O'REILLY®

Compliments of TensorFlow

Considering TensorFlow for the Enterprise

An Overview of the Deep Learning Ecosystem

Sean Murphy & Allen Leis

Considering TensorFlow for the Enterprise

An Overview of the Deep Learning Ecosystem

Sean Murphy and Allen Leis

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Considering TensorFlow for the Enterprise
by Sean Murphy and Allen Leis

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com/safari*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Shannon Cutt
Production Editor: Colleen Cole
Copyeditor: Octal Publishing, Inc.
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

November 2017: First Edition

Revision History for the First Edition
2017-11-01: First Release

978-1-491-99504-4

[LSI]

Table of Contents

Introduction

This report examines the TensorFlow library and its ecosystem from the perspective of the enterprise considering the adoption of deep learning in general and TensorFlow in particular. Many enterprises will not be jumping into deep learning "cold" but will instead consider the technology as an augmentation or replacement of existing data analysis pipelines. What we have found is that the decision to use deep learning sets off a branching chain reaction of additional, compounding decisions. In considering this transition, we highlight these branches and frame the options available, hopefully illuminating the path ahead for those considering the journey. More specifically, we examine the potential rationales for adopting deep learning, examine the various deep learning frameworks that are available, and, finally, take a close look at some aspects of TensorFlow and its growing ecosystem.

Due to the popularity of TensorFlow, there is no shortage of tutorials, reports, overviews, walk-throughs, and even books (such as O'Reilly's own *Learning TensorFlow* or *TensorFlow for Deep Learning*) about the framework. We will not go in-depth on neural network basics, linear algebra, neuron types, deep learning network types, or even how to get up and running with TensorFlow. This report is intended as an overview to facilitate enterprise learning and decision making.

We provide this information from both a high-level viewpoint and also two different enterprise perspectives. One view comes from discussions with key technical personnel at Jet.com, Inc., a large, online shopping platform acquired by Walmart, Inc., in the fall of 2016. Jet.com uses deep learning and TensorFlow to improve a number of

tasks currently completed by other algorithms. The second comes from PingThings, an Industrial Internet of Things (IIoT) startup that brings a time–series-focused data platform, including machine learning and artificial intelligence (AI), to the nation's electric grid from power generation all the way to electricity distribution. Although PingThings is a startup, the company interacts with streaming time–series data from sensors on the transmission and distribution portions of the electric power grid. This requires extensive collaboration with utilities, themselves large, traditional enterprises; thus, PingThings faces information technology concerns and demands commensurate of a larger company.

CHAPTER 1

Choosing to Use Deep Learning

The first questions an enterprise must ask before it adopts this new technology are what is deep learning and why make the change? For the first question, Microsoft Research's Li Deng succinctly answers:[1]

> [d]eep learning refers to a class of machine learning techniques, developed largely since 2006, where many stages of nonlinear information processing in hierarchical architectures are exploited for pattern classification and for feature learning.

The terminology "deep" refers to the number of hidden layers in the network, often larger than some relatively arbitrary number like five or seven.

We will not dwell on this question, because there are many books and articles available on deep learning. However, the second question remains: if existing data science pipelines are already effective and operational, why go through the effort and consume the organizational resources to make this transition?

General Rationale

From a general perspective, there is a strong argument to be made for investing in deep learning. True technological revolutions—those that affect multiple segments of society—do so by fundamen-

1 Li Deng, "Three Classes of Deep Learning Architectures and Their Applications: A Tutorial Survey", *APSIPA Transactions on Signal and Information Processing* (January 2012).

tally changing the cost curve of a particular capability or task. Let's consider the conventional microprocessor as an example. Before computers, performing mathematical calculations (think addition, multiplication, square roots, etc.) was expensive and time consuming for people to do. With the advent of the digital computer, the cost of arithmetic dropped precipitously, plummeting toward zero, and this had two important impacts. First, everything that relied on calculations eventually dropped in cost and became more widely adopted. Second, many of the assumptions that had constrained previous solutions to problems were no longer valid (the key assumption was that doing math is expensive). Numerous opportunities arose to revisit old problems with new approaches previously deemed impossible or financially infeasible. Thus, the proliferation of computers allowed problems to be recast as math problems.

One could argue that this latest wave of "artificial intelligence," represented by deep learning, is another such step change in technology. Instead of forever altering the price of performing calculations, artificial intelligence is irrevocably decreasing the cost of making predictions.[2] As the cost of making predictions decreases and the accuracy of those predictions increases, goods and services based on prediction will decrease in price (and likely improve in quality). Some contemporary services, such as weather forecasts, are obviously based on prediction. Others, such as enterprise logistics and operations will continue to evolve in this direction. Amazon's ability to stock local warehouses with exactly the goods that will be ordered next week by local customers will no longer be the exception but the new normal.

Further, other problems will be recast as predictions. Take for example the very unconstrained problem of autonomously driving a car. The number of situations that the software would need to consider driving on the average road is nearly infinite and could never be explicitly enumerated in software. However, if the problem is recast as predicting what a human driver would do in a particular situation, the challenge becomes more tractable. Given the extent that the enterprise is run on forecasts, deep learning will become an enabler for the next generation of successful companies regardless of

2 Ajay Agrawal, Joshua S. Gans, and Avi Goldfarb, "What to Expect from Artificial Intelligence," *MIT Sloan Management Review Magazine* (Spring 2017).

whether the actual capability resides within or outside of the organization.

Specific Incentives

Adopting deep learning can provide significant advantages. The immense popularity of deep learning and AI is backed by impressive and repeatable results. Deep learning is a subset of machine learning, which can be considered a subset of AI (Figure 1-1).

