:38:47

Figure 6-67 Resource usage window for a job

6.7 Supported deep learning network and training engine in DLI

This section describes the supported DL network and training engine in DLI.

6.7.1 Deep learning network samples

When you want to use the DL network to do training or inference workload on the DLI platform, you can use network samples from IBM.

At the time of writing, DLI supports the networks that are shown in Table 6-5.

Table 6-5 Supported deep learning networks in DLI V1.1

Frameworks	Network	Supported training engine
Caffe	Cifar10, Resnet, and VGG19	Single-node training Distributed training with Caffe Distributed training with IBM Fabric Distributed training with IBM Fabric and auto-scaling
TensorFlow	Cifar10, Inception, and VGG19	Distributed training with TensorFlow Distributed training with IBM Fabric Distributed training with IBM Fabric and auto-scaling

DLI provides networks, including Caffenet, Darkflow, Facenet, LSTM, Triplet-loss, Faster-RCNN, and others with the following characteristics.

Note: The networks that are listed in the previous link are from open source websites and optimized by the IBM DLI developer team. Check for updates.

- Single-node training (Caffe)

  Using local Caffe mode for a training task.
- Distributed training with Caffe

  CaffeOnSpark supports distributed mode for training tasks.
- Distributed training with TensorFlow

  Using TensorFlow native distributed mode for training tasks.

Distributed training with IBM Fabric

This section describes distributed training with IBM Fabric.

IBM Fabric single-node and multi-GPU training

If you select the number of workers as equal to 1, the fabric training engine does not use the PS. The IBM Fabric single-node and multi-GPU training process is as follows and shown in Figure 6-68.

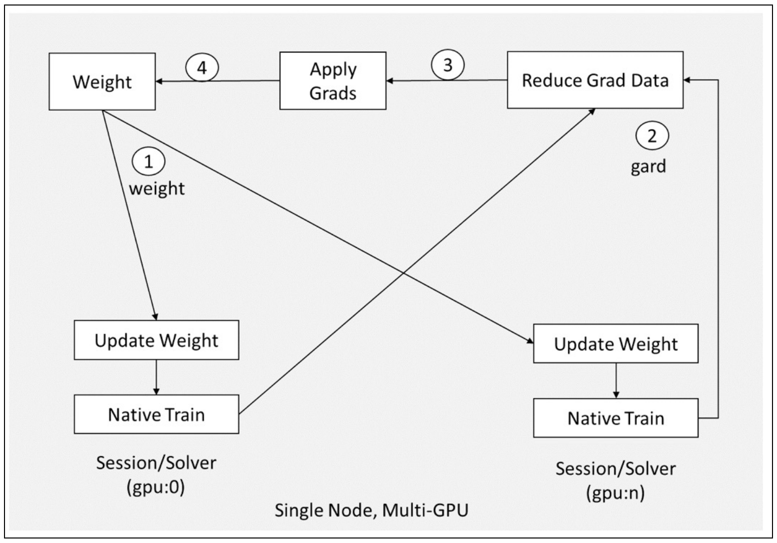


Figure 6-68 IBM Fabric single-node and multiple-GPU training process

1. Updates the weight value of a model and starts the model native train in multi-GPUs.
2. Gathers the gradient data from the GPU cards.
3. Reduces the gradient data.
4. Applies the reduced gradient data to weight values, and goes to step 1.

IBM Fabric multi-nodes and multi-GPU training

If you select the number of workers as greater than 1, the fabric training engine uses the PS. The IBM Fabric multi-nodes and multi-GPU training process is as follows and depicted in Figure 6-69.

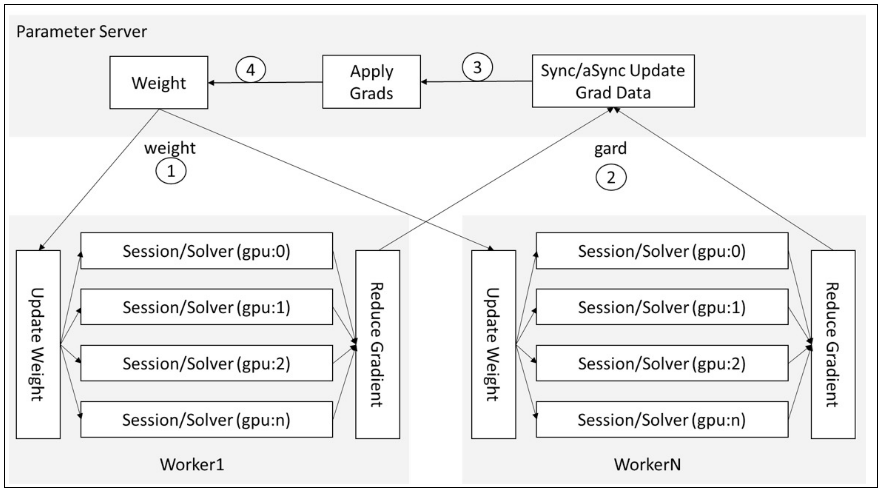


Figure 6-69 IBM Fabric multiple nodes and GPUs training process

1. Pulls the weight from the PS, updates the weight value to GPUs in the worker, and starts the model training.
2. Gathers and reduces the gradient data in the worker and pushes the reduced gradient data to PS.
3. Sync and async updates the gradient data in PS.
4. Applies the gradient data to the weight value in PS, and then goes to step 1.

If you select the number of workers as equal to 1, the fabric training engine does not use PS.

IBM Fabric core engine support by using NVIDIA Collective Communications Library

The IBM Fabric core engine supports the NCCL function for data exchange schema between GPU cards. To use it, you do not need to do any other configuration. Complete the following steps:

1. NCCL broadcasts the weight data schema, as shown in Figure 6-70.

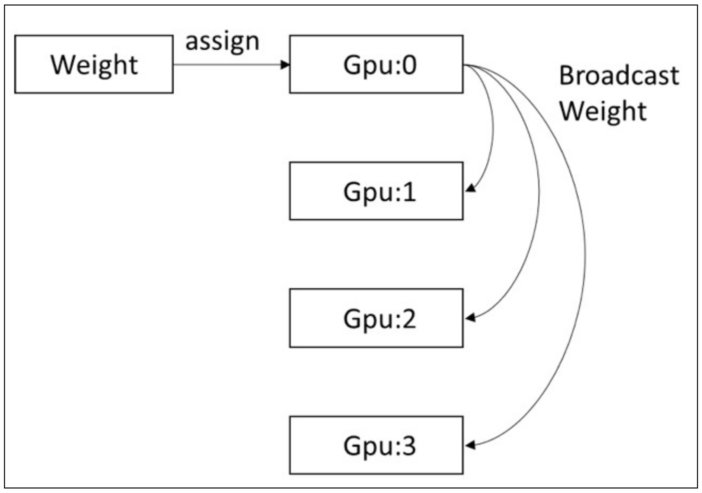


Figure 6-70 NVIDIA Collective Communications Library broadcasts the weight data

When the weight data is assigned to GPU:0, NCCL can broadcast the weight data to other GPU cards in the worker.

2. NCCL reduces the gradient data schema, as shown in Figure 6-71. GPU:0 reduces the gradient data with other GPU cards.

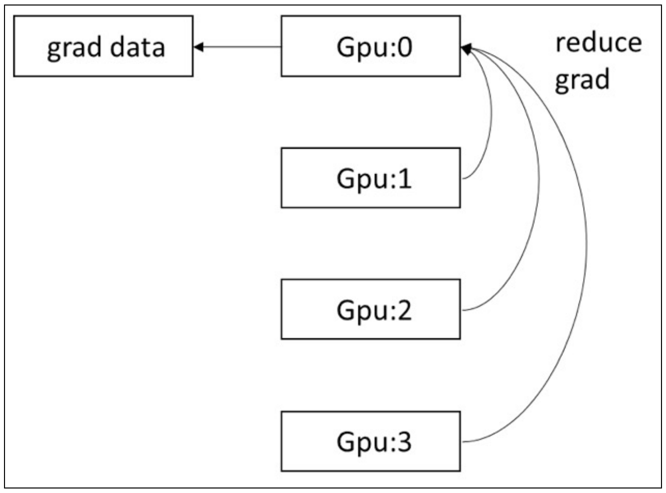
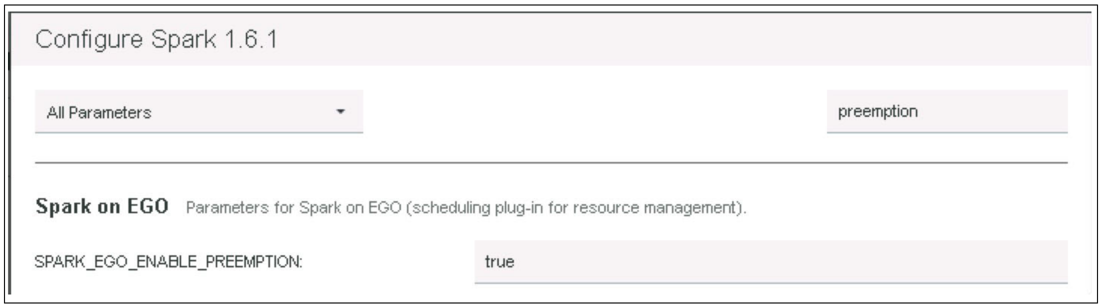


Figure 6-71 NVIDIA Collective Communications Library reduces the gradient data

Distributed training with IBM Fabric and auto-scaling

To use this feature, there are two parameters that you must set for the SIG:

- `SPARK_EGO_ENABLE_PREEMPTION`, as shown in Figure 6-72.

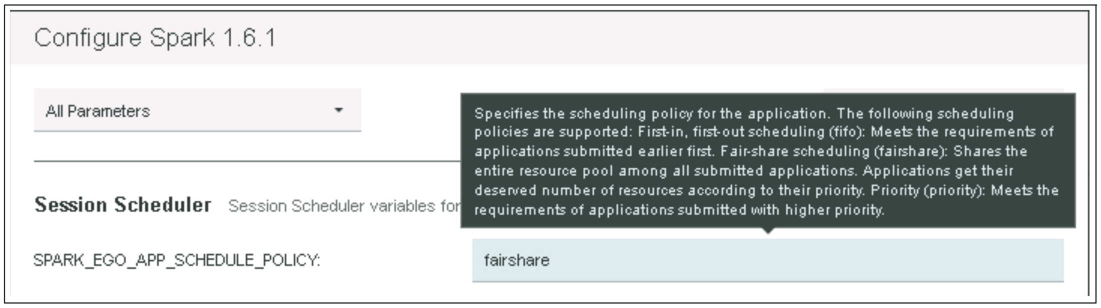


Spark on EGO Parameters for Spark on EGO (scheduling plug-in for resource management).

SPARK_EGO_ENABLE_PREEMPTION: true

Figure 6-72 Setting SPARK_EGO_ENABLE_PREEMPTION for the Elastic Fabric function

- `SPARK_EGO_APP_SCHEDULE_POLICY`, as shown in Figure 6-73.



Session Scheduler Session Scheduler variables for

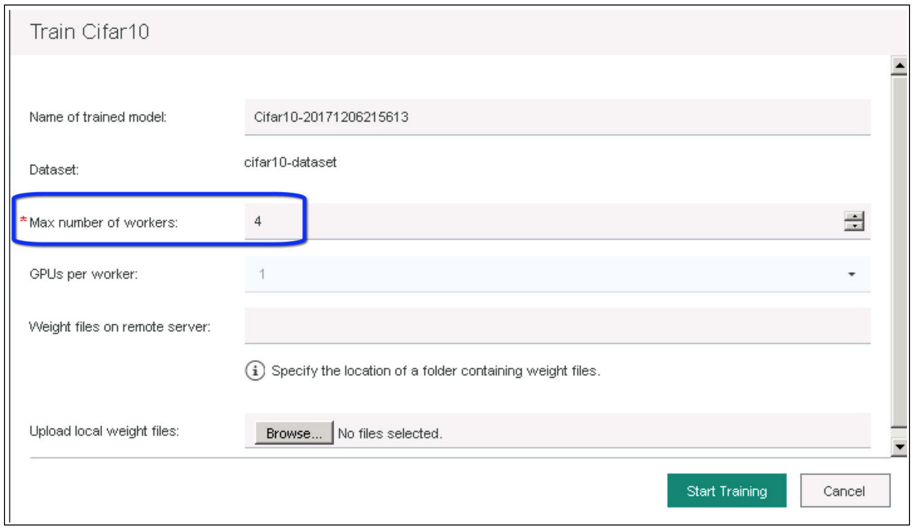
Specifies the scheduling policy for the application. The following scheduling policies are supported: First-in, first-out scheduling (fifo): Meets the requirements of applications submitted earlier first. Fair-share scheduling (fairshare): Shares the entire resource pool among all submitted applications. Applications get their deserved number of resources according to their priority. Priority (priority): Meets the requirements of applications submitted with higher priority.

SPARK_EGO_APP_SCHEDULE_POLICY: fairshare

Figure 6-73 Setting SPARK_EGO_APP_SCHEDULE_POLICY for the Elastic Fabric function

Note: If you stop the SIG service first, then you can modify the parameters.

After you choose **Distributed training with IBM Fabric and auto-scaling** as the training engine parameter for a training model, when you start a training task, there is one parameter that is named Max number of workers, as shown in Figure 6-74.


Train Cifar10

Name of trained model:	Cifar10-20171206215613
Dataset:	cifar10-dataset
* Max number of workers:	4
GPUs per worker:	1
Weight files on remote server:	
	Specify the location of a folder containing weight files.
Upload local weight files:	Browse... No files selected.

Figure 6-74 Setting the model's parameter for a training task

If there are less than four available GPU slots, this task still can start and run. During its run, if there are some GPU slots that are released by other tasks, this GPU resource can be added to this running task to maximize resource utilization.

6.7.2 Integrating with a customer's network in DLI

This section describes how to integrate the customer's network by using DLI.

Caffe single-node and distributed CaffeOnSpark training engine

You can import a Caffe model into the DLI platform for training and inference. A typical Caffe model that can run in the DLI platform has following structure:

- The `solver.prototxt` file: Caffe solver definition
- The `train_test.prototxt` file: Caffe train model definition
- The `inference.prototxt` file: Caffe inference model definition

You must rename your own files to these three fixed names before importing your model into DLI.

The Caffe models for different training engines are slightly different. For example, the Caffe model for the CaffeOnSpark training engine usually uses a `MemoryData` type of input data layer, and a single-node training engine does not. DLI does some conversion automatically for those known and compatible parameters. However, choose the compatible training engine or you might have to troubleshoot and change it after a failure.

There are sample Caffe models that are shared by DLI in IBM Bluemix®. For more information, see the model readme file.

TensorFlow single-node and distributed training engine

You can import a TensorFlow model into the DLI platform for training and inference. A typical TensorFlow model that can run in a DLI platform has following structure:

- The `main.py` file: The TensorFlow training model program main entrance
- The `inference.py` file: The TensorFlow inference model program main entrance
- The `fabricmodel.py` file: The callback program to convert a training model into a TensorFlow compute graph
- The `ps.conf` file: The training parameters, which are optional

The `main.py` and `inference.py` files are mandatory for a TensorFlow model for any training engine. The `fabricmodel.py` file is required if your model wants to run by using the IBM Fabric training engine. For more information about how to write the IBM Fabric TensorFlow model, see “Fitting the TensorFlow model to fit IBM Fabric” on page 210. The `main.py` file is the main training program.

If you want to train this model in single-node TensorFlow, the following command line options are mandatory:

- **`train_dir <TRAIN_DIR>`** is the directory where you write event logs and checkpoints.
- If you train this model in distributed TensorFlow, you must follow TensorFlow’s standard distributed training program guide and support the following extra command-line options:
  - **`job_name`**: One `ps` or `worker`
  - **`ps_hosts`**: Comma-separated list of `hostname:port` for the PS jobs
  - **`worker_hosts`**: Comma-separated list of `hostname:port` for the worker jobs
  - **`task_id`**: Task ID of the worker or replica that runs the training

DLI parameter injection API for TensorFlow model

TensorFlow models are written in Python, so these models are flexible. To work with DLI, DLI provides a TensorFlow parameter injection API for Python programs. You can use this API to get training parameters from a DLI platform, as shown in Example 6-8.

Example 6-8 API utilization example

```
…
import tf_parameter_mgr
…
FLAGS.max_steps=tf_parameter_mgr.getMaxSteps()
FLAGS.test_interval=tf_parameter_mgr.getTestInterval()
…
  filenames = tf_parameter_mgr.getTrainData()

  # Create a queue that produces the filenames to read.
  filename_queue = tf.train.string_input_producer(filenames)
…
  # Compute gradients.
  with tf.control_dependencies([loss_averages_op]):
    lr = tf_parameter_mgr.getLearningRate(global_step)
    opt = tf_parameter_mgr.getOptimizer(lr)
    grads = opt.compute_gradients(total_loss)
```

Here are the functions for the **`tf_parameter_mgr`**:

- **`getBaseLearningRate()`**
- **`getLearningRateDecay()`**
- **`getLearningRateDecaySteps()`**
- **`getTrainBatchSize()`**
- **`getTestBatchSize()`**
- **`getRNNHiddenStateSize()`**
- **`getTrainData(getfile=True)`**
- **`getTestData(getfile=True)`**
- **`getValData(getfile=True)`**
- **`getMaxSteps()`**
- **`getTestInterval()`**
- **`getOptimizer(learning_rate)`**
- **`getLearningRate(global_step=None)`**

You must change you code to call previously written API functions to use the training parameters. There are sample TensorFlow models that are shared by DLI by IBM Bluemix. For more information about detailed usages of these API functions, see the model readme file.

Fitting the Caffe model to IBM Fabric

This section introduces the process of fitting the Caffe model to IBM Fabric. When you have a Caffe model and want to use IBM Fabric to accelerate the training process, you must modify some Caffe model configurations:

1. You must name the Caffe solution file as `solver.prototxt`. In the `solver.prototxt` file, add the configuration that is shown in Example 6-9.

Example 6-9 Example of solver.prototxt

```
test_compute_loss:true
```

2. You must name the Caffe train and test net file as `train_test.prototxt`. In the `train_test.prototxt` file, the **`input_param`** data must be the `shape` type, as shown in Example 6-10.