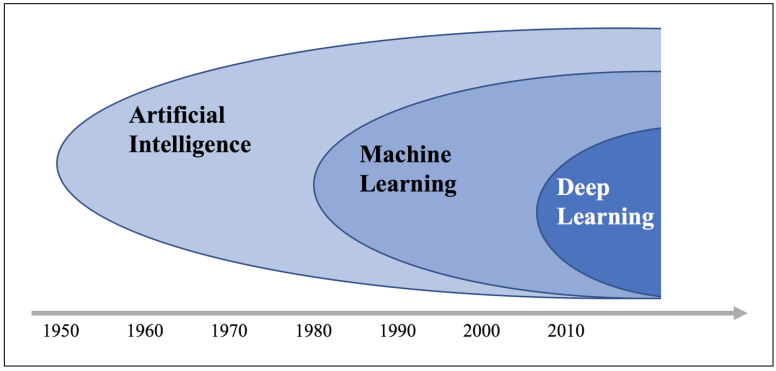

Figure 1-1. A Venn diagram stretching over time showing the relationship between, AI, machine learning, and deep learning

Deep learning–based solutions have not only exceeded other machine learning techniques in performance for certain tasks, they have even reached parity with people and, in some cases, surpassed human-level capability. As an example, let's examine the implications of improvements in the performance of automated machine translation (translating text from one language to another). Early on, this type of software was a novelty, potentially useful in a small number of limited cases. As the translation accuracy improved, the economics of translation began to change, and the number of cases amenable to automated translation increased. As translation accuracy approaches that of a human translator, the potential economic impact is far greater. It might be possible for a single human translator to review quickly the output of software translation, increasing translation output. At this point, it might be possible to reduce the number of translators needed for a given translation load. Eventually, as the software-based approach exceeds the performance of human translation, the human translators are entirely replaced with software that can run on demand 24 hours per day, seven days per

week. Note that as of late 2016, the Google Translate service moved entirely to Google's Neural Machine Translation system, a deep Long short-term memory (LSTM) network.[3]

Let's look at some additional examples of what is now possible with deep learning to begin considering the potential impact that this technology could have on the enterprise.

Using Sequence Data

Audio and text are both examples of sequence data, a type in which the relationship between adjacent letters and words or audio segments is stronger the closer they occur. Free form or unstructured text tends to be a challenging data type to handle with traditional algorithmic approaches. Other examples of sequence data include sensor streams captured as a time–series of floating-point values. If you have a traditional engineering or hard-sciences background, think of sequence data as a one-dimensional signal or time series.

Automated speech recognition

Gaussian Mixture Models (GMM) had been the state of the art in transcribing speech into text until deep neural networks, and then recurrent neural networks, stole the performance crown over approximately the past five years. Anyone who has used the Google Assistant on Android phones has experienced the capabilities of this technology first hand, and the business implications are vast.

Using Images and Video

Images are data sources for which two-dimensional spatial relationships are important. It is inherently assumed that points in an image nearer to one another have a stronger relationship than points further apart. Video data can be considered a sequence of images and the spatial relationship holds within each frame and, often, across subsequent frames.

3 Yonghui Wu et al., "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", technical report (October 2016).

Image classification

Image classification is a classic problem within computer science and computer vision. To classify an input image, the algorithm assigns a label to the image from a finite set of predefined categories. Note that image classification assumes that there is one object within each image.

Classifying images has always been considered a very challenging problem within computer science, so much so that competitions for visual recognition are held every year. In the ImageNet Large Scale Visual Recognition Challenge LSVRC-2010, a deep convolutional neural network from the University of Toronto with 60 million parameters and 650,000 neurons won.[4] In the 2012 competition, a variation of the same network won with an error rate of 15.3%—significantly beyond the 26.2% error rate of the second-place competitor. This type of leap in performance is exceedingly rare and helped to establish the convolutional neural network as the dominant approach.

Automated game playing

Google's DeepMind team showed that a neural network could learn to play seven different video games from the very old Atari 2600 game console, performing better than all previous algorithmic efforts and outperforming even human experts on three of the games. This convolutional neural network was trained using a direct feed from the console (basically, the same thing that a person playing the game would see) and trained with a variation of reinforcement learning called Q-learning.[5] Training artificial intelligence with video feeds to perform goal-oriented, complex tasks has significant implications for the enterprise.

Automatic black-and-white image/movie colorization

Convolutional neural networks have been used to colorize black-and-white photos, a traditionally time-intensive process done by

4 A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", *Advances in Neural Information Processing Systems* 25 (2012).

5 V. Mnih, K. Kavukcuoglu, D. Silver et al, "Playing Atari with Deep Reinforcement Learning", NIPS Deep Learning Workshop (2013).

specialists. Researchers at the University of California, Berkeley have "attack[ed] the problem of hallucinating a plausible color version of the photograph" with both a fully automated system and a human-assisted version.[6] Although most enterprises are not colorizing old movies, this research path helps to demonstrate not only how deep learning–based techniques can automate tasks previously requiring creative expertise, but also that AI and humans can collaborate to accelerate traditionally time-intensive tasks.

Mimicking Picasso

Generative Adversarial Networks (GANs) made quite a splash on the world of deep learning and are based on the idea of having two models compete to improve the whole. In the words of the original paper:[7]

> The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles.

GANs have been used to "learn" the style of famous painters like Picasso and then transform photos into representations mimicking that painter's style.

Specific Enterprise Examples

To get a better understanding of how deep learning is used, let's examine a couple of examples from industry.

Jet.com

Jet.com provides an excellent and potentially unexpected example of how this technology and these new capabilities translate to benefits for the enterprise. Jet.com offers millions of products for sale, many of which are provided by third-party partners. Each new product must be placed into one of thousands of categories. In the past, this

6 R. Zhang, P. Isola, and A. Efros. "Colorful Image Colorization", in ECCV 2016 (oral), October 2016.

7 I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair et al., "Generative Adversarial Nets", *Advances in Neural Information Processing Systems* 27 (2014).

categorization was done based on the text-based descriptions and information provided by the partner. Sometimes, this information is incorrect or inaccurate or inconsistent. However, using TensorFlow and deep learning, Jet.com integrated photos of the product into the categorization pipeline, significantly boosting the accuracy of the classifications. Thus, deep learning allowed a new data type (images) to be quickly integrated into a classic data analysis pipeline to improve business operations.

Another problem that Jet.com has addressed with deep learning is processing text input in the search box. As an example, if a customer types "low profile television stand" into the search box, the system must accurately parse this natural language to locate the item intended by the customer.

PingThings

Most "smart grid"–oriented startups are focused only on residential or commercial smart meters that capture data describing the very end of the electric grid. PingThings ingests, stores, and analyzes in real time data streaming from sensors attached to high-value utility assets from all three segments of the grid: generation, transmission, and distribution. These sensors, of which there are hundreds or thousands within each utility, typically record measurements 30 times per second. The use cases for such high-frequency data are numerous, from the simple prediction of the expected next reading from each sensor to the detection of events of interest to control room operators or the prediction of asset failure and higher-level system state.

Traditionally, time series analysis work has been focused on feature engineering—extracting the right characteristics of the time series under examination. To understand the complexity of such efforts, let's examine one anomaly detection system developed by the Pacific Northwest National Laboratory. This algorithm divides the time series into windows of data (a very common approach) and then fits a quadratic curve to each window, extracting parameters describing the curve and how well it approximates the signal. This simple signature for multiple windows is summarized with statistical descriptors that are then compressed to remove any repetitive information. Finally, an "atypicality" score is computed for these compressed values. With deep learning, the feature engineering step can be left to the neural network, simplifying a great deal of the effort and analy-

sis. Further, dedicated hardware can be used for inference that promises substantial reduction in execution time for such tasks as anomaly detection.

Potential Downsides

Every new technology has its drawbacks and deep learning, especially from the enterprise perspective, is no different. The first and foremost is common for every new technology but especially so when dealing with a technology emerging from a small number of universities. Numerous reports have indicated that there is a drastic shortage of data scientists and that shortage is even more extreme when dealing with deep learning specialists. Further, most of the major technology companies—Google, Facebook, Microsoft, Amazon, Twitter, and more—are fighting for this scarce talent, driving up both the cost of finding personnel and resulting salaries.

From a technology standpoint, deep learning–based approaches have several potential downsides. First, it is almost always best to solve a problem with the simplest technology possible. If linear regression works, use it in lieu of a much more complicated deep learning methodology. Second, when needed, deep learning requires significant resources. Training neural networks requires not only large datasets (which should be nothing new for the enterprise well versed in machine learning) but also larger computational muscle to train the networks. This means either multiple graphics processing units (GPUs) or access and willingness to use cloud resources with GPUs. If the desired use case for the enterprise falls well within the confines of what has been done before with deep learning, the payoff could be large. However, depending on how far outside the box the need is, there are no guarantees that the project or product will be a success.

Finally, the broader field is called data science for a reason. Although there are some use cases relevant to the enterprise that are directly supported by prebuilt networks (image classification for example), many enterprises will be interested in extending these ideas and developing adjacent capabilities with deep learning networks. This should be considered more of a research and development effort as opposed to an engineering initiative with the concomitant risk.

Summary

Deep learning represents the state of the art in machine learning for numerous tasks involving many different data modalities, including text, images, video, and sound, or any data that has structurally or spatially constructed features that can be exploited. Further, deep learning has somewhat replaced the significant issues of feature extraction and feature engineering with the selection of the appropriate neural network architecture. As deep learning continues to evolve, a better understanding of the capabilities and performance attributes of networks and network components will arise. The use of deep learning will transition to more of an engineering problem addressing the following question: how do we assemble the needed neuron types, network layers, and entire networks into a system capable of handling the business challenge at hand.

Ultimately, the question will not be whether enterprises will use deep learning, but how involved each organization becomes with the technology. Many companies might already use services built with deep learning. For companies building products and services that could directly benefit from deep learning, the operative question is "do we buy" or "do we build?" If a high-level API exists that provides the necessary functionality meeting performance requirements, an organization should use it. However, if this is not possible, the organization must develop the core competency in house. If the latter path is chosen, the next question is, can we simply re-create the work of others with minor tweaks or do we need to invent completely new systems, architectures, layers, or neuron-types to advance the state of the art? The answer to this question dictates the staffing that must be sourced.

CHAPTER 2

Selecting a Deep Learning Framework

When the decision is made to adopt deep learning, the first question that arises is which deep learning library should you choose (and why)? Deep learning has become a crucial differentiator for many large technology firms and each has either developed or is championing a particular option. Google has TensorFlow. Microsoft has the Microsoft Cognitive Toolkit (aka CNTK). Amazon is supporting the academia-built MXNet, causing some to question the longevity of the internally developed DSSTNE (Deep Scalable Sparse Tensor Network Engineer). Baidu has the PArallel Distributed Deep LEarning (PADDLE) library. Facebook has Torch and PyTorch. Intel has BigDL. The list goes on and more options will inevitably appear.

We can evaluate the various deep learning libraries on a large number of characteristics: performance, supported neural network types, ease of use, supported programming languages, the author, supporting industry players, and so on. To be a contender at this point, each library should offer support for the use of graphics processing units (GPUs)—preferably multiple GPUs—and distributed compute clusters. Table 2-1 summarizes a dozen of the top, open source deep learning libraries available.

Table 2-1. General information and GitHub statistics for the 12 selected deep learning frameworks (the "best" value in each applicable column is highlighted in bold)

	General information				GitHub statistics			
	Org	Year	License	Current version	Time since last commit	Watches	Commits	Contributors
Caffe (GitHub)	UC Berkeley	2014	BSD 2-Clause	1.0	21 days	2037	4045	248
Caffe2 (GitHub)	Facebook	2017	BSD 2-Clause	0.8.0	**1 hour**	437	2406	113
BigDL (GitHub)	Intel	2017	Apache 2.0	0.2.0	13 hours	179	1752	37
Deeplearning4J (GitHub)	Skymind	2014	Apache 2.0	0.9.2	11 hours	712	8621	124
DyNet (GitHub)	Carnegie Mellon	2015	Apache 2.0	2.0	8 hours	160	2769	79
DSSTNE (GitHub)	Amazon	2016	Apache 2.0		255 days	345	221	22
Microsoft Cognitive Toolkit (CNTK) (GitHub)	Microsoft Research	2015	MIT	2.1	3 hours	1235	14791	145
MXNet (GitHub)	Amazon	2015	Apache 2.0	0.1	**1 hour**	987	5820	405
PADDLE (GitHub)	Baidu	2016	Apache 2.0	0.10.0	2 hours	492	6324	72
TensorFlow (GitHub)	Google	2015	Apache 2.0	1.3	8 hours	**6087**	21600	**1028**
Theano (GitHub)	Université de Montréal	2008	BSD License	0.9	1 day	552	27421	321
Torch7 (GitHub)	Several	2002	BSD License	7.0	2 days	680	1331	134

Supported programming languages

Nearly every framework listed was implemented in C++ (and potentially use Nvidia's CUDA for GPU acceleration) with the exception of Torch, which has a backend written in Lua, and Deeplearning4J, which has a backend written for the Java Virtual Machine (JVM). However, the important issue when using

these frameworks is which programming languages are supported for training—the compute-intensive task of allowing the neural network to learn from data and update internal weights—and which languages are supported to inference—showing the previously trained network new data and reading out predictions. As inference is a much more common task for production, one could argue that the more languages a library supports for inference, the easier it will be to plug in to existing enterprise infrastructures. Training is somewhat more specialized, so the language support might be more limited. Ideally, a framework would support the same set of languages for both tasks.