Example 6-10 Example of train_test.prototxt for input_param

```
input_param { shape: { dim:200 dim: 3 dim: 32 dim: 32 } }
```

3. In the `train_test.prototxt` file, add the `accuracy` layer configuration, as shown in Example 6-11.

Example 6-11 Example of train_test.prototxt for the accuracy layer

```
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
}
```

Fitting the TensorFlow model to fit IBM Fabric

This section introduces the process of fitting the TensorFlow model to IBM Fabric. When you have single-node single-card TensorFlow model and want to use IBM Fabric for distributed training to accelerate the training process, you must call the IBM Fabric API and modify some of the model's code.

Before fitting the model to IBM Fabric, there are two limitations for the fabric TensorFlow model that you must understand:

1. IBM Fabric does not support **`tf.placeholder()`** as a data input schema. You must use the TensorFlow queue schema as data input. For more information about the TensorFlow queue schema, see the TensorFlow Programmers Guide.
2. In the TensorFlow model, you do not need to define the **`tf.device()`** operation. IBM Fabric auto-deploys the model to single-node/multi-nodes, and multi-GPU devices.

Note: Because of the first limitation, The IBM Fabric TensorFlow training engine is suitable for classification and object detection DL scenarios.

When you fit the TensorFlow model to DLI IBM Fabric, complete the following steps:

1. You must define the `fabricmodel.py` file.
2. In the `fabricmodel.py` file, you must import the fabric API as shown in Example 6-12. For development, you can obtain the API files `meta_writer.py` and `tf_meta_pb2.py` from the DLI IBM Fabric tools.

Example 6-12 Importing the IBM Fabric module

```
from meta_writer import *
```

3. For DLI IBM Fabric platform integration, you must define the API as shown in Example 6-13.

Example 6-13 Example of using an IBM Fabric API

```
DEFAULT_CKPT_DIR = './train'
DEFAULT_WEIGHT_FILE = ''
DEFAULT_MODEL_FILE = 'mymodel.model'
DEFAULT_META_FILE = 'mymodel.meta'
DEFAULT_GRAPH_FILE = 'mymodel.graph'

FLAGS = tf.app.flags.FLAGS
tf.app.flags.DEFINE_string('weights', DEFAULT_WEIGHT_FILE,
                           "Weight file to be loaded in order to validate,
inference or continue train")
tf.app.flags.DEFINE_string('train_dir', DEFAULT_CKPT_DIR,
                           "checkpoint directory to resume previous train and/or
snapshot current train, default to \"%s\"" % (DEFAULT_CKPT_DIR))
tf.app.flags.DEFINE_string('model_file', DEFAULT_MODEL_FILE,
                           "model file name to export, default to \"%s\"" %
(DEFAULT_MODEL_FILE))
tf.app.flags.DEFINE_string('meta_file', DEFAULT_META_FILE,
                           "meta file name to export, default to \"%s\"" %
(DEFAULT_META_FILE))
tf.app.flags.DEFINE_string('graph_file', DEFAULT_GRAPH_FILE,
                           "graph file name to export, default to \"%s\"" %
(DEFAULT_GRAPH_FILE))
```

```
#Notes:
#DEFAULT_CKPT_DIR is for tensorflow checkpoint file.
#DEFAULT_WEIGHT_FILE is for continue training with BlueMind API.
#DEFAULT_MODEL_FILE is for tensorflow graph Protobuf file which is used by fabric.
#DEFAULT_META_FILE is for fabric metadata file.
#DEFAULT_GRAPH_FILE is for tensorflow graph text file.
#The DEFAULT_MODEL_FILE, DEFAULT_META_FILE, DEFAULT_GRAPH_FILE will be generated
by the fabric API and used in DLI Fabric engine.
```

4. In the `fabricmodel.py` file, you must define the main method, as shown in Example 6-14. The main method is called by the DLI platform.

Example 6-14 Main method example in fabricmodel.py

```
def main(argv=None):
//deep learning model code
//……
//deep learning model code

//Finally call fabric write_meta API as follows
#the path to save model checkpoint
checkpoint_file = os.path.join(FLAGS.train_dir, "model.ckpt")

#the path to load prior weight file
restore_file = FLAGS.weights

#the snapshot interval to save checkpoint
snapshot_interval = 100

write_meta
(
tf,                                            # the tensorflow object.
None,       # the input placeholders, should be None.
train_accuracy,    # the train accuracy operation.
train_loss,     # the train loss operation.
test_accuracy,     # the test accuracy operation.
test_loss,     # the test loss operation.
optApplyOp,     # the apply gradient operation.
grads_and_vars,    # the grads_and_vars.
global_step,    # the global step.
FLAGS.model_file,     # the path to save the tensorflow graph protobuf file.
FLAGS.meta_file,    # the path to save fabric metadata file.
FLAGS.graph_file,    # the path to save tensorflow graph text file.
restore_file,    # the path to load prior weigh file for continue training.
checkpoint_file,    # the path to save model checkpoint file.
snapshot_interval    # the interval for saving model checkpoint file.
)

if __name__ == '__main__':
    tf.app.run()
```

Based on the IBM Fabric **write_meta** API requirements, the DL model code provides the follow DL operations:

- The train accuracy operation
- The train loss operation
- The test accuracy operation
- The test loss operation
- The global step operation
- The **grads_and_vars** operation, which can be obtained by **optimizer.compute_gradients(train_loss)**
- The apply gradient operation, which can be obtained by **optimizer.apply_gradients (grads, global_step=global_step)**

Note: The tensor of the train and test, accuracy, and loss operation must be TensorFlow scalar. If you cannot provide the train accuracy, test accuracy, and test loss operation in a specific model, you can set them to **tf.constant()**. For example:

```
train_accuracy=tf.constant(0.5,dtype=tf.float32)
```

6.8 Use case: Using a Caffe Cifar-10 network with DLI

Caffe Cifar-10 is one of the neural networks that DLI provides. This section uses this network and a Cifar-10 data set to demonstrate how to DLI performs a DL project. Figure 6-75 on page 213 shows the high-level steps.

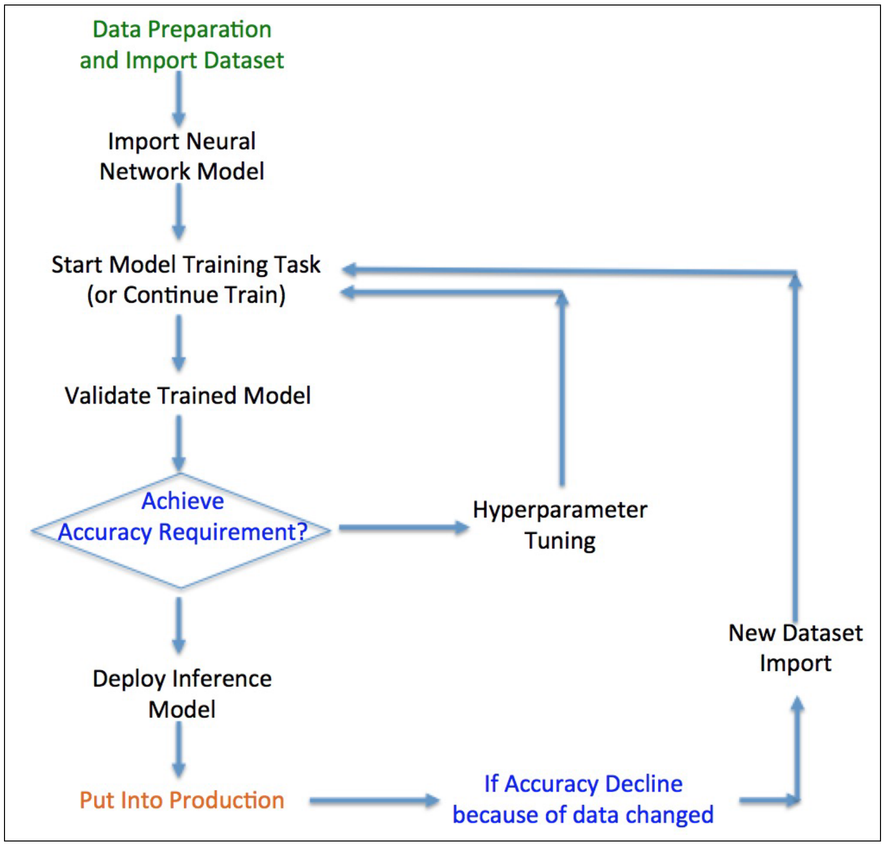

Data Preparation and Import Dataset
Import Neural Network Model
Start Model Training Task (or Continue Train)
Validate Trained Model
Achieve Accuracy Requirement?
Hyperparameter Tuning
New Dataset Import
Deploy Inference Model
Put Into Production
If Accuracy Decline because of data changed

Figure 6-75 Workflow of this use case

6.8.1 Data preparation

The Cifar-10 data set consists of 60,000 32x32 color images in 10 classes, with 6000 images per class. There are 50,000 training images and 10,000 test images.

The original data set is divided into five training batches and one test batch, each with 10,000 images. The test batch contains exactly 1000 randomly selected images from each class. Example 6-15 shows a script to prepare the data set for importing.

Example 6-15 Cifar-10 data set preparation

```
#!/bin/bash
wget --no-check-certificate http://www.cs.toronto.edu/~kriz/cifar-10-binary.tar.gz
tar -xvf cifar-10-binary.tar.gz

lv_dir=`pwd`
export target_dir=${lv_dir}/cifar10_testdata
export source_dir=${lv_dir}/cifar-10-batches-bin
mkdir -p ${target_dir}
rm -rf ${target_dir}/cifar10_train_lmdb ${target_dir}/cifar10_test_lmdb


echo "Generate LMDB files..."
source /opt/DL/caffe/bin/caffe-activate
/opt/DL/caffe/bin/convert_cifar_data.bin ${source_dir} ${target_dir} lmdb
```

```
echo "Compute image mean..."
/opt/DL/caffe/bin/compute_image_mean -backend=lmdb
${target_dir}/cifar10_train_lmdb ${target_dir}/mean.binaryproto

chown -R egoadmin:egoadmin ${source_dir}
chown -R egoadmin:egoadmin ${target_dir}

echo "Done."
```

Example 6-16 displays the output of the script that is shown in Example 6-15 on page 213.

Example 6-16 Output of this data preparation script

```
Generate LMDB files...
I1030 09:25:35.053671  6147 db_lmdb.cpp:35] Opened lmdb
/demodata/userdata/cifar10_binary/cifar10_testdata/cifar10_train_lmdb
I1030 09:25:35.054055  6147 convert_cifar_data.cpp:52] Writing Training data
I1030 09:25:35.054069  6147 convert_cifar_data.cpp:55] Training Batch 1
I1030 09:25:35.105211  6147 convert_cifar_data.cpp:55] Training Batch 2
I1030 09:25:35.214681  6147 convert_cifar_data.cpp:55] Training Batch 3
I1030 09:25:35.262233  6147 convert_cifar_data.cpp:55] Training Batch 4
I1030 09:25:35.360141  6147 convert_cifar_data.cpp:55] Training Batch 5
I1030 09:25:38.657042  6147 convert_cifar_data.cpp:73] Writing Testing data
I1030 09:25:38.830090  6147 db_lmdb.cpp:35] Opened lmdb
/demodata/userdata/cifar10_binary/cifar10_testdata/cifar10_test_lmdb
Compute image mean...
I1030 09:25:41.052309  6941 db_lmdb.cpp:35] Opened lmdb
/demodata/userdata/cifar10_binary/cifar10_testdata/cifar10_train_lmdb
I1030 09:25:41.060359  6941 compute_image_mean.cpp:70] Starting iteration
I1030 09:25:41.120590  6941 compute_image_mean.cpp:95] Processed 10000 files.
I1030 09:25:41.156601  6941 compute_image_mean.cpp:95] Processed 20000 files.
I1030 09:25:41.205094  6941 compute_image_mean.cpp:95] Processed 30000 files.
I1030 09:25:41.267137  6941 compute_image_mean.cpp:95] Processed 40000 files.
I1030 09:25:41.323477  6941 compute_image_mean.cpp:95] Processed 50000 files.
I1030 09:25:41.323568  6941 compute_image_mean.cpp:108] Write to
/demodata/userdata/cifar10_binary/cifar10_testdata/mean.binaryproto
I1030 09:25:41.348294  6941 compute_image_mean.cpp:114] Number of channels: 3
I1030 09:25:41.348325  6941 compute_image_mean.cpp:119] mean_value channel [0]:
125.307
I1030 09:25:41.348400  6941 compute_image_mean.cpp:119] mean_value channel [1]:
122.95
I1030 09:25:41.348415  6941 compute_image_mean.cpp:119] mean_value channel [2]:
113.865
Done.

# ls -l cifar10_testdata/*
-rw-r--r-- 1 egoadmin egoadmin 12299 Oct 30 09:25
cifar10_testdata/mean.binaryproto

cifar10_testdata/cifar10_test_lmdb:
total 64520
-rw-r--r-- 1 egoadmin egoadmin 36503552 Oct 30 09:25 data.mdb
-rw-r--r-- 1 egoadmin egoadmin     8192 Oct 30 09:25 lock.mdb

cifar10_testdata/cifar10_train_lmdb:
total 261128
```

```
-rw-r--r-- 1 egoadmin egoadmin 182255616 Oct 30 09:25 data.mdb
-rw-r--r-- 1 egoadmin egoadmin      8192 Oct 30 09:25 lock.mdb

#pwd
/demodata/userdata/cifar10_binary
```

6.8.2 Data set import

To import the data set, complete the following steps:

1. Log in to the DLI GUI, as described in “How to log in to the IBM Spectrum Computing Management Console” on page 177.
2. Click **data sets**, select your SIG, and click **New** →**Select LMDBs**. The New data sets window opens, where you input parameters. In this case, the parameters are shown in Table 6-6.

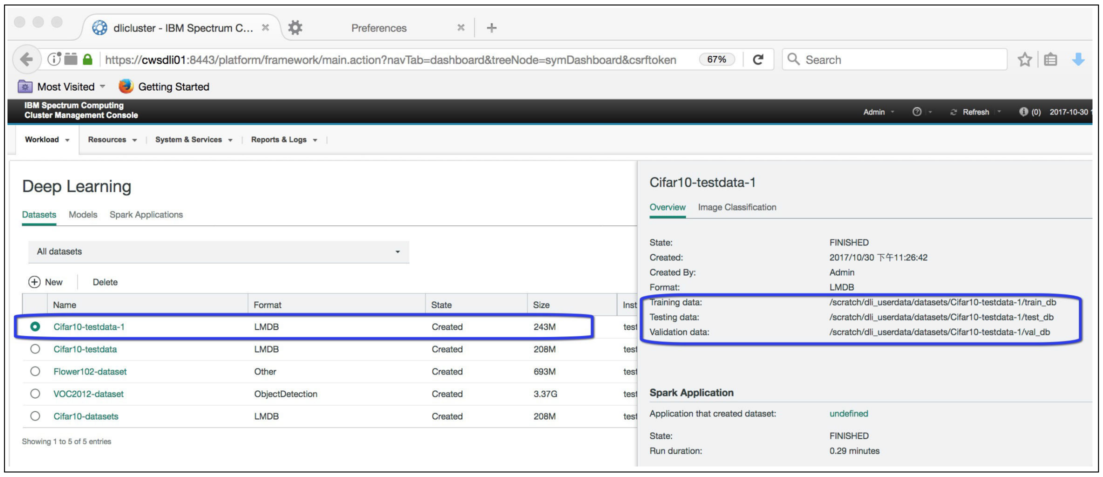
Table 6-6 Parameters setting for data set import

Parameter	Value
Data set name	Cifar10-testdata-1
Create in SIG	testsig
Training folder	/demodata/userdata/cifar10_binary/cifar10_testdata/cifar10_train_lmdb
Validation folder (optional)	/demodata/userdata/cifar10_binary/cifar10_testdata/cifar10_test_lmdb
Testing folder	/demodata/userdata/cifar10_binary/cifar10_testdata/cifar10_test_lmdb
Mean file	/demodata/userdata/cifar10_binary/cifar10_testdata/mean.binaryproto
Label file (optional)	/demodata/userdata/cifar10_binary/cifar-10-batches-bin/batches.meta.txt

Note: This data set does not include a validation data set. In general, it does not need to configure a validation folder. However, we want to show the validation function, so we use a test data set as a validation data set.

This data set is binary format. If there is no label file that is configured, when you do the inference, the result shows only the class number. With a label file, it converts to a specific class name. The Cifar-10 data set provides a label file, and the file name is `batches.meta.txt`.

3. Click **Create**. DLI transforms the data in its working directory. When this task is complete, select this task and click the name. You can see the result as shown in Figure 6-76.


dlicluster - IBM Spectrum C...
Preferences
https://cwsdli01:8443/platform/framework/main.action?navTab=dashboard&treeNode=symDashboard&csrftoken
67%
Search
Most Visited
Getting Started
IBM Spectrum Computing
Cluster Management Console
Admin
Refresh
(0)
2017-10-30
Workload
Resources
System & Services
Reports & Logs

Deep Learning

Datasets Models Spark Applications

All datasets

New Delete

	Name	Format	State	Size	Inst
●	Cifar10-testdata-1	LMDB	Created	243M	test
○	Cifar10-testdata	LMDB	Created	208M	test
○	Flower102-dataset	Other	Created	693M	test
○	VOC2012-dataset	ObjectDetection	Created	3.37G	test
○	Cifar10-datasets	LMDB	Created	208M	test

Showing 1 to 5 of 5 entries

Cifar10-testdata-1

Overview Image Classification

State:	FINISHED
Created:	2017/10/30 下午11:26:42
Created By:	Admin
Format:	LMDB
Training data:	/scratch/dli_userdata/datasets/Cifar10-testdata-1/train_db
Testing data:	/scratch/dli_userdata/datasets/Cifar10-testdata-1/test_db
Validation data:	/scratch/dli_userdata/datasets/Cifar10-testdata-1/val_db

Spark Application

Application that created dataset:	undefined
State:	FINISHED
Run duration:	0.29 minutes

Figure 6-76 Showing the result of a data set import

6.8.3 Model creation

After a data set is imported, you import the model template and create a model for training by completing the following steps:

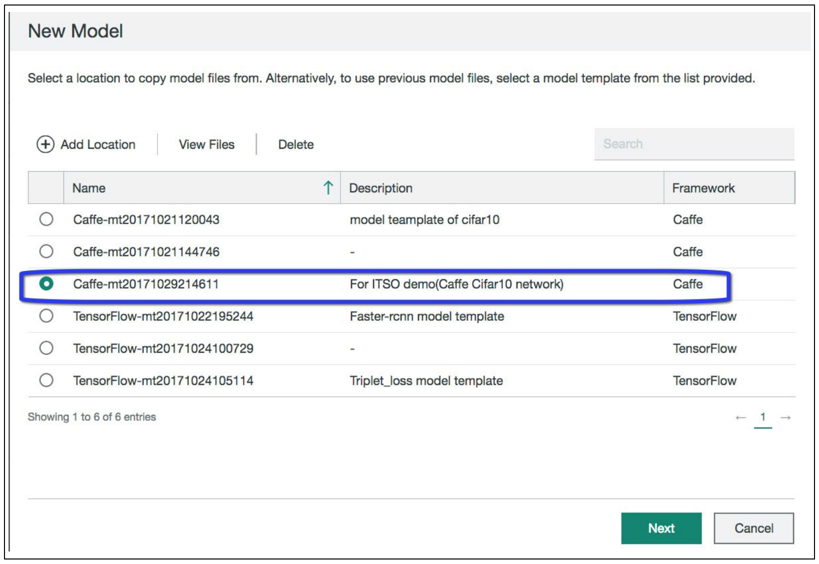
1. Click **Models** →**New** →**Add Location**. DLI prompts you to add a location for the model file, which is also known as adding a model template. You use the parameters that are shown in Table 6-7.