Different types of networks
: There are many different types of neural networks, and researchers in academia and industry are developing new network types with corresponding new acronyms almost daily. To name just a few, there are feed forward networks, fully connected networks, convolutional neural networks (CNNs), restricted Boltzman machines (RBMs), deep belief networks (DBNs), denoising autoencoders, stacked denoising autoencoders, generative adversarial network (GANs), recurrent neural networks (RNNs), recursive neural networks, and many more. If you would like graphical representations of the above or an even longer list of different neural network types/architectures, the Neural Network Zoo is a good place to start.

  Two network types that have received significant press are convolutional neural networks that can handle images as inputs, and recurrent neural networks and variations, such as LSTM, that can handle sequences—think text in sentence, time–series data, audio streams, and so on—as input. The deep learning library that you choose should support the broadest range of networks and, at the very least, those most relevant to business needs.

Deployment and operationalization options
: Although both machine learning and deep learning often require a significant amount of data for training, deep learning truly heralded the transition from big data to big compute. For the enterprise, this is likely the largest issue and potential obstacle transitioning from more traditional machine learning techniques to deep learning. Training large-scale neural networks can take weeks or even months; thus, even a 50% performance

gain can offer enormous benefits. To make the process feasible, training networks requires significant raw computing power that often comes in the form of one or more GPUs or even more specialized processors. Ideally, a framework would support both single and multi- CPU and GPU environments and heterogeneous combinations.

Accessibility of help
: The degree to which help is available is a very important component to the usefulness and success of a library. The volume of documentation is a strong indicator of the success (and potential longevity) of a platform and the adoption and use of the library easier. As the ecosystem grows, so too should the documentation in numerous forms including online tutorials, electronic and in-print books, videos, online and offline courses, and even conferences. Of particular note to the enterprise is the issue of commercial support. Although all of the aforementioned libraries are open source, only one offers direct commercial support: Deeplearning4J. It is highly likely that third parties will be more than eager to offer consulting services to support the use of each library.

Enterprise-Ready Deep Learning

Down selecting from the dozen deep learning frameworks, we examine four of the libraries in depth due to their potential enterprise readiness: TensorFlow, MXNet, Microsoft Cognitive Toolkit, and Deeplearning4J. To give an approximate estimate of popularity, Figure 2-1 presents the relative worldwide interest by search term as measured by Google search volume.

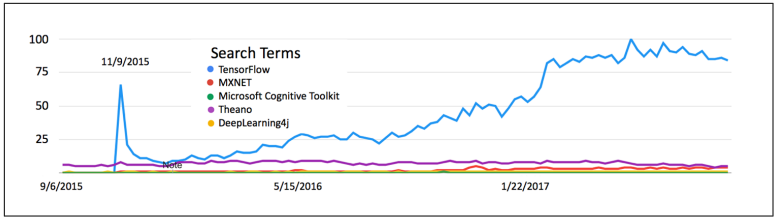

Figure 2-1. The relative, worldwide search "interest over time" for several of the deep learning open source frameworks—the maximum relative value (100) occurred during the week of May 14th, 2017

TensorFlow

Google has a rich and storied history in handling data at scale and applying machine learning and deep learning to create useful services for both consumers and enterprises. When Google open sources software, the industry takes notice, especially when it is version two. In 2011, Google internally used a system called DistBelief for deep learning that was capable of using "large-scale clusters of machines to distribute training and inference in deep networks."[1] The lessons learned from years of operating this platform ultimately guided the development of TensorFlow, announced in November of 2015.[2]

> TensorFlow [1] is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms. A computation expressed using TensorFlow can be executed with little or no change on a wide variety of heterogeneous systems, ranging from mobile devices such as phones and tablets up to large-scale distributed systems of hundreds of machines and thousands of computational devices such as GPU cards. The system is flexible and can be used to express a wide variety of algorithms, including training and inference algorithms for deep neural network models, and it has been used for conducting research and for deploying machine learning systems into production across more than a dozen areas of computer science and other fields, including speech recognition, computer vision, robotics, information retrieval, natural language processing, geographic information extraction, and computational drug discovery.

Some who believe that this is a winner-take-all space would say that TensorFlow has already won the war for developer mindshare. Although that pronouncement is likely premature, TensorFlow does currently have impressive momentum. By nearly all metrics, TensorFlow is the most active open source project in the deep learning space. It also has the most books written about it, has an official conference, has generated the most worldwide interest as measured by Google search volume, and has the most associated meetups. This type of lead will be difficult to overcome for its competitors.

---

1 J. Dean et al., "Large Scale Distributed Deep Networks," *Advances in Neural Information Processing Systems* 25 (2012).

2 M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems", preliminary white paper (2015).

MXNet

MXNet is the youngest deep learning framework that we will examine more closely. It entered Apache incubation in January 2017 and the latest version as of October 2017 was the 0.11.0 release. The question is, given its youth and the credentials of competitors, should enterprises even be aware of this alternative deep learning framework? The very loud answer to that question came from Amazon, which announced in November 2016 that "Apache MXNet is Amazon Web Services' deep learning framework of choice". It is probably no coincidence that one of the founding institutions behind MXNet was Carnegie Mellon University and Dr. Alexander Smola, a professor in the CMU machine learning department, joined Amazon in July 2017.

To further make the case that MXNet is a contender, the latest release candidate of the framework allows developers to convert MXNet deep learning models to Apple's Core machine learning format, meaning that billions of iOS devices can now provide inference capability to applications using MXNet. Also note that Apple is the one large tech company not associated with any of the aforementioned deep learning frameworks.

The next question is, what is MXNet and how does it improve upon existing libraries?[3]

> MXNet is a multilanguage machine learning library to ease the development of machine learning algorithms, especially for deep neural networks. Embedded in the host language, it blends declarative symbolic expression with imperative tensor computation. It offers auto differentiation to derive gradients. MXNet is computation and memory efficient and runs on various heterogeneous systems, ranging from mobile devices to distributed GPU clusters.

MXNet arose out of a collaboration between a number of top universities including CMU, MIT, Stanford, NYU, the University of Washington, and the University of Alberta. Given its more recent development, the authors had an opportunity to learn from the deep learning frameworks that have come before and potentially improve upon them. The framework strives to provide both flexibility and performance. Developers can mix symbolic and imperative

3 T. Chen et al., "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems", NIPS Machine Learning Systems Workshop (2016).

programming models, and both can be parallelized by the dynamic dependency scheduler. Developers can also take advantage of the predefined neural network layers to construct complex networks with little code. Importantly, MXNet goes far beyond supporting Python; it also has full APIs for Scala, R, Julia, C++, and even Perl. Finally, the MXNet codebase is small and was designed for efficient scaling over both GPUs and CPUs.

Microsoft Cognitive Toolkit (CNTK)

Despite the rise of the web, Microsoft is still one of the dominant vendors in the enterprise space. Thus, it should come as no surprise that the Microsoft Research deep learning framework is one to examine. Formerly known as the Computational Neural Toolkit (CNTK), the toolkit apparently emerged from the world class speech transcription team at Microsoft Research and was then generalized for additional problem sets. The first general paper emerged in 2014 and the software appeared on Github in January of 2016.[4] It was used in 2016 to achieve human-level performance in conversational speech recognition. The toolkit promises efficient scalability and impressive performance in comparison to competitors.

Deeplearning4J

Deeplearning4J is somewhat of the odd framework in this list. Even though Python has become the nearly de facto language for deep learning, Deeplearning4J was developed in Java and designed to use the JVM and be compatible with JVM-based languages such as Scala, Clojure, Groovy, Kotlin, and JRuby. Note that the underlying calculations are coded in C/C++ and CUDA. This also means that Deeplearning4J works with both Hadoop and Spark out of the box. Second, many of the earlier deep learning frameworks arose out of academia and the second wave rose out of larger technology companies. Deeplearning4J is different because it was created by a smaller technology startup (Skymind) based in San Francisco and started in 2014. Although Deeplearning4J is open source, there is a company that is willing to provide paid support for customers using the framework.

4 A. Agarwal et al. (2014). "An Introduction to Computational Networks and the Computational Network Toolkit", Microsoft Technical Report MSR-TR-2014-112 (2014).

Industry Perspectives

Jet.com provides an interesting example of deep learning library selection. Jet.com is a Microsoft shop, from top to bottom, and is one of the few remaining shops focused on the F# programming language in the United States (they also use C# and the .NET framework). For cloud services, the company uses Microsoft Azure. Despite the strong focus on Microsoft, Jet.com uses TensorFlow for deep learning. The company had originally started with Theano but transitioned to TensorFlow quickly after it was released. TensorFlow is run on virtual instances with GPUs running within the Microsoft Azure cloud.

As a technology startup, PingThings has substantial leeway to select different technologies. Google's prominence in the field and the volume of documentation and tutorials were both strong motivators for choosing TensorFlow. However, PingThings is working on projects funded by the National Science Foundation (NSF) and the Advanced Research Projects Agency-Energy (ARPA-E) with collaborators from multiple research institutes while also deploying hardware inside of utilities. Thus, the fact that Google designed TensorFlow to balance the needs of both research and robust operation at scale was particularly important. Tensor2Tensor (a part of the ecosystem that we discuss later) was particularly appealing because of its focus on sequence data. MXNet is an interesting new option whose future development will be watched, especially given its strong performance and support from Amazon.

Summary

TensorFlow does everything well enough: competitive performance, strong support for different neural network types, numerous hardware deployment options, multi-GPU support, numerous programming language options, and more. However, the library's allure transcends this feature set.

Google played a large part in helping to start the big data revolution with the combination of a distributed file system and the MapReduce computing framework, and continues to lead the industry

today.[5] [6] Further, there are numerous successful examples of technologies iterating within Google and then seeing a widespread release. Kubernetes, the popular container orchestration system, is one such example, being the result of years of experience and lessons learned from earlier internal systems like Borg and Omega.[7] Google is well regarded for both advancing the state of the art and software engineering at web scale, a balance between academia and industry that seems particularly appropriate for the gold rush that is deep learning.

TensorFlow inherits this goodwill with an aim to be sufficiently flexible for intense research while also robust enough to allow production deployment of its models. Newer frameworks might arise that could build upon the lessons learned from TensorFlow, improving various aspects of the library, making available multiple programming methodologies, or offering a more improved performance. Many of the libraries described above attempt to do one or more of these things. However, TensorFlow is constantly applying these lessons, striving for better performance and exploring new approaches, as well. As long as Google's weight and effort remain behind TensorFlow, it will continue to be a strong, safe, and practically the default choice for deep learning libraries, especially given the ecosystem that we describe in Chapter 3.

5 S. Ghemawat, H. Gobioff, and S. Leung, "The Google File System," *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (2003).

6 J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation* 6 (2004): 10.

7 B. Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes, "Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade," *System Evolution* 14, no. 1 (2016).

CHAPTER 3
Exploring the Library and the Ecosystem

TensorFlow itself, while impressive, is "just" an open source library for numerical computation using data flow graphs. As described in Chapter 2, there are plenty of open source competitors that you could use to build, train, and run inference with complex neural networks; more will arise. It is the ecosystem surrounding a library, built not only by the original author(s) but also by the community, that forms a long-term solution in an ever-evolving space. It is TensorFlow's rich and growing ecosystem that compels many to use it. To go through the detailed use of each TensorFlow component would be beyond the scope of this report, but we will strive to introduce the relevant pieces and provide some perspective on the larger overall puzzle.