Table 6-7 Parameters for Cifar-10 model import

Parameter	**Value**
Framework	Caffe
Path	/opt/ibm/spectrumcomputing/dli/dlpd/examples/caffe/cifar10
Description (optional)	For ITSO demonstration (Caffe Cifar-10 network)

Note: Caffe Cifar-10 model files are installed automatically during the DLI installation.

A model template is created with the name Caffe-mt20171029214611, as shown in Figure 6-77 on page 217.


New Model

Select a location to copy model files from. Alternatively, to use previous model files, select a model template from the list provided.

	Name	Description	Framework
○	Caffe-mt20171021120043	model teamplate of cifar10	Caffe
○	Caffe-mt20171021144746	-	Caffe
●	Caffe-mt20171029214611	For ITSO demo(Caffe Cifar10 network)	Caffe
○	TensorFlow-mt20171022195244	Faster-rcnn model template	TensorFlow
○	TensorFlow-mt20171024100729	-	TensorFlow
○	TensorFlow-mt20171024105114	Triplet_loss model template	TensorFlow

Showing 1 to 6 of 6 entries

Figure 6-77 Creating a Cifar-10 model

2. Select this model template, and click **Next**. The Add Caffe model window opens. Table 6-8 shows the parameters to be entered in this window.

Table 6-8 List of parameters and values for the template model

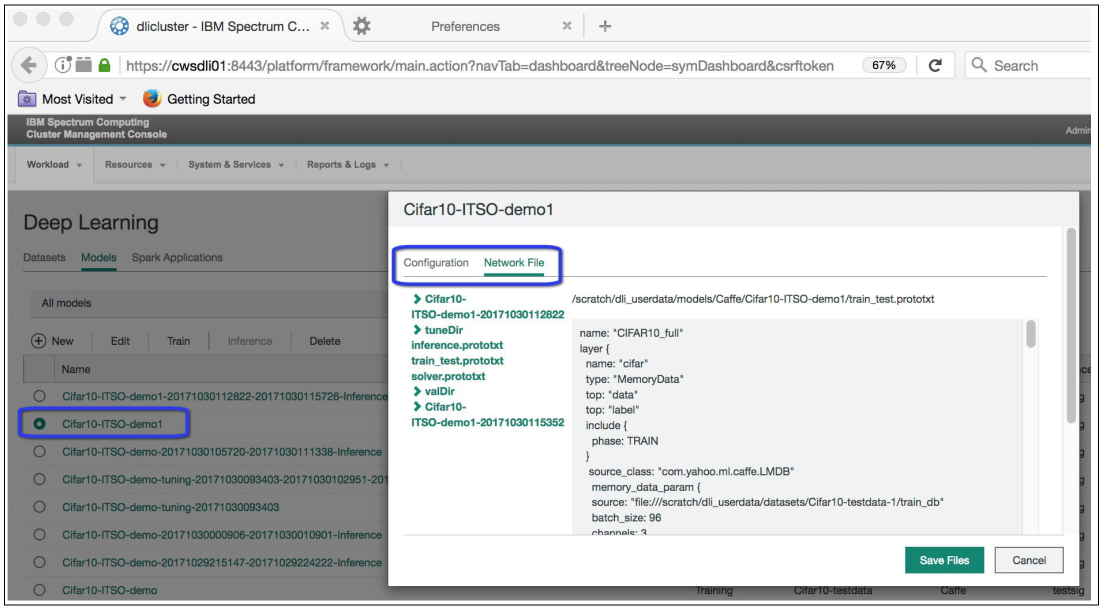
Parameter	Value
Create an inference model	unmark
Model name	Cifar10-ITSO-Demo-1
Model description	For ITSO demonstration (Caffe Cifar-10 network)
Training engine	Native
Training data set	Cifar10-testdata
Base learning rate	0.01 (Default)
Momentum	0.9 (Default)
Weight decay	0.0004 (Default)
Max Iterations	40000 (Default)
Optimizer type	Stochastic gradient descent (SGD) (Default)
Learning rate policy	Fix (Default)
Batch size	96

Note: For the first parameter, because you are creating a model for training, unmask it. If you want to create a model for inference, select it.

For the training engine type choices, there are Single, Native, Fabric, and Elastic.

For the hyperparameters, you can change them based on your experiences.

3. Click **Add** to complete model creation. A new item is added to the Models list. Select it and click **Edit**. Then, you can modify the configuration and prototxt files, as shown in Figure 6-78.


dlicluster - IBM Spectrum C...
Preferences
https://cwsdli01:8443/platform/framework/main.action?navTab=dashboard&treeNode=symDashboard&csrftoken
Most Visited
Getting Started
IBM Spectrum Computing Cluster Management Console
Workload
Resources
System & Services
Reports & Logs
Deep Learning
Datasets
Models
Spark Applications
All models
New
Edit
Train
Inference
Delete
Name
Cifar10-ITSO-demo1-20171030112822-20171030115726-Inference
Cifar10-ITSO-demo1
Cifar10-ITSO-demo-20171030105720-20171030111338-Inference
Cifar10-ITSO-demo-tuning-20171030093403-20171030102951-201
Cifar10-ITSO-demo-tuning-20171030093403
Cifar10-ITSO-demo-20171030000906-20171030010901-Inference
Cifar10-ITSO-demo-20171029215147-20171029224222-Inference
Cifar10-ITSO-demo
Cifar10-ITSO-demo1
Configuration
Network File
Cifar10-ITSO-demo1-20171030112822
tuneDir
inference.prototxt
train_test.prototxt
solver.prototxt
valDir
Cifar10-ITSO-demo1-20171030115352
Save Files
Cancel

/scratch/dli_userdata/models/Caffe/Cifar10-ITSO-demo1/train_test.prototxt

```
name: "CIFAR10_full"
layer {
  name: "cifar"
  type: "MemoryData"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
   source_class: "com.yahoo.ml.caffe.LMDB"
    memory_data_param {
    source: "file:///scratch/dli_userdata/datasets/Cifar10-testdata-1/train_db"
    batch_size: 96
    channels: 3
```

Figure 6-78 Editing the Cifar-10 network files

6.8.4 Model training

After the model is created, you can start a training task by completing the following steps:

1. Select the model and click **Train**. The Train Cifar10-ITSO-demo window opens. Table 6-9 shows the parameters that are used for the training task.

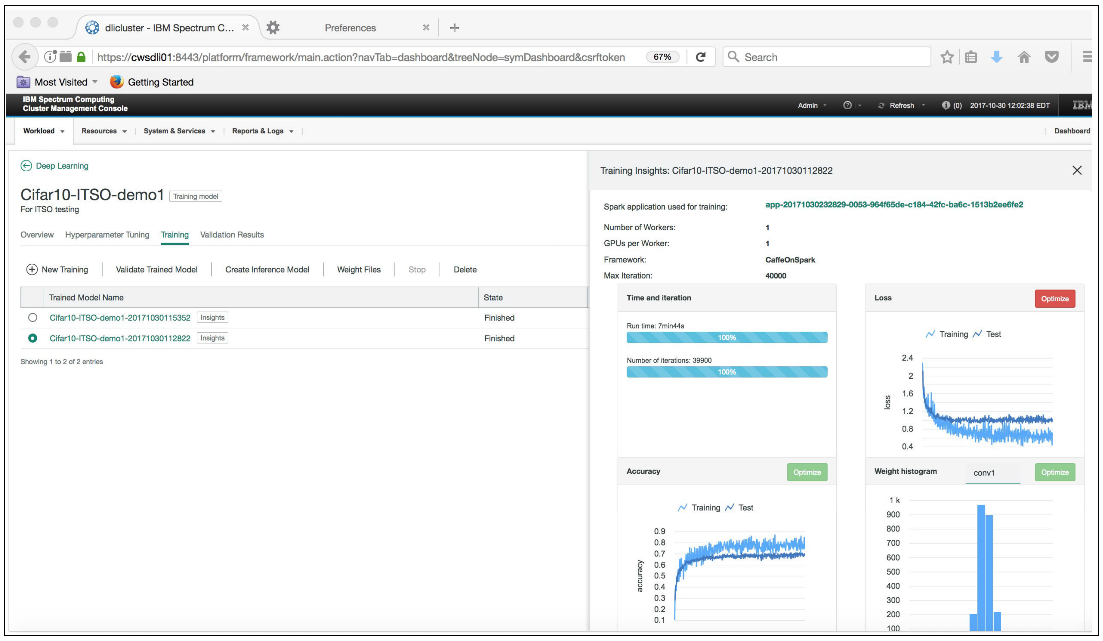
Table 6-9 Parameters for a training job

Parameter	Value
Name of the trained model	Cifar10-ITSO-demo1-20171030112822 (Default, can change)
Data set	cifar10-testdata-1
Number of workers	1
GPUs per worker	1
Framework	Caffe on Spark
Weight files on the remote server	See note below.

Note: In this environment, we have only one node, which has one GPU, so the Number of workers and GPUs per worker should be 1.

For the last parameter, there is one weight file that we want to use to continue training for this scenario. You can fill in the path for the weight file that you want to use.

2. Click **Start training**, and the training task starts. To monitor the process of this training job, open the MAO window. Click the current training model Cifar10-ITSO-demo, click **Training**, select one training task, and click **Insights**. The MAO window is shown in Figure 6-79.



Cifar10-ITSO-demo1 Training model
For ITSO testing

	Trained Model Name	State
○	Cifar10-ITSO-demo1-20171030115352 Insights	Finished
●	Cifar10-ITSO-demo1-20171030112822 Insights	Finished

Showing 1 to 2 of 2 entries

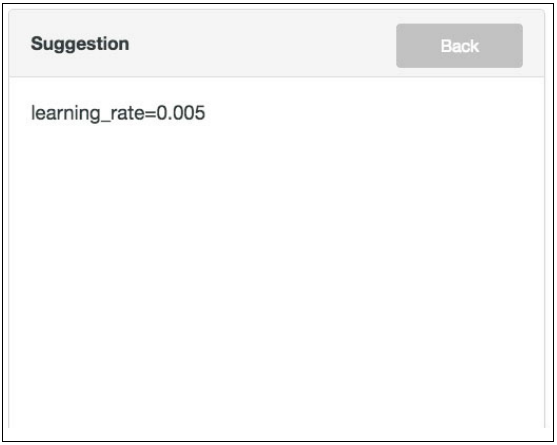
Training Insights: Cifar10-ITSO-demo1-20171030112822

Spark application used for training:	app-20171030232829-0053-964f65de-c184-42fc-ba6c-1513b2ee6fe2
Number of Workers:	1
GPUs per Worker:	1
Framework:	CaffeOnSpark
Max Iteration:	40000

Time and iteration
Run time: 7min44s
100%
Number of iterations: 39900
100%

Figure 6-79 Monitoring and Optimization window for this training model

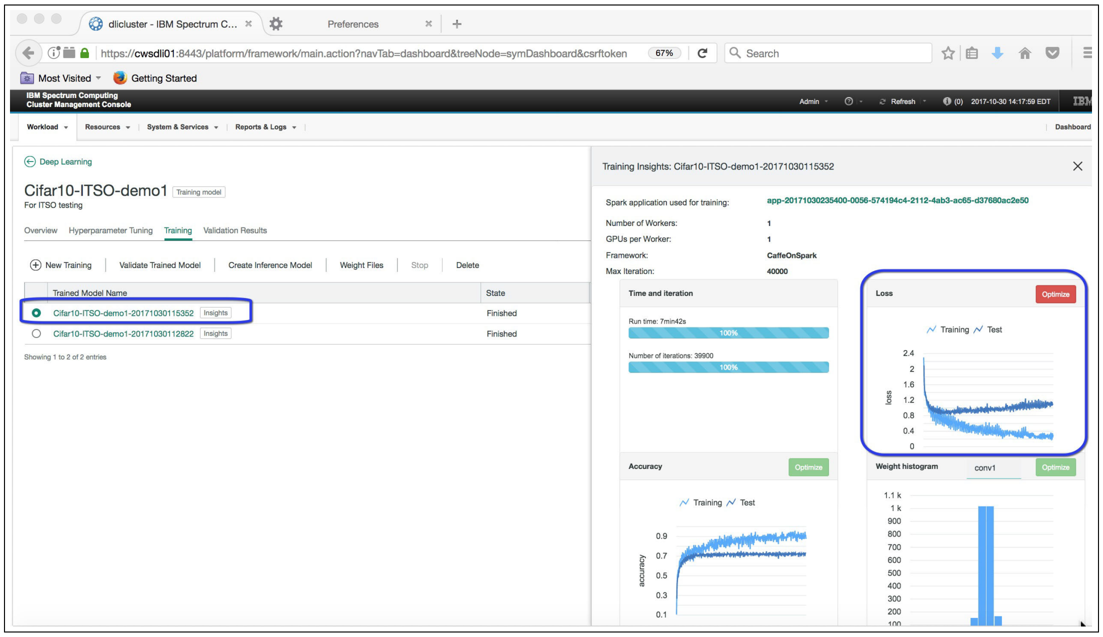
3. Figure 6-79 shows a suggestion for this training job. Click Optimize at the upper right of the loss graph. Figure 6-80 shows the optimization suggestion for this training job.



learning_rate=0.005

Figure 6-80 Optimization suggestion for this training job

4. Use this new learning rate value (Figure 6-80 on page 219) to start another training job, and get the result, as shown in Figure 6-81.



Cifar10-ITSO-demo1
For ITSO testing

Trained Model Name	State
Cifar10-ITSO-demo1-20171030115352	Finished
Cifar10-ITSO-demo1-20171030112822	Finished

Showing 1 to 2 of 2 entries

Training Insights: Cifar10-ITSO-demo1-20171030115352

Spark application used for training:	app-20171030235400-0056-574194c4-2112-4ab3-ac65-d37680ac2e50
Number of Workers:	1
GPUs per Worker:	1
Framework:	CaffeOnSpark
Max Iteration:	40000

Time and iteration

Run time: 7min42s 100%

Number of iterations: 39900 100%

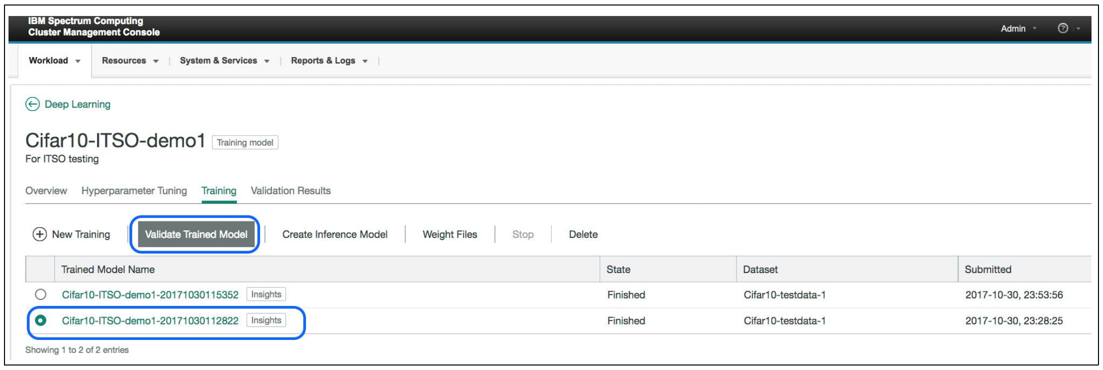
Figure 6-81 Monitoring and Optimization window for this training model

You can compare the accuracy of these two models by using the model validation function, which is described in 6.8.5, "Model validation" on page 220.