Python serves as an excellent example of the power of the ecosystem. One of the language's selling points was its "batteries-included" philosophy; it came with a standard library that made many tasks (such as making HTTP requests) simple. Even with this approach, Python owes some of its success to its ecosystem. The Numpy and Scipy libraries created a strong foundation for numerical and scientific computing, extending the language's core capabilities and the community that uses it. Libraries such as scikit-learn, which serves almost as a reference implementation for algorithms within the field of machine learning, and Pandas, the de facto standard for Python-based data analysis, have built upon Numpy and Scipy and have helped Python contest for the throne of data science programming

languages. Companies like Enthought and then Continuum Analytics created distributions of Python that included critical libraries whose numerous external dependencies had made installation difficult. This simplified the deployment of Python, broadening the community of users. The IPython Notebook has evolved into Project Jupyter (*Ju*lia *Pyt*hon *R*) to support new languages beyond Python. Jupyter is emerging as the standard IDE for data science, deep learning, and artificial intelligence. Even the deep learning libraries based on Python not only extend Python's ecosystem but also are only possible because of that ecosystem.

We divide the TensorFlow ecosystem into several functional categories. The first group increases the direct utility of the library by making it easier for you to design, build, and train neural networks with or on top of TensorFlow. Several examples of this are prebuilt and even pretrained deep neural networks, graphical interfaces for tracking training progress (TensorBoard), and a higher-level interface to TensorFlow (Keras). The second category contain tools that make inference possible and easier to manage. The next category are the components used to connect to and interact with other popular open source projects such as Hadoop, Spark, Docker, and Kubernetes. The last category are technologies that decrease the time and cost to train deep neural networks because this is often the rate-limiting step.

This division loosely follows the three stages of the TensorFlow pipeline: (1) data preparation, (2) training, and (3) inference and model serving. We will not focus significant prose on preparing data for use with TensorFlow; it is assumed that enterprises transitioning from other types of machine learning will already have mechanisms in place to clean, wrangle, and otherwise prepare data for analysis and training. Figure 3-1 shows how the ecosystem lines up with the pipeline.

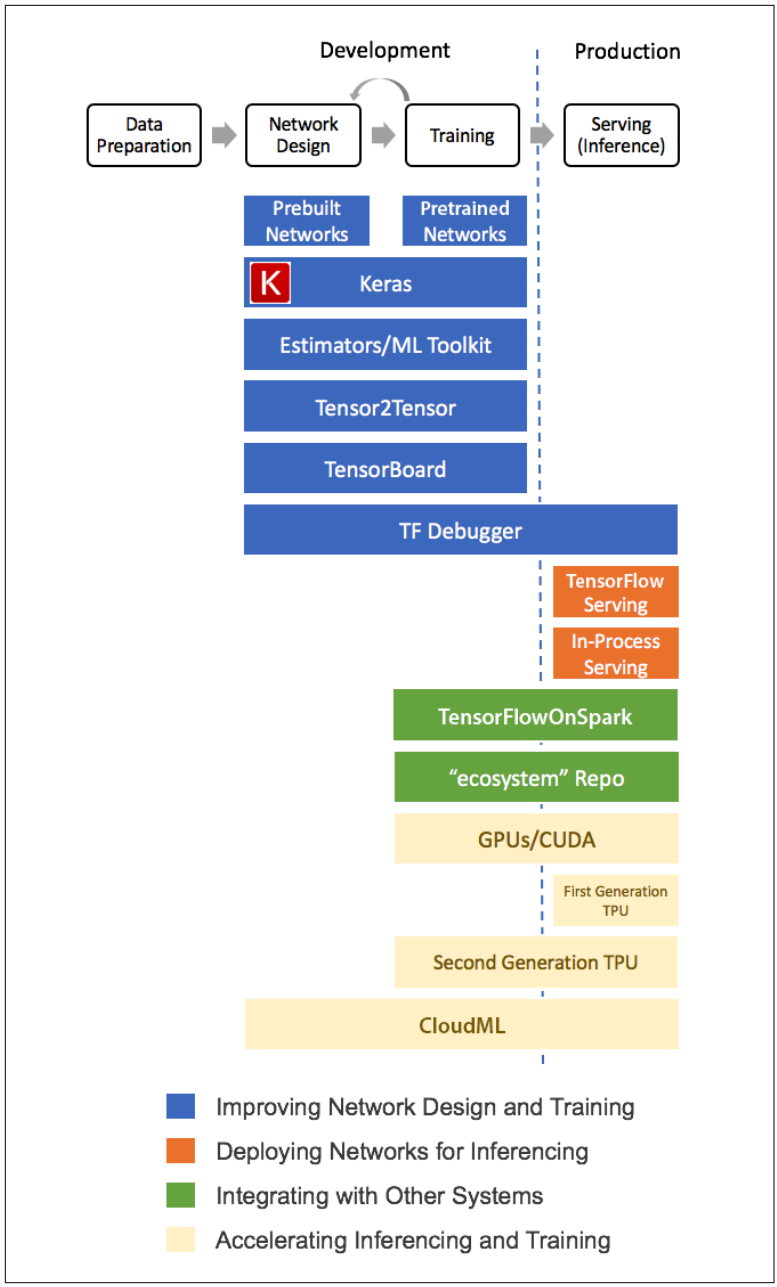

Figure 3-1. The alignment between various parts of the TensorFlow ecosystem and the overall workflow

Improving Network Design and Training

The following tools and open source projects help the software engineer and data scientist to design, build, and train deep learning models, seeking to create immediate value for the TensorFlow user. If you're new, we recommend that you take a look at the relevant prebuilt neural networks as a starting point and then take a close look at Keras, which can simplify the creation of more complex networks and offers some portability for models. If your application involves sequence data (text, audio, time–series, etc.), do not skip Tensor2Tensor. Regardless of your experience level, expect to use TensorBoard.

Estimators

TensorFlow offers a higher-level API for machine learning (tf.estimator). It contains a number of built in models—linear classifier, linear regressor, neural network classifier, neural network regressor, and combined models—and allows more rapid configuration, training, and inference or evaluation to occur.

Prebuilt Neural Networks

Deep neural network design remains somewhat of an academic pursuit and an artform. To speed the adoption and use of DL, TensorFlow comes with a number of example neural networks available for immediate use. Before starting any project, check this directory to see if a potential jumpstart is available. Of special note is the Inception network, a convolutional neural network that achieved state-of-the-art performance in both classification and detection in the 2014 ImageNet Large-Scale Visual Recognition Challenge.[1]

Keras

Keras is a high-level API written in Python and designed for humans to build and experiment with complex neural networks in the shortest amount of time possible. You can use Keras as a model definition abstraction layer for TensorFlow, and it's also compatible

1 Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, "Going Deeper with Convolutions", *Computer Vision and Pattern Recognition* (2015).

with other TF-related tools. Interestingly, Keras offers a potential portability pathway to move networks from one deep learning library to another, coming closest to achieving a standard model abstraction. It is currently capable of running on top of not only TensorFlow, but also Theano, Microsoft Cognitive Toolkit, and, recently, MXNet. Further, Keras is the Python API for Deeplearning4J.

Machine Learning Toolkit for TensorFlow

This toolkit provides out-of-the-box, high-level machine learning algorithms (see the list that follows) inspired by the popular scikit-learn library for immediate use versus rewriting the algorithms using TF's lower-level API.

- Neural networks (DNN, RNN, LSTM, etc.)
- Linear and logistic regression
- K-means clustering
- Gaussian mixture models
- WALS matrix factorization
- Support vector machine (with L1 and L2 regularization)
- Stochastic dual coordinate ascent for context optimization
- Random forests
- Decision trees

Importantly, all of these algorithms have distributed implementations and can execute across machines in parallel, offering significant performance increases over nonparallelized implementations.

Tensor2Tensor (T2T)

T2T is an open source system built on top of TensorFlow to support the development and training of state-of-the-art deep learning networks with a particular focus on sequence-to-sequence models (the kinds used to translate text or provide a caption for images). This library, released in 2017, is being actively used, developed, maintained, and supported by the Google Brain team. It also includes the following:

- Many datasets of different types of data (text, audio, and images)
- Hyperparameter configurations
- The best models from a number of recent academic papers, including these:
    - "Attention Is All You Need"
    - "Depthwise Separable Convolutions for Neural Machine Translation"
    - "One Model to Learn Them All"

The goal of the software is to provide a level of abstraction higher than that provided by the base TensorFlow API and encapsulate many best practices and hard-learned "tricks" of the trade into software that enforces a standardized interface between all of its pieces.

TensorBoard

Even though machine learning in general is difficult to visualize, neural networks have long been criticized for being a black box, affording almost no transparency into their inner workings. Deep network graphs can be difficult to visualize. Dataflow within the many layers of a graph is difficult to observe *in situ* or even *a posteriori*. Understanding and debugging neural networks can be notoriously difficult from a practitioner's perspective.

TensorBoard is a collection of visualization tools that provides insight into a TensorFlow graph and allows the developer or analyst to understand, debug, and optimize the network. The UI for the tools is browser based. It provides three core capabilities:

Visualization of the graph structure
: The first step to understanding a neural network, which could be composed of dozens of layers, with hundreds of thousands of nodes or more, is to inspect and verify the structure of the network visually.

Visualization of summaries
: TensorBoard allows you to attach summarizations to capture various types of tensors flowing through the graph during training and execution. These tensors could represent input data or

network weights, with histograms demonstrating how network weights or other tensors in the network change over time.

Embedding visualizer
: TensorBoard also allows you to visualize the machine learning results in a three-dimensional interface.

Typical functions are available such as graphing summary statistics during learning. You can also gain insight into the outputs of specific layers to run your own analysis. This makes it possible to review the distribution of outputs from one layer before those values serve as input to the next layer. TensorBoard reads serialized TensorFlow event data. Although some features and visualizations come for free without setup, others require code changes to capture the data to be visualized, and you can choose the nodes or objects about which you collect summary information.

Google has big goals for continued TensorBoard development. First, the TensorFlow debugger will be integrated with TensorBoard so that you can visualize debugging information through this tool. Next, TensorBoard will soon support plug-ins that allow for complex and custom visualizations designed for interrogating specific neural networks types with unique visualization needs in various problem domains. Finally, Google plans to release an "organizational scale" TensorBoard designed not just for the individual but also for the team so that results can be rapidly disseminated and a shared history of development can be kept.

TensorFlow Debugger

TensorFlow comes out of the box with a specialized debugger (tfdbg) that allows for introspection *of any data as it flows through* TensorFlow graphs while doing both training and inference. There was an interesting third-party open source debugger for TensorFlow (tdb) with robust visualization capabilities. It was described by the author as, "TDB is to TensorBoard as GDB is to printf. Both are useful in different contexts." However, the author, Eric Jang, was apparently hired by Google Brain and the external effort has been abandoned.

Deploying Networks for Inference

Deep learning training often gets most of the press due to the large computational demands that it requires. However, a state-of-the-art deep neural network is without value if no one uses it. Providing inference capabilities in a robust, scalable, and efficient way is critical for the success of the deep learning library and ecosystem.

TensorFlow Serving

After training, the enterprise faces the decision of how to operationalize deep learning networks and machine learning models. There will be some use cases, such as in research, experimentation, or asynchronous prediction/classification activities, for which operationalization is not required. However, in many instances, the enterprise will want to provide real-time inference for user-facing applications (like object detection in a mobile application such as the "Not Hotdog" application from HBO's *Silicon Valley*), human decision-making support, or automated command and control systems; this requires moving a previously trained network into production for inference.

Operationalizing machine learning models opens a Pandora's box of problems in terms of designing a production system. How can we provide the highest level of performance? What if we need to expose multiple models for different operations? How do we manage a deployment process or deal with the configuration management of multiple model versions?

TensorFlow Serving provides a production-oriented and high-performance system to address this issue of model deployment; it hosts TensorFlow models and allows remote access to them to meet client requests. Importantly, the models served are versionable, making it easy to update networks with new weights or iterations while maintaining separate research and production branches. You cannot make HTTP requests via the browser to communicate with TensorFlow Serving. Instead, the server, built on C++ for performance, implements a gRPC interface. gRPC is Google's Remote Procedure Call framework designed to performantly connect services in and across datacenters in a scalable fashion. Thus, a client will need to be built to communicate with the server. Deep learning and machine learning models must be saved in Google's protobuf

format. TensorFlow Serving can (auto) scale within CloudML or by using Docker/Kubernetes.

In-Process Serving

In some cases, organizations might not want to deploy TensorFlow Serving or cannot use the TensorFlow Serving RPC server to serve models. In these situations, you can still use saved models directly by including core TensorFlow libraries in the application. In-process serving offers a very lightweight mechanism to provide inference capabilities but none of the benefits provided by TensorFlow Serving, like automated request batching or model versioning.