6.8.5 Model validation

To do model validation, complete the validation data set when you import the data set by completing the following steps:

1. Select the trained model, and click **Validate Trained Model**, as shown in Figure 6-82.



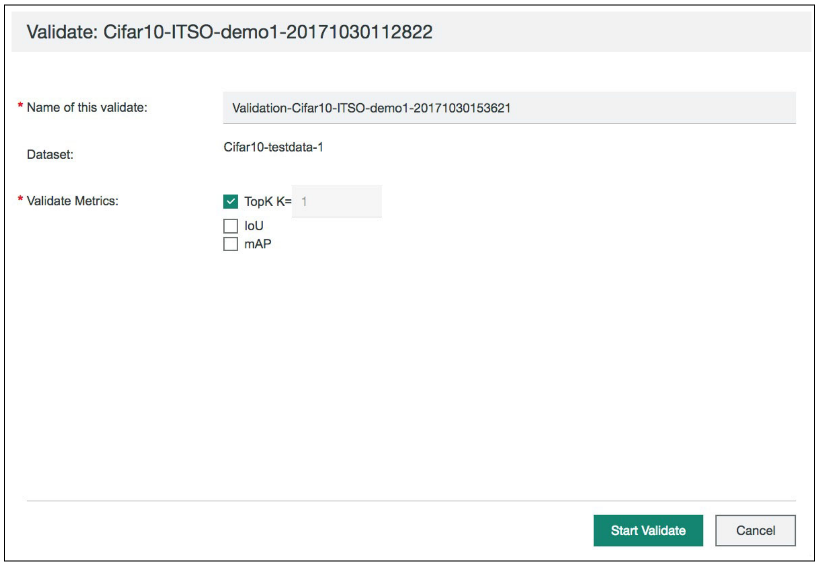
Cifar10-ITSO-demo1
For ITSO testing

Trained Model Name	State	Dataset	Submitted
Cifar10-ITSO-demo1-20171030115352	Finished	Cifar10-testdata-1	2017-10-30, 23:53:56
Cifar10-ITSO-demo1-20171030112822	Finished	Cifar10-testdata-1	2017-10-30, 23:28:25

Showing 1 to 2 of 2 entries

Figure 6-82 Starting a validation job

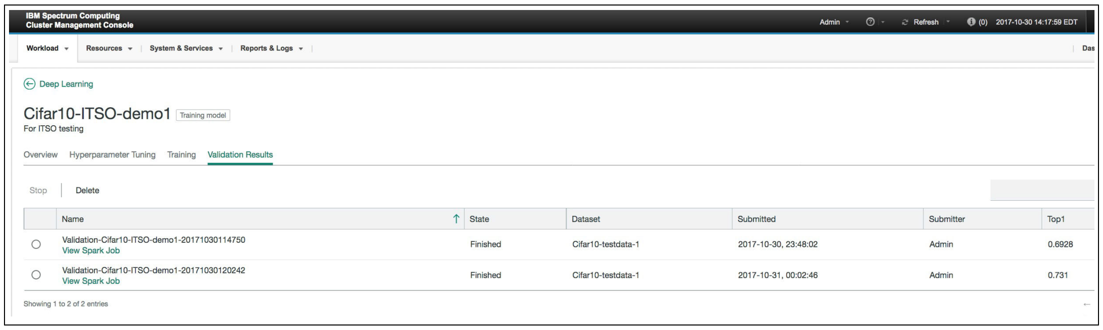
2. Because this is an image classification scenario, select **Top K** and set the number to 1, as shown in Figure 6-83 on page 221.


Validate: Cifar10-ITSO-demo1-20171030112822

* Name of this validate:	Validation-Cifar10-ITSO-demo1-20171030153621
Dataset:	Cifar10-testdata-1
* Validate Metrics:	✓ TopK K= 1
	☐ IoU
	☐ mAP

Figure 6-83 Selecting the validation method

When the two validation jobs are complete, click **Validation Results** to get the result, as shown in Figure 6-84.



Cifar10-ITSO-demo1 Training model
For ITSO testing

	Name	State	Dataset	Submitted	Submitter	Top1
○	Validation-Cifar10-ITSO-demo1-20171030114750 View Spark Job	Finished	Cifar10-testdata-1	2017-10-30, 23:48:02	Admin	0.6928
○	Validation-Cifar10-ITSO-demo1-20171030120242 View Spark Job	Finished	Cifar10-testdata-1	2017-10-31, 00:02:46	Admin	0.731

Showing 1 to 2 of 2 entries

Figure 6-84 Checking the validation result

From the output that is shown in Figure 6-84, the accuracy of the original trained model is 69.28%. After you tune the learning rate from 0.01 to 0.005, the accuracy of the new trained model is 73.1%.

6.8.6 Model tuning

This section describes the DLI's hyperparameter tuning function to tune the model. Complete the following steps:

1. In the training model window, click **Hyperparameter Tuning** →**New Tuning**, as shown in Figure 6-85.

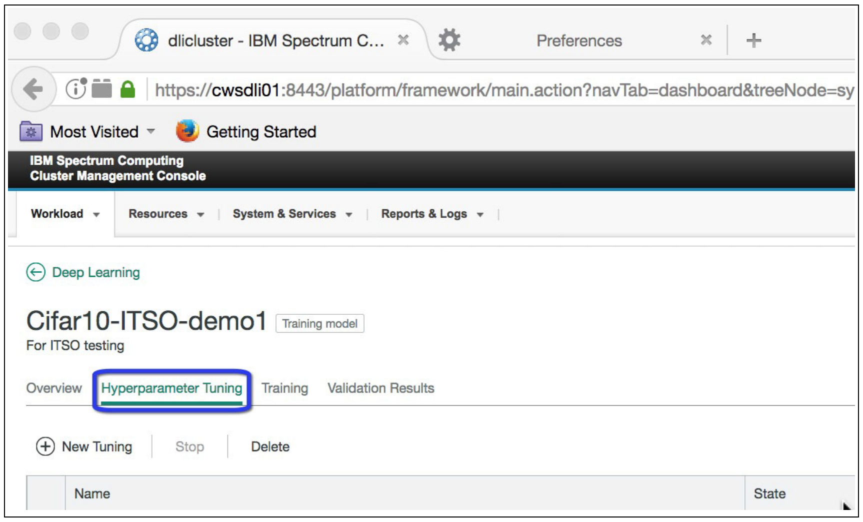


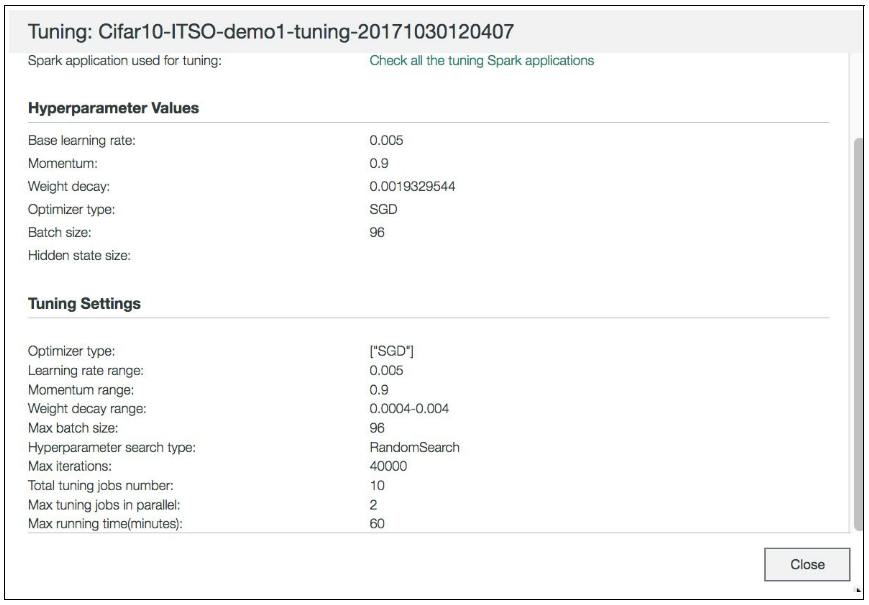
Figure 6-85 Starting a hyperparameter tuning task

The Hyperparameter Tuning window opens. In this use case, we tune only the Weight decay to explain how DLI works. Table 6-10 shows the parameter configuration for this tuning task.

Table 6-10 Parameters for this model tuning task

Parameter	Value
Name of tuning attempt	Cifar10-ITSO-demo1-20171030120407
Hyperparameter search type	Random Search
Framework	CaffeOnSpark
Optimizer type	SGD
Learning rate range	0.005
Weight decay range	0.0004-0.004 (it was 0.0004 in previous models)
Momentum range	0.9
Max batch size	96
Number of workers	1
GPUs per worker	1
Max iterations	40000
Total tuning jobs number	10 (default)
Max tuning jobs in parallel	2 (default)
Max running time (minutes)	60 (default)

2. Click **Start Tuning** and wait for several minutes for this tuning task to complete. Select the tuning task and click the name. The window shows the tuning result, as shown in Figure 6-86.



Tuning: Cifar10-ITSO-demo1-tuning-20171030120407

Spark application used for tuning:	Check all the tuning Spark applications

Hyperparameter Values

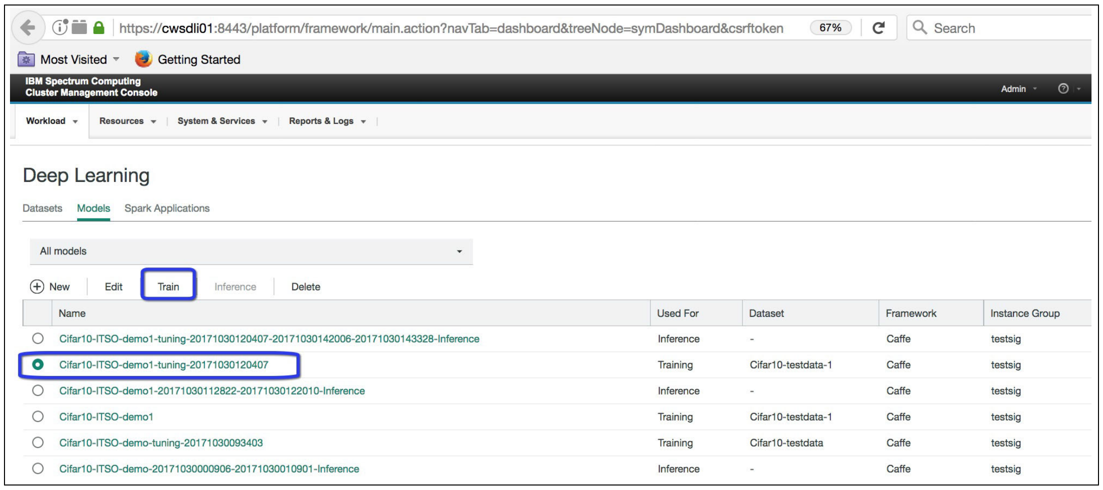
Base learning rate:	0.005
Momentum:	0.9
Weight decay:	0.0019329544
Optimizer type:	SGD
Batch size:	96
Hidden state size:	

Tuning Settings

Optimizer type:	["SGD"]
Learning rate range:	0.005
Momentum range:	0.9
Weight decay range:	0.0004-0.004
Max batch size:	96
Hyperparameter search type:	RandomSearch
Max iterations:	40000
Total tuning jobs number:	10
Max tuning jobs in parallel:	2
Max running time(minutes):	60

Figure 6-86 Viewing a tuning result

During this tuning task, DLI identifies the best Weight decay value as 0.0019329544. DLI created a training model with the tuned hyperparameters. Figure 6-87 shows the new item in the training model list.



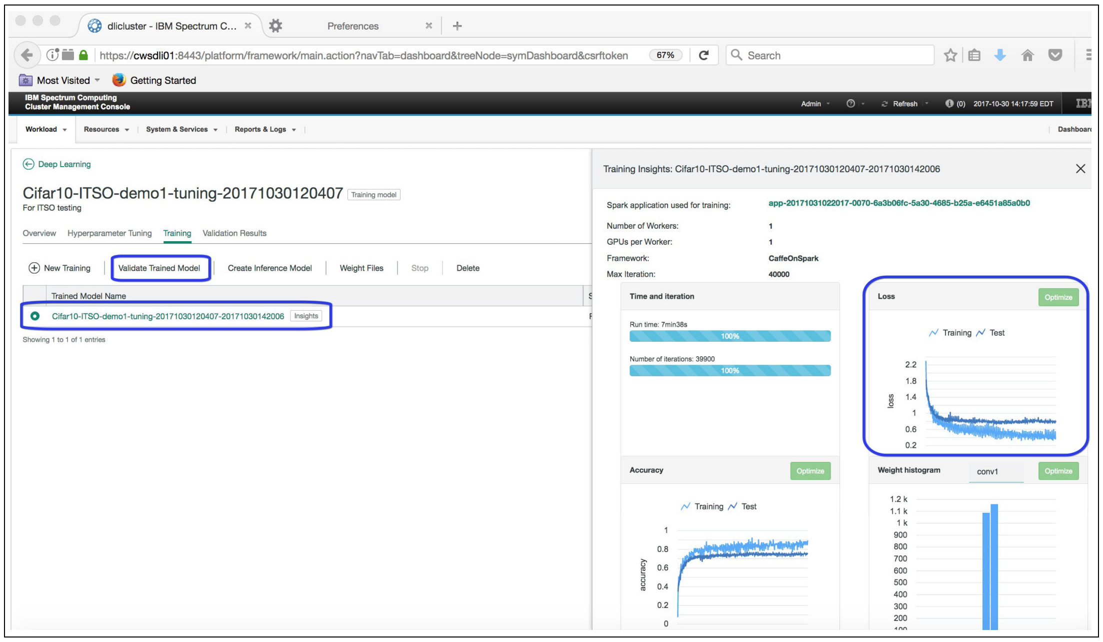
https://cwsdli01:8443/platform/framework/main.action?navTab=dashboard&treeNode=symDashboard&csrftoken

Deep Learning

	Name	Used For	Dataset	Framework	Instance Group
○	Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006-20171030143328-Inference	Inference	-	Caffe	testsig
●	Cifar10-ITSO-demo1-tuning-20171030120407	Training	Cifar10-testdata-1	Caffe	testsig
○	Cifar10-ITSO-demo1-20171030112822-20171030122010-Inference	Inference	-	Caffe	testsig
○	Cifar10-ITSO-demo1	Training	Cifar10-testdata-1	Caffe	testsig
○	Cifar10-ITSO-demo-tuning-20171030093403	Training	Cifar10-testdata	Caffe	testsig
○	Cifar10-ITSO-demo-20171030000906-20171030010901-Inference	Inference	-	Caffe	testsig

Figure 6-87 Starting a train job base on the tuned model

3. Start a new training job with this model, as shown in the MAO dashboard in Figure 6-88.



Training Insights: Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006

Spark application used for training:	app-20171031022017-0070-6a3b06fc-5a30-4685-b25a-e6451a85a0b0
Number of Workers:	1
GPUs per Worker:	1
Framework:	CaffeOnSpark
Max Iteration:	40000

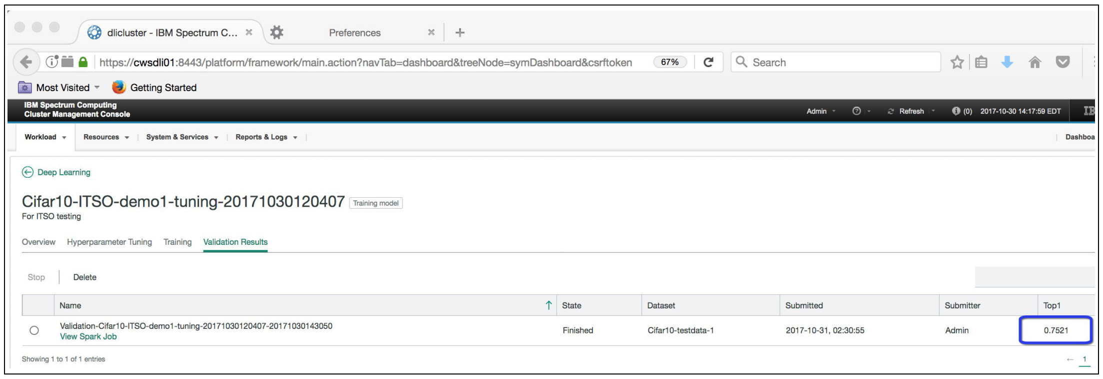
Time and iteration

Run time: 7min38s 100%

Number of iterations: 39900 100%

Figure 6-88   Monitoring and Optimization window for this training model

4. Select this trained job (Figure 6-88) and click **Validation Trained Model** to trigger a validation job. The result is shown in Figure 6-89.



Name	State	Dataset	Submitted	Submitter	Top1
Validation-Cifar10-ITSO-demo1-tuning-20171030120407-20171030143050	Finished	Cifar10-testdata-1	2017-10-31, 02:30:55	Admin	0.7521

Figure 6-89   Viewing a validation result for the tuned model

The output shows that the accuracy is 75.21%, which means that with the DLI's hyperparameter tuning's help, the accuracy changed from 73.1% to 75.21%. Table 6-11 on page 225 shows the summary of accuracy for the different scenarios.

Table 6-11 Summary of accuracy result for different trained jobs

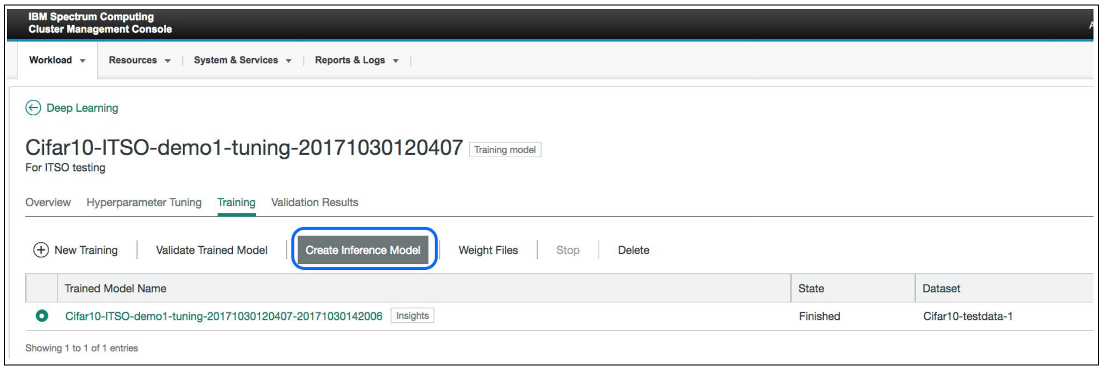
Scenario	Operation	Accuracy
Baseline	Not applicable	69.28%
DLI suggestion during training	Learning rate: 0.01 - 0.005	73.1%
DLI suggestion during hyperparameter search	Weight decay: 0.0004 - 0.0019329544	75.21%

Note: This is a straightforward example to demonstrate how to use DLI to tune a neural network model. In production cases, DLI can help you to keep to the correct tuning direction and get better accuracy.

6.8.7 Model prediction

When you are satisfied with the accuracy of one trained model, create an inference model for prediction by completing the following steps:

1. Select one trained model and click **Create Inference Model**, as shown in Figure 6-90.



Cifar10-ITSO-demo1-tuning-20171030120407

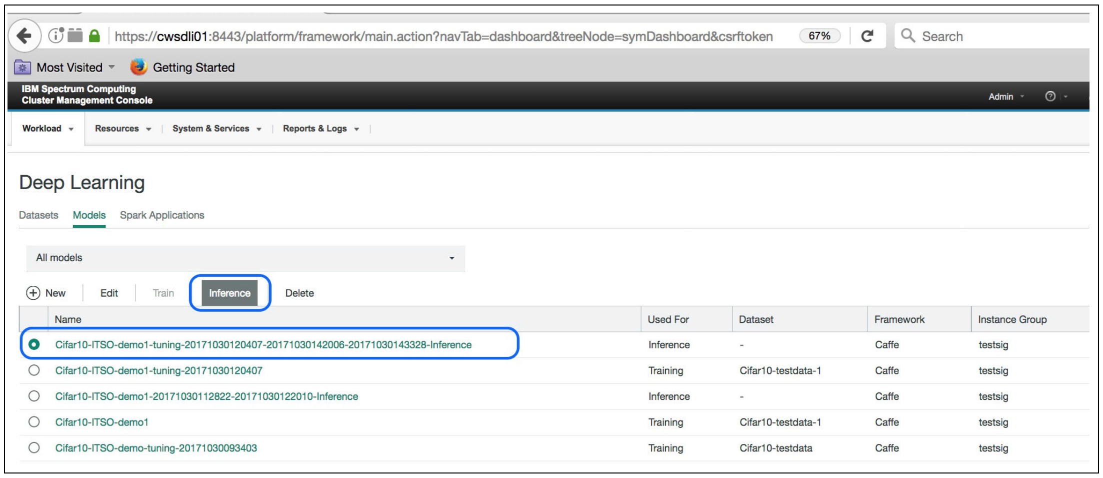
For ITSO testing

Trained Model Name	State	Dataset
Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006	Finished	Cifar10-testdata-1

Showing 1 to 1 of 1 entries

Figure 6-90 Generating an inference model

2. DLI creates a new inference model, and you can find it in the model list, as shown in Figure 6-91. Select this model and click **Inference**.

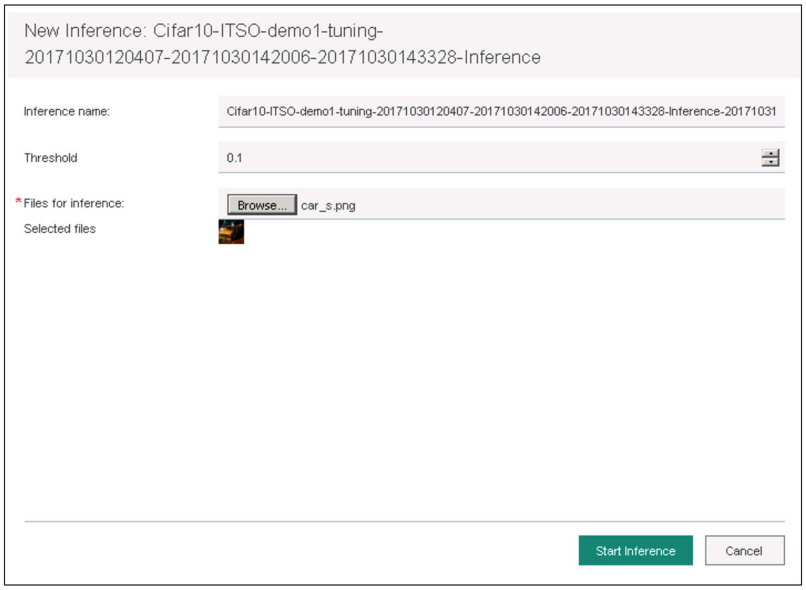


	Name	Used For	Dataset	Framework	Instance Group
●	Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006-20171030143328-Inference	Inference	-	Caffe	testsig
○	Cifar10-ITSO-demo1-tuning-20171030120407	Training	Cifar10-testdata-1	Caffe	testsig
○	Cifar10-ITSO-demo1-20171030112822-20171030122010-Inference	Inference	-	Caffe	testsig
○	Cifar10-ITSO-demo1	Training	Cifar10-testdata-1	Caffe	testsig
○	Cifar10-ITSO-demo-tuning-20171030093403	Training	Cifar10-testdata	Caffe	testsig

Figure 6-91 Starting an inference job

3. Complete the threshold and select the files on which you want to do prediction. Click **Start Inference**, as shown in Figure 6-92.

Note: If you set the threshold to 0.1, this means DLI displays all the classes with similar possibilities to >= 10%.

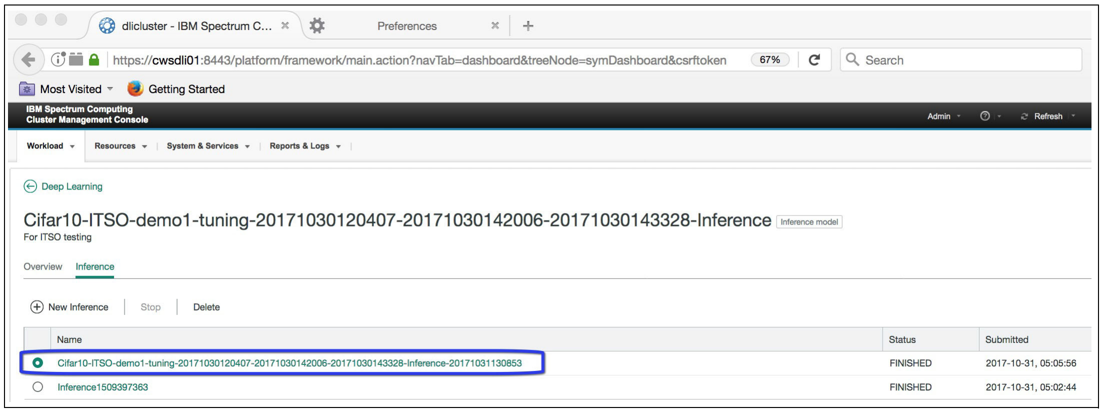


New Inference: Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006-20171030143328-Inference

Inference name:	Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006-20171030143328-Inference-20171031
Threshold	0.1
* Files for inference:	Browse... car_s.png
Selected files	

Figure 6-92 Setting the parameters for an inference job

4. When this inference job completes, DLI creates an inference result item in a reference list. Click the name and you can see the result, as shown in Figure 6-93 on page 227.


dlicluster - IBM Spectrum C...
Preferences
https://cwsdli01:8443/platform/framework/main.action?navTab=dashboard&treeNode=symDashboard&csrftoken
67%
Search
Most Visited
Getting Started
IBM Spectrum Computing
Cluster Management Console
Admin
Refresh
Workload
Resources
System & Services
Reports & Logs
Deep Learning

Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006-20171030143328-Inference Inference model

For ITSO testing

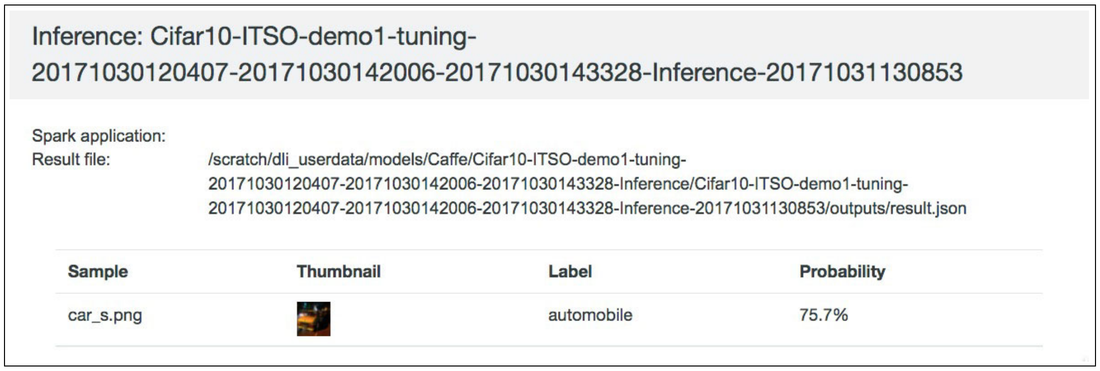
Overview Inference

New Inference | Stop | Delete

	Name	Status	Submitted
●	Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006-20171030143328-Inference-20171031130853	FINISHED	2017-10-31, 05:05:56
○	Inference1509397363	FINISHED	2017-10-31, 05:02:44

Figure 6-93 Selecting an inference job

Figure 6-94 shows the prediction result of the inference job.



Inference: Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006-20171030143328-Inference-20171031130853

Spark application:
Result file: /scratch/dli_userdata/models/Caffe/Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006-20171030143328-Inference/Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006-20171030143328-Inference-20171031130853/outputs/result.json

Sample	Thumbnail	Label	Probability
car_s.png		automobile	75.7%

Figure 6-94 Viewing the result of the inference job

Prediction through the RESTful API

You can also use the RESTful API to call the inference service. Example 6-17 shows how to implement it with a Python script and integrate the application to call the DL prediction service.

Example 6-17 Using a RESTful API to call the DLI inference service

```
import requests
from requests.packages.urllib3.exceptions import InsecureRequestWarning
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)

import base64
import json
import time
import os

###Set variables, changed by real requirement###
lv_host='cwsdli01.dli.com'
lv_inference_model='Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006-20171030143328-Inference'
lv_threshold=0.1
lv_img_dir='/demodata/userdata/cifar10_test/'
lv_user='Admin'
lv_passwd='Admin'


###Logon###
logonUrl='https://'+lv_host+':8643/platform/rest/conductor/v1/auth/logon'
base64string = base64.encodestring('%s:%s' % (lv_user, lv_passwd)).replace('\n', '')
auth='Basic ' + base64string
headers = {'Authorization': auth}
req = requests.Session()
r = req.get(logonUrl, headers=headers, verify=False)
if 200 != r.status_code:
    print('logon failed')
print('logon to rest server succeed')

###Get file list from directory###
files = {}
for idirpath, dirnames, filenames in os.walk(lv_img_dir, followlinks=True):
    count = 0
    for filep in filenames:
        count += 1
        tmp_file = os.path.join(lv_img_dir,filep)
        files['file'+str(count)] = open(tmp_file,'rb')
print files

###Start a new inference task###
startInferenceUrl='https://' + lv_host + ':9243/platform/rest/deeplearning/v1/inferences/startpredict'
inf_name='Inference' + str(int(time.time()))
data={'predictname':inf_name, 'modelname':lv_inference_model,'threshold':lv_threshold}
r = req.post(startInferenceUrl,data=data, files=files,verify=False)
print('Start inference task: %s'%inf_name)
print('Waiting for Inference spark application lanuched...')

###Check inference task status###
getInferenceStatusUrl='https://' + lv_host + ':9243/platform/rest/deeplearning/v1/inferences/' + inf_name
getInferenceResUrl='https://' + lv_host + ':9243/platform/rest/deeplearning/v1/inferences/' + inf_name + '/predicts'
commonHeaders={'accept': 'application/json'}
r = req.get(getInferenceStatusUrl,headers=commonHeaders, verify=False)
json_out=r.json()

if json_out.has_key('status'):
    while json_out['status'] != 'FINISHED':
        print('Inference task current status: %s'%json_out['status'])
        r = req.get(getInferenceStatusUrl,headers=commonHeaders, verify=False)
        json_out=r.json()
    ###This task is finished and print output###
    print('Inference task is FINISHED')
    r = req.get(getInferenceResUrl,headers=commonHeaders, verify=False)
    print('Here is the inference output:')
    print(json.dumps(r.json(), sort_keys=True, indent=4))
else:
    print('Failed to get Inference task status, exit')
```

Example 6-18 shows the output after running this script. The output can be used by your application directly.

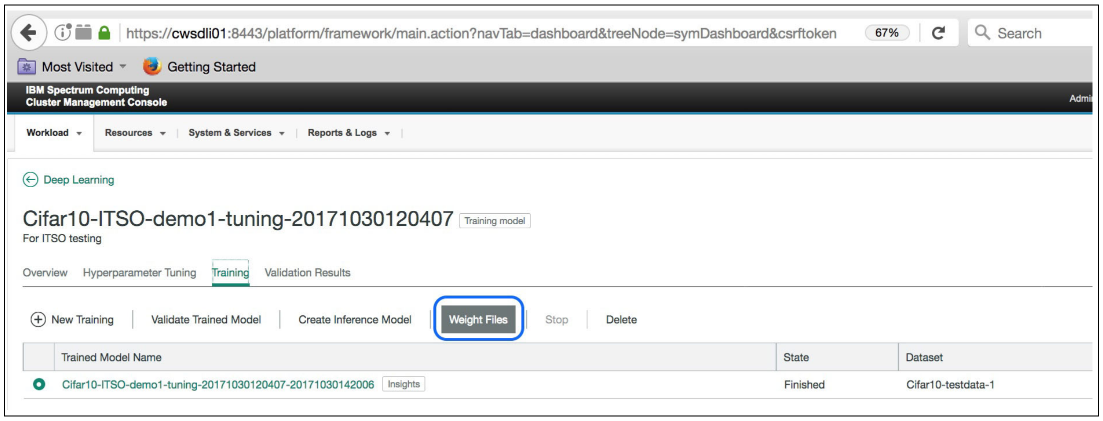
Example 6-18 Output of the prediction script

```
#python cifar10_inference.py
logon to rest server succeed
{'file1': <open file '/demodata/userdata/cifar10_test/car_s.png', mode 'rb' at 0x3fffae6abf60>}
Start inference task: Inference1509378367
Waiting for Inference spark application lanuched...
...
Inference task is FINISHED
Here is the inference output:
{
    "classificationResults": [
        {
            "image": true,
            "inputPath":
"/scratch/dli_userdata/models/Caffe/Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006-20171030143328-Inference/
Inference1509378367/inputs",
            "results": [
                {
                    "label": "automobile",
                    "prob": 0.757
                }
            ],
            "sampleId": "car_s.png"
        }
    ],
    "type": "classification"
}
```

6.8.8 Training model weight file management

When one training model job is complete, download its weight file. This weight file is used to deploy the inference model in any other server or use it for continuous training. Then, complete the following steps:

1. Select a trained model and click **Weight Files**, as shown in Figure 6-95.

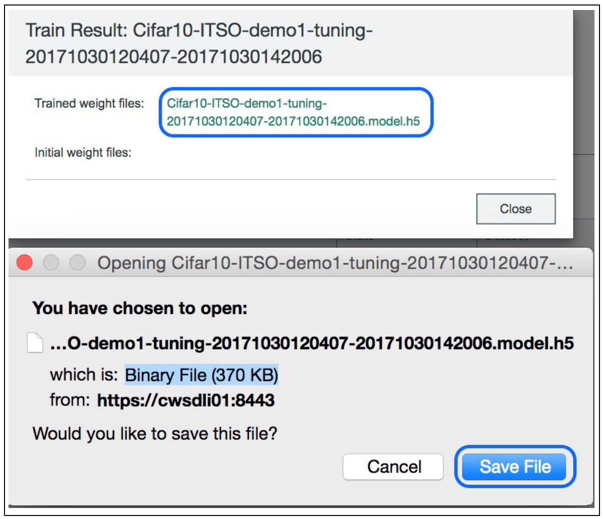


Cifar10-ITSO-demo1-tuning-20171030120407 Training model
For ITSO testing

Trained Model Name	State	Dataset
Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006 Insights	Finished	Cifar10-testdata-1

Figure 6-95 Downloading the weight file

2. A window opens and shows all the weight files. Click any of them to download, as shown in Figure 6-96.


Train Result: Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006

Trained weight files: Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006.model.h5

Initial weight files:

You have chosen to open:

...O-demo1-tuning-20171030120407-20171030142006.model.h5

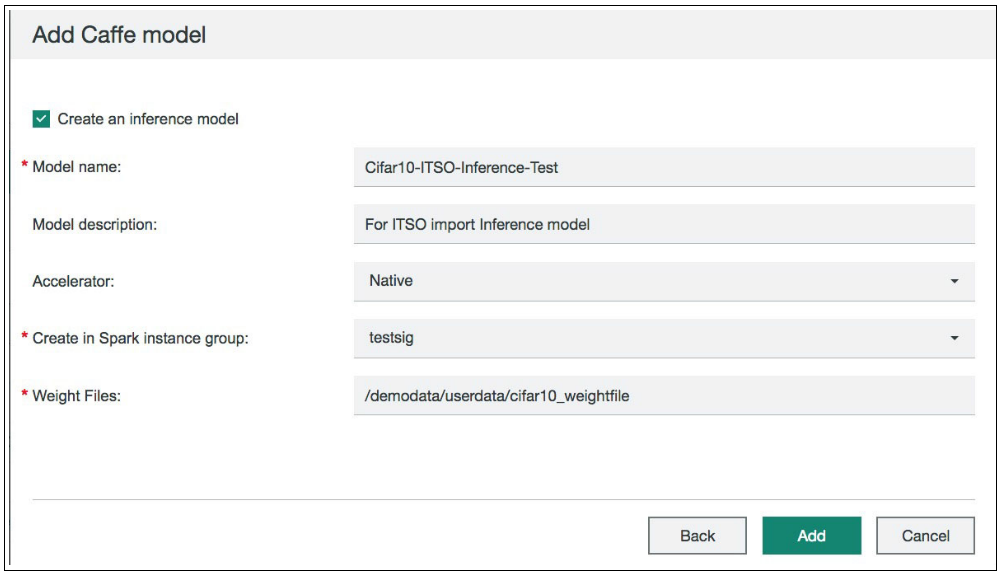
which is: Binary File (370 KB)

from: https://cwsdli01:8443

Would you like to save this file?

Figure 6-96 Downloading a weight file

3. Transfer this file to another server, for example, upload it to `/demodata/userdata/cifar10_weightfile`, as shown in Figure 6-97.

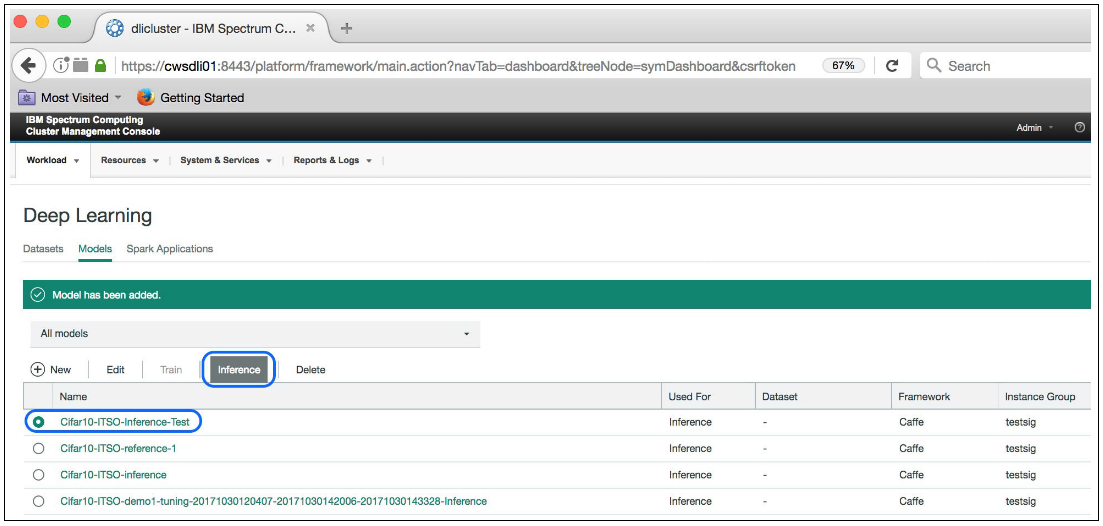

Add Caffe model

☑ Create an inference model

* Model name:	Cifar10-ITSO-Inference-Test
Model description:	For ITSO import Inference model
Accelerator:	Native
* Create in Spark instance group:	testsig
* Weight Files:	/demodata/userdata/cifar10_weightfile

Figure 6-97 Generating an inference model with one existing weight file

4. Click **Add** to add one new inference model, as shown in Figure 6-98.


dlicluster - IBM Spectrum C...
https://cwsdli01:8443/platform/framework/main.action?navTab=dashboard&treeNode=symDashboard&csrftoken
67%
Search
Most Visited
Getting Started
IBM Spectrum Computing
Cluster Management Console
Admin
Workload
Resources
System & Services
Reports & Logs
Deep Learning
Datasets
Models
Spark Applications

Model has been added.

	Name	Used For	Dataset	Framework	Instance Group
●	Cifar10-ITSO-Inference-Test	Inference	-	Caffe	testsig
○	Cifar10-ITSO-reference-1	Inference	-	Caffe	testsig
○	Cifar10-ITSO-inference	Inference	-	Caffe	testsig
○	Cifar10-ITSO-demo1-tuning-20171030120407-20171030142006-20171030143328-Inference	Inference	-	Caffe	testsig

Figure 6-98 Viewing an inference model list and starting an inference job

This inference model can be used for prediction by using the GUI or the RESTful API.

Case scenarios: Using IBM PowerAI

To identify opportunities for using deep learning (DL) and IBM PowerAI, this chapter describes three different use cases that are based on our field experiences. For each scenario, we describe the requirement, solution, and benefits.

Note: The scenarios in this chapter make many assumptions regarding the number of servers, networking, operating systems, open source software, and other items to help you understand what you can do with IBM PowerAI DL. There are many ways to help solve your challenges, and the scenarios illustrate one of the many ways.

The scenarios that are described in this chapter shows how to use IBM PowerAI, but do not represent customers' configurations for their productions systems. Even though there might be some likeness or similarities to the solutions that are represented in this chapter, it is purely coincidental.

This chapter contains the following topics:

7.1 Use case one: Bare metal environment

In many modern industries, companies require and depend on real-time responses from their applications and systems to maintain a successful business. Such customers are receptive to new technology (such as DL) and have expectations about any new technology that quickly contributes to satisfying their business goals and requirements.

7.1.1 Customer requirements

When users and data scientists initially experiment with implementation, they do not consider the infrastructure that hosts their DL frameworks because their focus on the environment is the accuracy results that their models produce. However, as their familiarity grows and they implement more complex models with larger data sets, it is inevitable that the training phase takes longer. As a result, their focus shifts to the infrastructure under the frameworks. They start to evaluate differences in performance depending on the servers' configurations. As a criteria for vendor selection, customers tend to evaluate the performance of one specific framework, and testing a specific data set such as an *imagenet*.

Here are examples of customer requests:

- Capability to host an on-premises DL infrastructure for real-time analysis of large quantities of internal data
- Capability to run frameworks, such as TensorFlow and Torch, on the DL platform
- Proven expectation of reduction for the training time compared to other providers (both for on-premises and cloud providers)
- Reduced total cost of ownership (TCO) compared to other solutions

7.1.2 IBM solution

The customer requirements that are described in 7.1.1, "Customer requirements" on page 232 can be satisfied by implementing the following environment:

- Hardware: IBM Power System S822LC for High Performance Computing server
- Operating system: Ubuntu 16.04
- Software: IBM PowerAI Release 4, including TensorFlow and Torch
  - CUDA: Version 8
  - CUDA Deep Neural Network (cuDNN): Version 6
  - NVDIA driver: Version 384.66
- Network: 10 Gb Ethernet (see Figure 7-1)

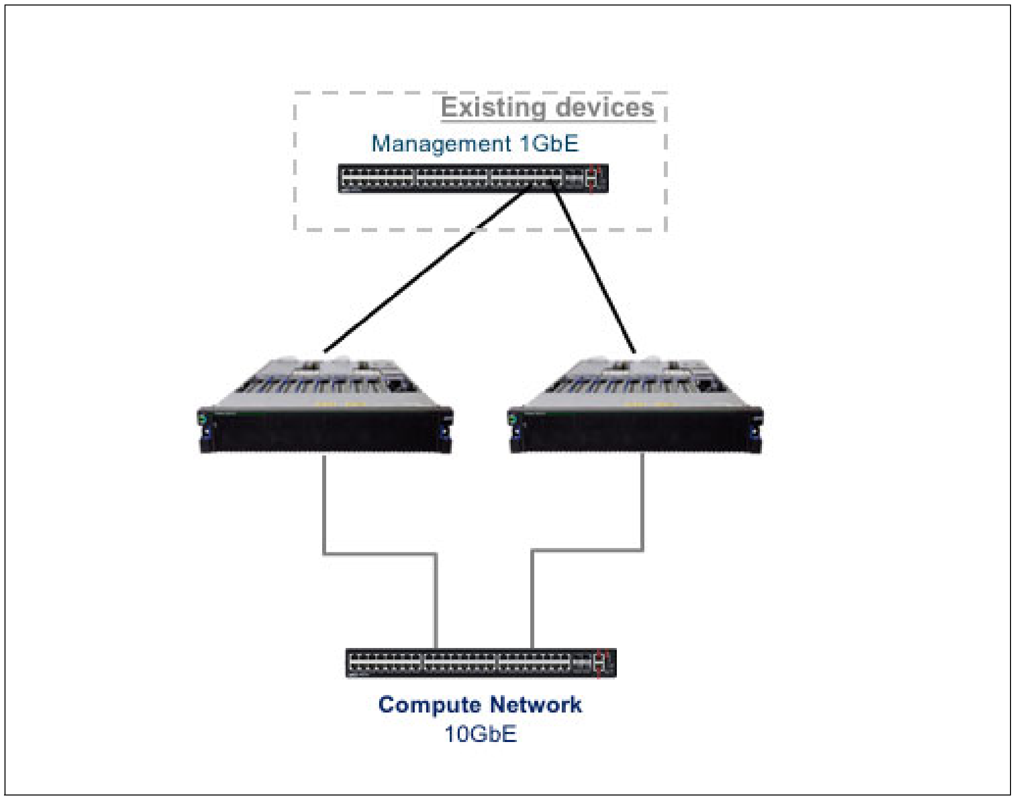


Figure 7-1 Network configuration for the bare metal environment

7.1.3 Benefits

As a result of the proposed solution that is described in 7.1.2, “IBM solution” on page 232, there are a few benefits:

- Better training performance for TensorFlow and Torch on Power S822LC for High Performance Computing with NVLink (proven by proof of concept)
- Evaluation item: Training time with single graphical processing unit (GPU), which results in faster performance than other offerings
- Lower running cost than a public cloud for a 3-year period
- Enterprise-ready package and quick installation of DL frameworks and related libraries

Figure 7-2 illustrates the image of the installation for IBM PowerAI Release 4 and IBM PowerAI Vision on a bare metal server.

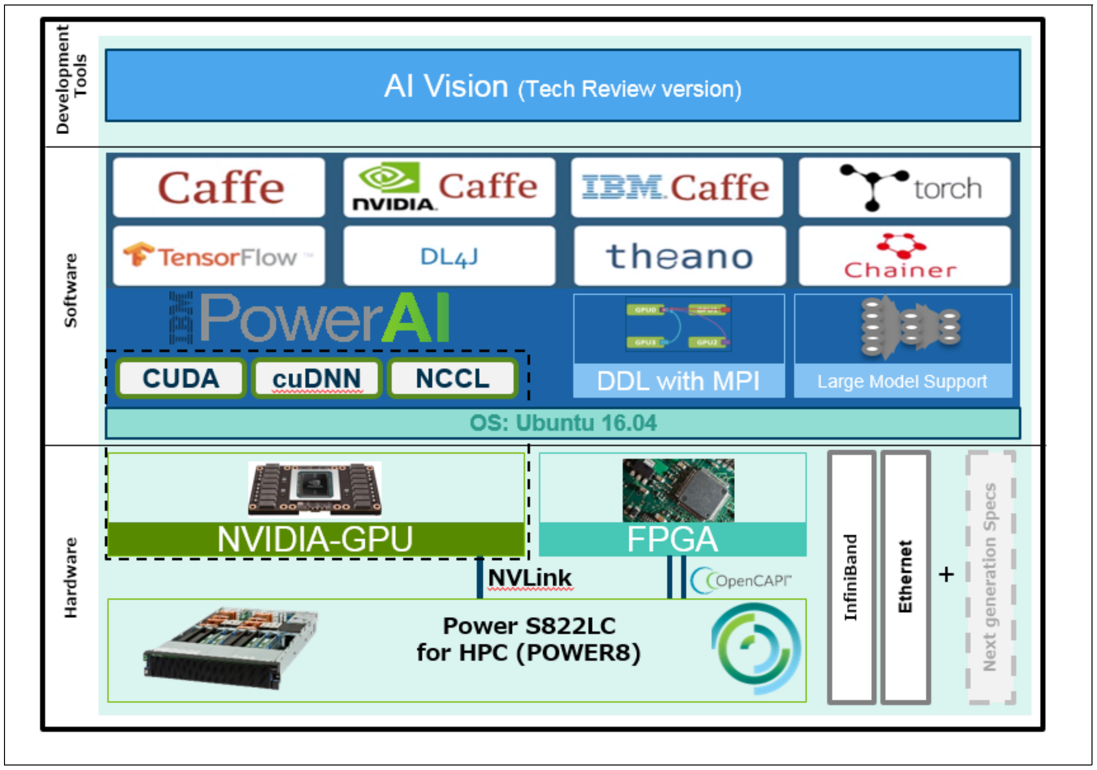


Figure 7-2 IBM PowerAI V1.4 integrated with artificial intelligence Vision on a bare metal machine

Figure 7-3 shows the image of the installation for IBM PowerAI Enterprise with all the benefits from IBM PowerAI Release 5 on bare metal server due to the new option of Red Hat Enterprise Linux with Release 5.

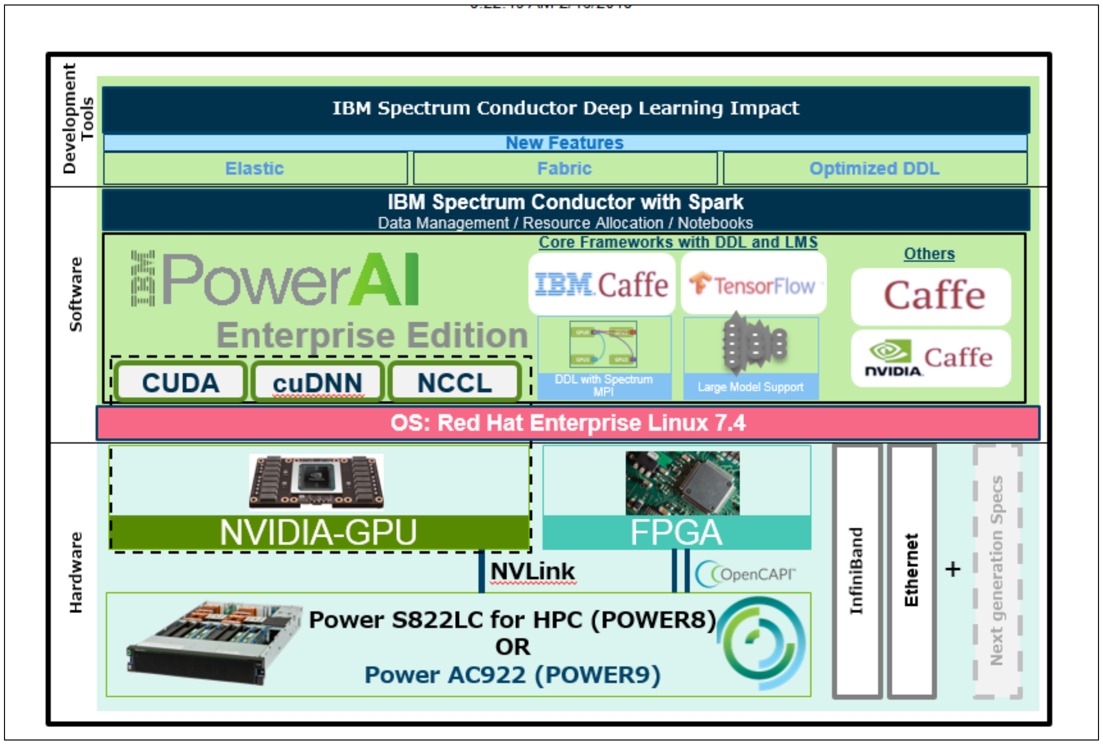


Figure 7-3 IBM PowerAI V1.5 integrated with IBM Spectrum Conductor with Spark and Deep Learning Impact

7.2 Use case two: Multitenant environment

Cloud computing is the standard platform for many companies from an IT consumer perspective as a viable alternative to hosting internal environments. Alternatively, from the viewpoint of IT providers, platforms must be on-premises servers and maintain the highest level of system and data security. Regarding data protection, healthcare institutions and research institutions are frequently mandated to host data within their institutions or within specific countries because of the required treatment of personal, financial, or regulatory information.

It is a common requirement to securely use servers for multiple and distinct customers. This section introduces the multitenancy environment.

7.2.1 Customer requirements

Here is one of the examples of running NVIDIA-Docker on Power S822LC for High Performance Computing and IBM PowerAI on a Docker container. The purpose for using this NVIDIA-Docker solution generally is for cloud services, and internal use across the departments within the company.

Here are examples of customer requests:

- Platform supports the current NVIDIA GPU.
- Use the servers as a shared platform across the departments.
- Use the platform for DL and other open source applications.

- Address concerns about the potential bottleneck of GPU computing that is caused by limited memory and storage I/O.
- High compatibility with existing storage file systems (for example, IBM Spectrum Scale).

7.2.2 IBM solution

In this situation, the flexibility of the platform is highly requested. Because of the GPU-enabled Docker technology, that is, NVIDIA-Docker, IBM can provide the following components in the solution:

- Hardware: FIve Power S822LC for High Performance Computing servers
- Host operating system: Red Hat Enterprise Linux 7.3
- Software:
  - IBM PowerAI Release 4
  - CUDA: Version 8
  - cuDNN: Version 6
  - NVDIA driver: Version 384.66

  NVIDIA-Docker with a guest operating system image: Ubuntu and CentOS
- Network: Ethernet (see Figure 7-4)

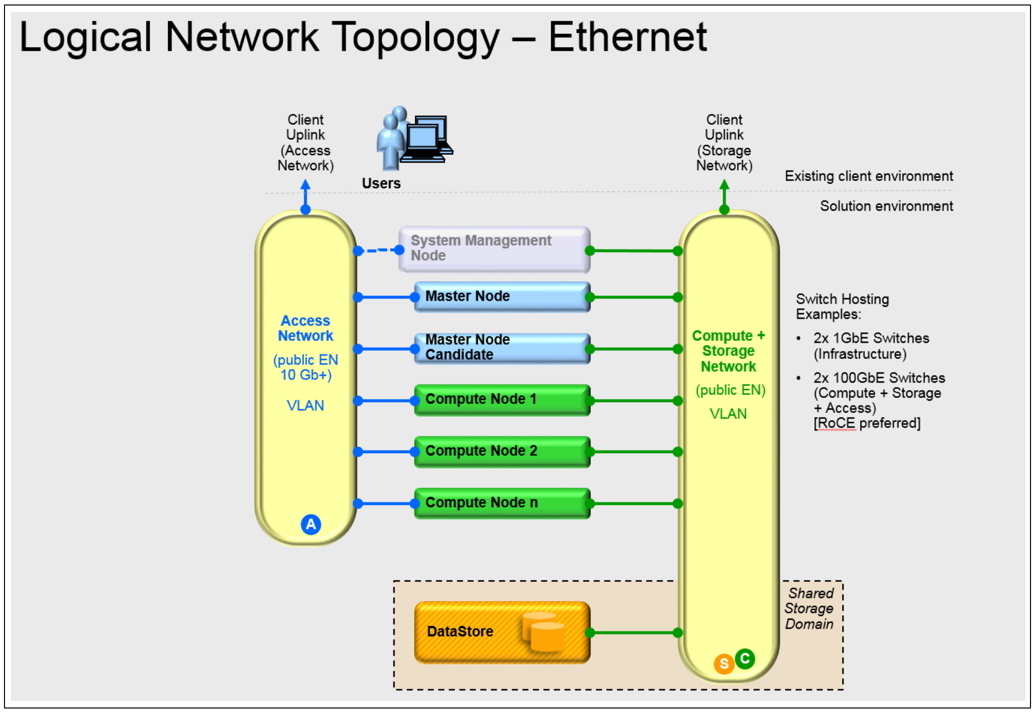

Logical Network Topology – Ethernet
Client Uplink (Access Network)
Users
Client Uplink (Storage Network)
Existing client environment
Solution environment
Access Network
(public EN 10 Gb+)
VLAN
A
System Management Node
Master Node
Master Node Candidate
Compute Node 1
Compute Node 2
Compute Node n
Compute + Storage Network
(public EN)
VLAN
Switch Hosting Examples:
• 2x 1GbE Switches (Infrastructure)
• 2x 100GbE Switches (Compute + Storage + Access) [RoCE preferred]
DataStore
Shared Storage Domain
S
C

Figure 7-4 Logical network configuration image for the NVIDIA-Docker case

7.2.3 Benefits

From the solution that is proposed in 7.2.2, “IBM solution” on page 236, the following benefits can be achieved:

- Establishment of a multitenant environment for the departments in the company
- Easier, faster implementation by the IBM PowerAI toolkit with optimized DL libraries

- Dedicated use of installed GPUs for each Docker container
- CPU-GPU NVLink support
- Reduced bottleneck in storage I/O with the InfiniBand connection

Figure 7-5 shows the architectural diagram of NVIDIA-Docker that is installed on Red Hat Enterprise Linux to obtain the multitenancy environment for business use.

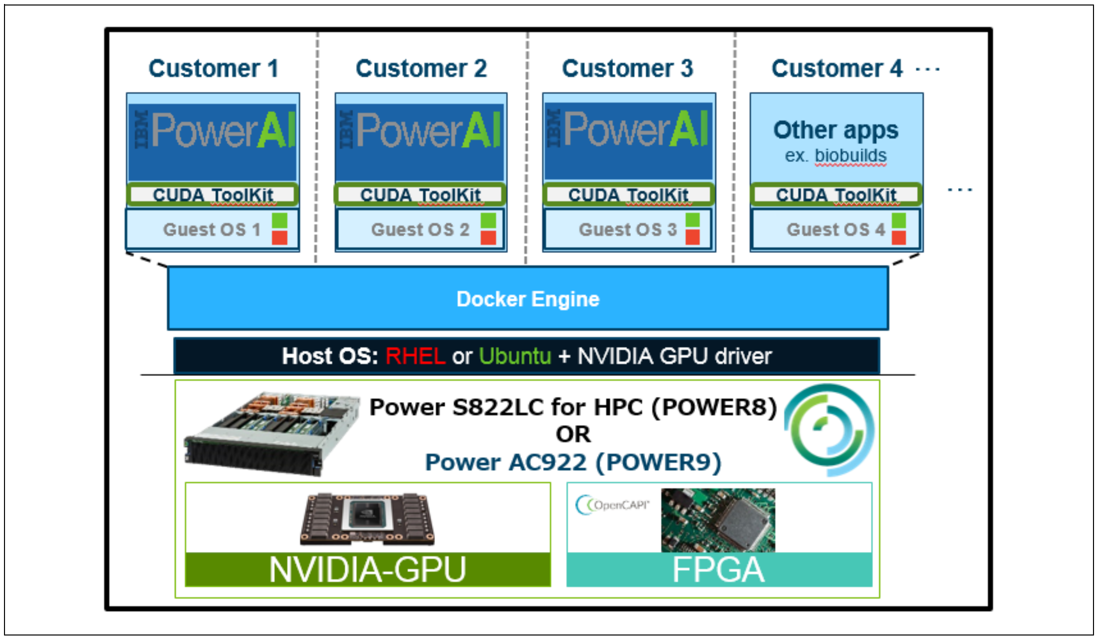


Figure 7-5 NVIDIA-Docker solution for IBM PowerAI

7.3 Use case three: High-performance computing environment

IBM Power Systems servers have been used within high-performance computing (HPC) environments since the days of the original IBM RS/6000® SP clusters. The scalability and flexibility of the hardware platform makes it an attractive choice for the HPC community. This relationship continues with the advent and popularity of DL. The usage of GPU-enabled IBM Power Systems for HPC environments is increasing. These servers provide an alternative model compared to traditional HPC environments, where it was expected to require the largest server models with the most quantity of resources, now the GPU-enabled models provide the same processing power in a much smaller footprint. Where previously rooms were required, now we can talk in quantities of racks.

Presently, there are many data scientists conducting DL frameworks in such clusters. The largest announced environment at the time of writing is the one that will be adopted for the Collaboration of Oak Ridge, Argonne, and Livermore (CORAL) project for the US Department of Energy. This environment will be constructed by using the recently announced POWER9 based Power AC922 machines. Especially for the DL frameworks in HPC, users manage the environment with job scheduling software, such as IBM Spectrum LSF®, and recently with Spark technology to enable distributed parallel processing.

7.3.1 Customer requirements

Here are examples of customer requests:

- Adopt the latest technology available.
- Support the following components:
  - NVIDIA GPU
  - InfiniBand network
  - Open source DL frameworks, such as Caffe, Torch, or TensorFlow
- Use other vendors technologies.
- Accommodate the HPC cluster with Spark.
- Mitigation of bottlenecks relevant to networking between devices.

7.3.2 IBM solution

To satisfy the customer requirements that are shown in 7.3.1, “Customer requirements” on page 238, IBM provides experiences and solutions that can be integrated with other software. As a basic proposal (Figure 7-6), IBM can provide a solution with the following components:

- Hardware: Thirty Power S822LC for High Performance Computing servers
- Host operating system: Ubuntu 16.04
- Software: IBM PowerAI Release 4 with distributed deep learning (DDL) and large model support (LMS) functions:
  - CUDA: Version 8
  - cuDNN: Version 6
  - NVDIA driver: Version 384.66
  - IBM Spectrum Conductor with Spark
  - LSF
- Network: InfiniBand switch and cable for EDR

7.3.3 Benefits

By providing an IBM PowerAI end-to-end solution consisting of all of the components that are shown in 7.3.2, “IBM solution” on page 238, customers can obtain these benefits:

- Reduce the training time by double digits with the latest technology that is available.
- LMS and DDL provide better scalability of GPUs.
- Easily alter both the data and the artificial intelligence (AI) model by using an on-premises Power System solution.
- Easier and faster implementation with the IBM PowerAI toolkit with optimized DL libraries.
- Easy management and control of resources with job scheduler.

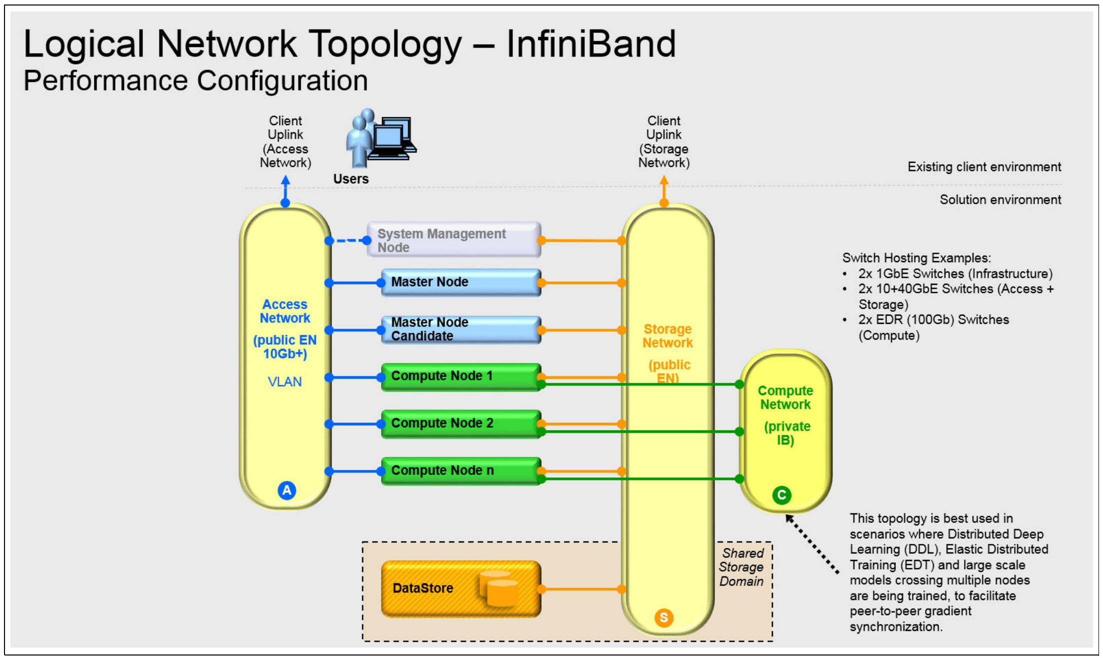

Logical Network Topology – InfiniBand
Performance Configuration
Client Uplink (Access Network)
Users
Client Uplink (Storage Network)
Existing client environment
Solution environment
System Management Node
Master Node
Master Node Candidate
Compute Node 1
Compute Node 2
Compute Node n
Access Network (public EN 10Gb+) VLAN
A
Storage Network (public EN)
S
Compute Network (private IB)
C
DataStore
Shared Storage Domain
Switch Hosting Examples:
• 2x 1GbE Switches (Infrastructure)
• 2x 10+40GbE Switches (Access + Storage)
• 2x EDR (100Gb) Switches (Compute)
This topology is best used in scenarios where Distributed Deep Learning (DDL), Elastic Distributed Training (EDT) and large scale models crossing multiple nodes are being trained, to facilitate peer-to-peer gradient synchronization.

Figure 7-6 Logical network topology

7.4 Conclusion

In the previous sections, we described the key components of the IBM PowerAI Deep Learning framework, summarized the benefits of the GPU-accelerated Power Systems hardware, and provided example installations and implementations of IBM PowerAI versions 4 and 5.

IBM PowerAI makes the field of DL available and consumable to every audience, including the causal user, small businesses, and traditional HPC organizations. The significance of the acceleration that is provided by GPUs results is realized in savings in both processing time and physical footprint. The smaller use cases that use capacity from a cloud provider (see IBM and NVIDIA Team Up on World's Fastest Deep Learning Enterprise Solution) provide a low-cost route for evaluation and implementation, and in the larger cases, the number of physical servers are reduced compared to traditional computing requirements from previous decades.

If you are unfamiliar with the field of DL these example use cases provide context and perspective. Understanding what data you have is key to using DL. What connections, trends, or observations might be hidden from your existing view? Using DL for data processing provides many previously unavailable opportunities and areas of interest for academic, scientific, industrial, retail, and commercial businesses. DL has been growing in popularity for some time, but it is the current ease of adoption that is making it commonplace in many areas of our everyday life.

A

Sentiment analysis code

This appendix provides the code for the sentiment analysis scenario that is presented in Chapter 5, “Working with data and creating models in IBM PowerAI” on page 121, and describes how the files are structured.

This appendix contains the following topic:

- Sentiment analysis with TensorFlow

Sentiment analysis with TensorFlow

This section describes sentiment analysis with the TensorFlow code.

How the code is organized

The sentiment analysis code is organized into three Python (`.py`) files:

- `data_prep.py`: This file is used to prepare the data. You can run it independently and generate the output data set and dictionaries to be used in further steps.
- `sentiment_neural_net.py`: This file contains the functions that are responsible for model creation and training. Because of the way the code is prepared, you can run the training from the `use_neural_net.py` file.
- `use_neural_net.py`: From this file, you can run the training and use the neural network to providing sentences to the **`get_sentiment`** function. If you do not want to run the training function, comment out the line that calls the **`training`** function.

The code is also available on GitHub.

Sentiment analysis code

This section presents the sentiment analysis code.

Example A-1 shows the `data_prep.py` preparation file.

Example A-1 The data_prep.py file

```
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
import numpy as np
import re
import random
import pandas as pd
import pickle
import os
from collections import Counter

lemm = WordNetLemmatizer()


def rand_list(lines, max_value):
    randlist = []
    for _ in range(lines):
        num = random.randint(0, max_value-1)
        while num in randlist:
            num = random.randint(0, max_value - 1)
        randlist.append(num)

    return randlist


def shuffler(input_ds, output_ds):
    df_source = pd.read_csv(input_ds, '<SP>', error_bad_lines=False)
    df_shuffled = df_source.iloc[np.random.permutation(len(df_source))]
    df_shuffled.to_csv(output_ds, 'µ', index=False)

def smaller_dataset_gen(ds, newds, dsrows, num_lines=1000):
    count = 0
    with open(ds, 'r', 5000, 'latin-1') as raw_ds:
        with open(newds, 'w', 5000) as target_ds:
            selected_lines = rand_list(num_lines, dsrows)
            for line in raw_ds:
                if len(selected_lines) == 0:
                    break

                if count in selected_lines:
                    target_ds.write(line)
                    selected_lines.remove(count)
                count += 1

    print("New dataset created with {} lines".format(num_lines))


def clean_dataset(ds, ods):
    with open(ds, 'r', 30000, 'latin-1') as raw_ds:
        with open('tempds.csv', 'w', 20000) as cleaned_ds:
            for line in raw_ds:
                result = re.search('^"(\d)",.*,"(.*)"$', line)
                new_line = result.group(1) + '<SP>' + result.group(2) + '\n'
                cleaned_ds.write(new_line)

        shuffler('tempds.csv', ods)
        os.remove('tempds.csv')
    print("Dataset cleanup done")


def create_word_dict(source_ds):
    word_dict = []
    with open(source_ds, 'r', 30000, 'latin-1') as ds:
        for line in ds:
            text = line.split('µ')[1]
            words = word_tokenize(text.lower())
            lemm_words = [lemm.lemmatize(w) for w in words]
            word_dict += list(lemm_words)

        word_count = Counter(word_dict)

    cleaned_word_dict = [word for word in word_count if 1000 >
                         word_count[word] > 60]
    dict_size = len(cleaned_word_dict)

    print("Word dictionary size: {}".format(dict_size))
    with open('word_dict.pickle', 'wb') as wd:
        pickle.dump(cleaned_word_dict, wd)

    print("Word dictionary generated and saved")
    return dict_size
```

```
def sentence_to_vector(word_dict_file, cleaned_ds, output_file):

    with open(cleaned_ds, 'r', 30000, 'latin-1') as ds:
        with open(word_dict_file, 'rb') as wd:
            word_dict = pickle.load(wd)
            num_lines = 0
            with open(output_file, 'wb') as hv:

                for line in ds:
                    # print(line)
                    hot_vector = np.zeros(len(word_dict))
                    if line.count('µ') == 1:
                        sentiment, text = line.split('µ')
                        words = word_tokenize(text.lower())
                        lemm_words = [lemm.lemmatize(w) for w in words]
                        for word in lemm_words:
                            if word in word_dict:
                                hot_vector[word_dict.index(word)] += 1
                        hot_vector = list(hot_vector)

                        clean_sentiment = re.search('.*(\d).*', sentiment)

                        if int(clean_sentiment.group(1)) == 0:
                            sentiment = [1, 0]
                        else:
                            sentiment = [0, 1]

                        # print(hot_vector, sentiment)
                        num_lines += 1

                        pickle.dump([hot_vector, sentiment], hv)

                print('Hot vectors file generated with {}
                       lines'.format(num_lines))
    return num_lines



clean_dataset('trainingandtestdata/testdata.manual.2009.06.14.csv', 'test.csv')

with open('data_details.pkl', 'wb') as details:
    dict_size = create_word_dict('small_train.csv')
    train_size = sentence_to_vector('word_dict.pickle', 'small_train.csv',
                                    'train_hot_vectors.pickle')
    test_size = sentence_to_vector('word_dict.pickle', 'test.csv',
                                    'test_hot_vectors.pickle')
    details_sizes = {'dict': dict_size, 'train': train_size, 'test': test_size}
    pickle.dump(details_sizes, details)
```

Model and training

Example A-2 shows the `sentiment_neural_net.py` model and training file.

Example A-2 The sentiment_neural_net.py file

```
import tensorflow as tf
import pickle

x = tf.placeholder('float')
y = tf.placeholder('float')

batch_size = 1000
num_epochs = 1


def load_details():
    with open('data_details.pkl', 'rb') as details:
        det = pickle.load(details)
        return det


line_sizes = load_details()


# Creates the neural network model
def ff_neural_net(input_data):
    neurons_hl1 = 1500
    neurons_hl2 = 1500
    neurons_hl3 = 1500

    output_neurons = 2

    l1_weight = tf.Variable(tf.random_normal([line_sizes['dict'], neurons_hl1]),
                            name='w1')
    l1_bias = tf.Variable(tf.random_normal([neurons_hl1]), name='b1')

    l2_weight = tf.Variable(tf.random_normal([neurons_hl1, neurons_hl2]),
                            name='w2')
    l2_bias = tf.Variable(tf.random_normal([neurons_hl2]), name='b2')

    l3_weight = tf.Variable(tf.random_normal([neurons_hl2, neurons_hl3]),
                            name='w3')
    l3_bias = tf.Variable(tf.random_normal([neurons_hl3]), name='b3')

    output_weight = tf.Variable(tf.random_normal([neurons_hl3, output_neurons]),
                                name='wo')
    output_bias = tf.Variable(tf.random_normal([output_neurons]), name='bo')

    l1 = tf.add(tf.matmul(input_data, l1_weight), l1_bias)
    l1 = tf.nn.relu(l1)

    l2 = tf.add(tf.matmul(l1, l2_weight), l2_bias)
    l2 = tf.nn.relu(l2)
    l3 = tf.add(tf.matmul(l2, l3_weight), l3_bias)
    l3 = tf.nn.relu(l3)

    output = tf.matmul(l3, output_weight) + output_bias

    return output


def training(in_placeholder):
    nn_output = ff_neural_net(in_placeholder)
    saver = tf.train.Saver()
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
                                                logits=nn_output, labels=y))

    optimizer = tf.train.AdamOptimizer(learning_rate=0.001).minimize(cost)

defined graph
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for epoch in range(num_epochs):
            epoch_loss = 0
            buffer_train = []
            buffer_label = []
            with open('train_hot_vectors.pickle', 'rb') as train_hot_vec:
                for i in range(line_sizes['train']):
                    hot_vector_line = pickle.load(train_hot_vec)
                    buffer_train.append(hot_vector_line[0])
                    buffer_label.append(hot_vector_line[1])

                    if len(buffer_train) >= batch_size:
                        _, cost_iter = sess.run([optimizer, cost],
                                        feed_dict={in_placeholder:
                                         buffer_train, y: buffer_label})
                        epoch_loss += cost_iter
                        buffer_train = []
                        buffer_label = []

            print('Epoch {} completed. Total loss: {}'.format(
                                                   epoch+1, epoch_loss))

        correct = tf.equal(tf.argmax(nn_output, 1), tf.argmax(y, 1))
        accuracy = tf.reduce_mean(tf.cast(correct, 'float'))

        with open('test_hot_vectors.pickle', 'rb') as train_hot_vec:
            buffer_test = []
            buffer_test_label = []
            for i in range(line_sizes['test']):
                test_hot_vector_line = pickle.load(train_hot_vec)
                buffer_test.append(test_hot_vector_line[0])
                buffer_test_label.append(test_hot_vector_line[1])

        print('Accuracy using test dataset: {}'
              .format(accuracy.eval({in_placeholder: buffer_test,
                                  y: buffer_test_label})))
        saver.save(sess, "model.ckpt")
```

Using the model

Example A-3 shows the `use_neural_net.py` model.

Example A-3 The use_neural_net.py file

```
import tensorflow as tf
import pickle
import numpy as np
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sentiment_neural_net import ff_neural_net
from sentiment_neural_net import training
lemm = WordNetLemmatizer()

x = tf.placeholder('float')


def get_sentiment(input_data):
    tf.reset_default_graph()
    pl = tf.placeholder('float')
    nn_output = ff_neural_net(pl)
    saver = tf.train.Saver()
    with open('word_dict.pickle', 'rb') as f:
        word_dict = pickle.load(f)

    with tf.Session() as sess:
        # sess.run(tf.global_variables_initializer())
        # saver = tf.train.Saver()
        saver.restore(sess, "model.ckpt")
        words = word_tokenize(input_data.lower())
        lemm_words = [lemm.lemmatize(w) for w in words]
        hot_vector = np.zeros(len(word_dict))

        for word in lemm_words:
            if word.lower() in word_dict:
                index_value = word_dict.index(word.lower())
                hot_vector[index_value] += 1

        hot_vector = np.array(list(hot_vector))

        result = (sess.run(tf.argmax(nn_output.eval(
                                      feed_dict={pl: [hot_vector]}), 1)))
        if result[0] == 0:
            print('Negative:', input_data)
        elif result[0] == 1:
            print('Positive:', input_data)


# Uncomment the row below to train the model
# training(x)
```

```
get_sentiment('Lebron is a beast... nobody in the NBA comes even close')
get_sentiment("This was the best store i've ever seen.")
get_sentiment("Why do you hate the world")
get_sentiment("we always need to do good things to help each other")
```

B

Problem determination tools

This appendix describes Linux tools that are frequently used to gather data from a running system. This appendix also describes troubleshooting practices.

This appendix contains the following topics:

- Logs and configuration data gathering tools
- Troubleshooting pointers for Linux on Power
- Solving a RAID failure

Logs and configuration data gathering tools

Linux, as an open technology, has many tools that help sysadmins gather relevant information from a system before, during, and after a problem occurs. This section covers two of these tools:

- The sosreport tool
- The Scale-out LC System Event Log Collection Tool

The sosreport tool

Sosreport is an extensible and portable data collection tool that is primarily aimed at Linux distributions and other UNIX-like operating systems.

This section describes the installation and use of this tool with Ubuntu. The tool also can be uses with Red Hat Enterprise Linux (RHEL).

The main purpose of this tool is to gather and collect system logs and configuration from a running system. This is a package that is provided as standard in Ubuntu where you can collect logs and configuration information in a batch file.

Installing sosreport

To install the sosreport tool, install the package by running the following command (the system needs connectivity to an Ubuntu package repository):

```
$ sudo apt-get install sosreport
Reading package lists ...
(...) Processing triggers for man-db (2.7.5-1) ...
Setting up sosreport (3.4-1 ~ ubuntu 16.04.1) ...
```

Running sosreport

To run the sosreport tool, run the following command (you need root permission):

```
$ sudo sosreport
sosreport (version 3.4)
(...)
No changes will be made to system configuration.
Press ENTER to continue, or CTRL-C to quit.
Please enter your first initial and last name [comp02]:
Please enter the case id that you are generating this report for []:
(...)
Your sos report has been generated and saved in:
/tmp/sosreport-comp02-20170825144916.tar.xz
The checksum is: 908cbe3c9a8ecf3e2cb916a79666b916.
Please send this file to your support representative.
```

A `.tgz` file is generated under `/tmp`. You can expand it to see the contents or send it for analysis.

The Scale-out LC System Event Log Collection Tool

This tool is a Perl script that is provided by IBM Support that collects logs from a remote or local system through an SSH connection to its baseboard management controller (BMC). The script can be used with an open problem management report (PMR) to provide more information to IBM Support for systems under a support contract with IBM. The tool can be used to gather and centralize logs and configurations from many Power Systems LC systems in a central Linux repository.

Prerequisites

Before you install the tool, you must meet these prerequisites:

- The operating system where the collector tool is run must be Linux.
- The Linux system must have network connectivity to the BMC.
- The Linux system that is used to perform the data collection must have the following tool packages installed:
  - ipmitool
  - perl
  - sshpass

To install these packages, run the following command:

```
$ sudo apt-get install perl sshpass ipmitool
```

Where to get the tool

The tool can be downloaded from Scale-out LC System Event Log Collection Tool.

There is more than one version of the tool, depending on the system’s model and type. You choose the one that applies to you and download the correct `plc.zip` file.

Installation steps

To install the tool, complete the following steps:

1. Copy the `plc.zip` package to a Linux system that has network connectivity to the BMC of the Scale-out LC server from which you need to collect data.
2. Extract the `plc.zip` file into a directory of your choice by running the following command:

   ```
   $ unzip plc.zip

   Archive: plc.zip
   inflating: eSEL 2. p
   inflating: led_status.sh
   inflating: plc.pl inflating:
   README
   ```

   The directory now contains the following files:

   - `plc.pl`
   - `eSEL2.pl`
   - `led_status.sh`
   - `README`

Note: The files that are generated by this script are saved in the same directory from which the script is run.

Usage

This section shows the tool command syntax, its flags, a sample run, and its result:

```
$ plc.pl { -b bmc_address | -i } [-a admin_pw] [-s sysadmin_pw] [-h host -u user
-p password] [-f]
```

Here are the flags:

-b	BMC host name or IP address
-a	BMC ADMIN password if changed from the default (admin)
-s	BMC sysadmin password if changed from the default (superuser)
-i	Interactive mode
-f	Collect BMC firmware image
-h	Linux host address
-u	Linux host user ID
-p	Linux host password

To use the tool, run the following command (in this example, the BMC IP is 10.10.10.10):

```
$ ./plc.pl -b 10.10.10.10 -a admin -f
Getting BMC data
Warning: Permanently added '10.10.10.10' (RSA) to the list of known hosts.
..........................
Getting IPMI Data
........
```

To list the resulting file, run the following command:

```
$ ls -l
10.7.22.1-2017-06-30.1045.powerlc.tar.gz
```

Errors

This section highlights a few errors messages and their exit codes:

- If sshpass is not found on the system, then the `plc.pl` script prints `sshpass is required and not found on this system` and exits with return code 1.
- If the BMC is not reachable by a ping, then the `plc.pl` script prints `Unable to ping bmchostname/IPaddress` and exits with return code 2.
- If a `command not found` error is returned when running the **`plc.pl`** command, try running the command by prefixing the command with **`./`** so that the command is **`./plc.pl`**.

Troubleshooting pointers for Linux on Power

IBM provides a troubleshooting and problem analysis list and techniques for Linux on Power Systems at IBM Knowledge Center.

IBM Knowledge Center is good source for error analysis codes, tools, procedures, and other resources to help you solve problems with Linux on Power, and especially for NVIDIA graphical processing units (GPUs) running in these systems.

Solving a RAID failure

Although a software RAID is not a requirement of IBM PowerAI, it is a preferred practice to mitigate against local disk failure. Creating a RAID device is not mandatory for the installation of IBM PowerAI, but it provides high availability (HA) in case of a failure of one of the disks (solid-state disks (SDDs) or hard disk drives (HDDs)) that are part of the RAID array. The steps that are required to create a two-disk RAID array are described in "Creating a RAID1 array" on page 82.

RAID failure

This section covers the recovery procedure after the occurrence of one failure in one disk that is part of a RAID array.

This test assumes that a failure occurs in `/dev/sda`, where the PRepBoot area is created.

In this example, we completed the following sequence of steps to simulate the disk failure scenario and subsequent recovery. In the case of an actual failure, we would have started from step 3.

1. Confirm the current software RAID configuration.
2. Simulate a failure in one disk (pseudo failure to `/dev/sda`).
3. Remove the disk in `/dev/sda`.
4. Add a disk and create a partition.

Confirming the current software RAID configuration

To confirm the current software RAID configuration, run the following commands to check the state of the software-defined RAID:

```
$ sudo mdadm --detail /dev/md0
   / dev / md 0:
   Version: 1.2
   Creation Time: Mon Mar 6 10: 38: 20 2017
   Raid Level: raid 1
   Array Size: 976622592 (931.38 GiB 1000.06 GB)
   Used Dev Size: 976622592 (931.38 GiB 1000.06 GB)
   Raid Devices: 2
   Total Devices: 2
   Persistence: Superblock is persistent
   Intent Bitmap: Internal
   Update Time: Wed Mar 8 16: 10: 02 2 2017
   State: clean
   Active Devices: 2
   Working Devices: 2
   Failed Devices: 0
   Spare Devices: 0
   Name: ubuntu-disktest: 0 (local to host ubuntu-disktest)
   UUID: 8d57fc2d:c43f9176:8cf093de:44a0db03
   Events: 1696
   Number Major Minor Raid Device State
   0 8 2 0 active sync /dev/sda2
   1 8 18 1 active sync /dev/sdb2

$ sudo cat /proc/mdstat
   Personalities: [raid 1] [linear] [multipath] [raid 0] [raid 6] [raid 5] [raid
   4] [raid 10]
   md 0: active raid 1 sda2 [0] sdb2 [1]
   976622592 blocks super 1.2 [2/2] [UU]
   bitmap: 0/1 pages [0 KB], 65536 KB chunk
   unused devices: <none>
```

Simulating a failure in one disk

To simulate a failure in one disk, complete the following steps.

Note: This exercise was completed on an established and understood system. The actual parameters, options, and device names depend on your actual system. As some of the commands that are listed are of a destructive nature, verify your configuration before performing these changes.

1. Simulate a failure on one disk that is part of the RAID array by running the following command:

```
$ sudo mdadm --fail /dev/md0 /dev/sda2
   mdadm: set /dev/sda2 faulty in /dev/md0
```

- Check the status of the RAID array after the failure by running the following command:

```
$ sudo mdadm --detail / dev / md0
   / dev / md 0:
   Version: 1.2
   Creation Time: Mon Mar 6 10: 38: 20 2017
   Raid Level: raid 1
   Array Size: 976622592 (931.38 GiB 1000.06 GB)
   Used Dev Size: 976622592 (931.38 GiB 1000.06 GB)
   Raid Devices: 2
   Total Devices: 2
   Persistence: Superblock is persistent
   Intent Bitmap: Internal
   Update Time: Wed Mar 8 16: 12: 11 2017
   State: clean, degraded
   Active Devices: 1
   Working Devices: 1
   Failed Devices: 1
   Spare Devices: 0
   Name: ubuntu-disktest: 0 (local to host ubuntu-disktest)
   UUID: 8d57fc2d:c43f9176:8cf093de:44a0db03
   Events: 1700
   Number Major Minor Raid Device State
   0 0 0 0 removed
   1 8 18 1 active sync /dev/sdb2
   0 8 2 - faulty /dev/sda2
```

Removing the disk in /dev/sda

To remove a disk in /dev/sda, complete the following steps:

1. Remove the disk from the RAID definition:
   - Exclude `/dev/sda2` from the software-defined RAID:

     ```
     $ sudo mdadm --remove /dev/md0 /dev/sda2
        mdadm: hot removed /dev/sda2 from /dev/md0
     ```
   - Exclude `/dev/sda` from the system:

     ```
     $ sudo echo 1> /sys/block/sda/device/delete
     ```
   - Confirm that `/dev/sda` is excluded from the system:

     ```
     $ sudo fdisk -l
     ```
2. Pull out the disk physically from the system.

Adding a disk and creating a partition

To add a disk and create a partition, complete the following steps:

1. Install the disk in the system.
2. Recognize the disk in the system:

   ```
   $ sudo echo "- - -" > /sys/class/scsi_host/host0/scan
   ```
3. Confirm that the disk is recognized as `/dev/sda`:

   ```
   $ sudo fdisk -l
   ```
4. Create a partition on the disk:

   ```
   $ sudo fdisk /dev/sda
      Welcome to fdisk (util-linux 2.27.1).
      Changes will remain in memory only until you decide to write them.
      Be careful before using the write command.
      Command (m for help): n
      Partition type
      p primary (0 primary, 0 extended, 4 free)
      e extended (container for logical partitions)
      Select (default p):

      Using default response p.
      Partition number (1-4, default 1):
      First sector (2048-1953525167, default 2048):
      Last sector, + sectors or + size {K,M,G,T,P} (2048-1953525167, default
      1953525167): 16383

      Created a new partition 1 of type 'Linux' and of size 7 MiB.
      Command (m for help): n
      Partition type
      p primary (1 primary, 0 extended, 3 free)
      e extended (container for logical partitions)
      Select (default p):

      Using default response p.

      Partition number (2-4, default 2):
      First sector (16384-1953525167, default 16384):
   ```

```
Last sector, + sectors or + size {K,M,G,T,P} (16384-1953525167, default
1953525167):
Created a new partition 2 of type 'Linux' and of size 931.5 GiB.
Command (m for help): t
Partition number (1, 2, default 2): 1
Partition type (type L to list all types): 41


Changed type of partition 'Linux' to 'PPC PReP Boot'.


Command (m for help): p
Disk /dev/sda: 931.5 GiB, 1000204886016 bytes, 1953525168 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical / physical): 512 bytes / 512 bytes
I / O size (minimum / optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x19 bcc 23 d

Device Boot Start End Sectors Size Id Type
/dev/sda1 2048 16383 14336 7 M 41 PPC PReP Boot
/dev/sda2 16384 1953525 167 195 3 50 8 78 4 931.5 G 83 Linux

Command (m for help): w
The partition table has been altered.
Calling ioctl () to re-read partition table.
Syncing disks.
```

5. Copy the boot area from `/dev/sdb1` to `/dev/sda1`:

```
$ sudo dd if=/dev/sdb1 of=/dev/sda1
   14336 + 0 records in
   14336 + 0 records out
   7340032 bytes (7.3 MB, 7.0 MiB) copied, 0.119115 s, 61.6 MB / s
```

6. Add the disk to the RAID definition:

```
# mdadm -add /dev/md0 /dev/sda2
   mdadm: added /dev/sda2
```

7. Check that the RAID array rebuilding has finished before completion:

   – Sample output when the array is still being rebuilt:

```
$ sudo mdadm --detail /dev/md0
   /dev/md0:
   Version: 1.2
   Creation Time: Mon Mar 6 10: 38: 20 2017
   Raid Level: raid 1
   Array Size: 976622592 (931.38 GiB 1000.06 GB)
   Used Dev Size: 976622592 (931.38 GiB 1000.06 GB)
   Raid Devices: 2
   Total Devices: 2
   Persistence: Superblock is persistent
   Intent Bitmap: Internal
   Update Time: Wed Mar 8 17: 20: 31 2017
   State: clean, degraded, recovering
   Active Devices: 1
   Working Devices: 2
```

```
Failed Devices: 0
Spare Devices: 1
Rebuild Status: 0% complete
Name: ubuntu-disktest: 0 (local to host ubuntu-disktest)
UUID: 8d57fc2d:c43f9176:8cf093de:44a0db03
Events: 1883
Number Major Minor Raid Device State
2 8 2 0 spare rebuilding /dev/sda2
1 8 18 1 active sync /dev/sdb2
```

– Sample output when the array rebuilding has finished:

```
$ sudo mdadm --detail /dev/md0

   /dev/md0:
   Version: 1.2
   Creation Time: Mon Mar 6 10: 38: 20 2017
   Raid Level: raid 1
   Array Size: 976622592 (931.38 GiB 1000.06 GB)
   Used Dev Size: 976622592 (931.38 GiB 1000.06 GB)
   Raid Devices: 2
   Total Devices: 2
   Persistence: Superblock is persistent
   Intent Bitmap: Internal
   Update Time: Thu Mar 9 09: 58: 42 2017
   State: clean
   Active Devices: 2
   Working Devices: 2
   Failed Devices: 0
   Spare Devices: 0
   Name: ubuntu-disktest: 0 (local to host ubuntu-disktest)
   UUID: 8d57fc2d:c43f9176:8cf093de:44a0db03
   Events: 3277
   Number Major Minor Raid Device State
   2 8 2 0 active sync /dev/sda2
   1 8 18 1 active sync /dev/sdb2
   Rebuild
   Rebuild completed
```

Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide more information about the topics in this document. Some publications that are referenced in this list might be available in softcopy only.

- *IBM Power System AC922 Introduction and Technical Overview*, REDP-5472
- *IBM Power System S822LC for High Performance Computing Introduction and Technical Overview*, REDP-5405
- *IBM Power System S822LC Technical Overview and Introduction*, REDP-5283
- *IBM Power Systems L and LC Server Positioning Guide*, REDP-5414

You can search for, view, download, or order these documents and other Redbooks, Redpapers, web docs, drafts, and extra materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- IBM Technology and Consulting Services

  https://www.ibm.com/services/technology-consulting-and-services/
- IBM Lab Services

  https://www.ibm.com/it-infrastructure/services/lab-services
- IBM PowerAI

  https://public.dhe.ibm.com/software/server/POWER/Linux/mldl/ubuntu/
- IBM PowerAI Vision

  https://ibm.co/2AXPZKJ
- IBM Spectrum Conductor Deep Learning Impact V1.1.0

  https://ibm.co/2hOFHJO
- NVIDIA CUDA Toolkit

  https://developer.nvidia.com/cuda-toolkit
- NVIDIA drivers download site

  http://www.nvidia.com

- OpenBLAS

  http://www.openblas.net/
- Ubuntu server installation guide

  https://help.ubuntu.com/lts/serverguide/index.html

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Redbooks

IBM PowerAI: Deep Learning Unleashed on IBM Power Systems

SG24-8409-00

ISBN 0738442941

IBM®

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

SG24-8409-00

ISBN 0738442941

Printed in U.S.A.