As an example, let's consider a basic website built with the Python library Flask. This website will allow a user to upload an image and identify objects within the image using a deep convolutional neural network. From our perspective, the interesting part happens after the user has uploaded the photo and after the trained convolutional neural network has been loaded. Upon receipt of a photo, the Flask server would show the network the input photo, perform the inference, and return the results. All of the inference capability would be provided by TensorFlow libraries that could easily be called by the Flask-based web server. A similar library approach is used for inference on mobile devices (see *http://tensorflow.org/mobile*).

Integrating with Other Systems

A critical aspect for any new technology being considered for adoption in the enterprise is how it fits into the existing corporate infrastructure. Although today's big data landscape is incredibly crowded and complex, there are some obvious technologies that play a role in many big data stacks across industries.

Data Ingestion Options

Key to deep learning are the often massive amounts of data that must be cleaned, conditioned, and then used to train neural networks. For this to happen, before anything else the data must be ingested. Fortunately, there are many options. First, TensorFlow supports its own native TensorFlow format (tf.Example and tf.SequenceExample) built on protocol buffers in TFRecords. Note that Apache Beam has native support for TFRecords. Second, and slightly slower, TensorFlow has built in functionality to read JSON,

comma-separated value (CSV), and Avro data files. Finally, the end user can use Python to read data, including data from Pandas data tables. Because this last option is slowest, it is best for testing and experimentation. Finally, TensorFlow supports several different distributed storage options including Apache Hadoop HDFS, Google Cloud Storage, and Amazon Elastic File System.

TensorFlowOnSpark

Yahoo was kind enough to open source code that allows distributed TensorFlow training and inference to run on clusters built for Apache Spark. From the enterprise perspective, this is potentially very powerful as many shops looking to use TensorFlow might already be using Spark for data analysis and machine learning and have a Spark cluster operational. Thus, the organization can reuse its existing cluster assets instead of setting up separate infrastructure solely for deep learning, making the transition significantly easier. Further, this can alleviate the need to move data from one cluster to another—an often painful and time-intensive process.

From a tactical perspective, TensorFlowOnSpark is compatible with TensorBoard, going so far as configuring a Spark executor to run Tensorboard during training on cluster setup. The API is minimal, making it quick to learn and use and requiring very few changes to existing TensorFlow code to run. TensorFlowOnSpark provides the means to do three things:

- Start/configure a TensorFlow cluster within spark
- Feed data to a TensforFlow graph by converting Spark's Resilient Distributed Datasets (RDDs) to feed_dict
- Shutdown the TensorFlow cluster when finished

To make the most use of Spark, you want to run the type of TensorFlow programs that will fully saturate the resources; otherwise, performance will not scale linearly as with any distributed application. In terms of downsides, TensorFlowOnSpark is not fully compatible with all community projects (like Keras). Further, Spark running in the Java Virtual Machine (JVM) can provide some relatively inscrutable error messages upon failure. Regardless, this is possibly the easiest way to run distributed TensorFlow for training if your enterprise is already using a Spark cluster.

"Ecosystem" Repo

The ecosystem repo is an Apache 2.0 licensed open source repository on GitHub from Google that contains examples integrating TensorFlow with numerous open source software, including those listed here:

Docker
: A set of example Dockerfiles to build containers with various TensorFlow configurations.

Kubernetes
: A YAML template file for running distributed TensorFlow on Kubernetes.

Marathon (on top of Mesos)
: A configuration file for TensorFlow running in Marathon, a container orchestration for Mesos, which is a cluster manager.

Hadoop
: An implementation of the InputFormat/OutputFormat for Apache Hadoop MapReduce using the TRRecords format.

Spark-tensorflow-connector
: A library for reading and writing TensorFlow records (TFRecords) in and out of Spark 2.0+ SQL DataFrames.

Consider the "ecosystem" repo a starting point to exploring how you can integrate TensorFlow with other software.

Accelerating Training and Inference

Training deep neural networks takes a significant amount of computational horsepower, often exceeding what a cluster of general-purpose microprocessors can deliver. However, as deep learning's value became more obvious, the search for higher performance hardware became critical. GPUs were quickly repurposed for this task and, later, custom hardware designed specifically for this use case was and is in development. The important thing to note is that without sufficient training data and sufficient computational horsepower, deep learning would be irrelevant and would not have experienced its impressive success to date.

GPUs and CUDA

Using the graphics processing units (GPUs) to perform floating-point calculations in a massively parallel fashion has intrigued performance-minded programmers for nearly two decades. In fact, the term general-purpose computing on GPUs (GPGPU) was coined in 2002. NVIDIA has been a long-standing promoter of this use case and developed its proprietary Compute Unified Device Architecture (CUDA) as a parallel computing platform and programming model for the company's GPUs.

Training deep learning networks has emerged as the killer application for this field and NVIDIA augmented its CUDA offering with the NVIDIA Deep Learning Software Development Kit, which contains a GPU-accelerated library of key primitives needed for neural networks called cuDNN. Using the fastest GPU available from NVIDIA can offer a 10- to 100-times speedup for training deep networks versus the fastest available CPU from Intel.

Tensor Processing Units

GPUs used to rule the benchmarks for accelerating deep neural networks until the world learned of Google's Tensor Processing Units (TPUs) at Google IO in May of 2016. The first-generation TPU was announced in May 2016 at the Google I/O conference and only accelerated inference workloads (not training) using quantized integer arithmetic and not floating point. An excellent technical overview of the first first-generation TPU is online here, and a very thorough technical article on its performance was presented this past year and is available online.[2] Importantly, this first-generation TPU has been in operation in Google's datacenters for more than a year and helped power Google's AlphaGo win over Go World Champion Lee Sedol.

The second generation of TPU was announced in 2017 and can perform both inference and training and do floating-point arithmetic. Each individual processor offers 45 teraflops of performance and are arranged into a four-chip, 180-teraflop device. Sixty-four such

2 N.P. Jouppi et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit", *Proceedings of the 44th Annual International Symposium on Computer Architecture* (June 2017): 1-12.

devices are assembled into a pod that offers 11.5 petaflops of deep learning performance.[3] The key to both chips, as is important for any server-oriented processor, is that they not only provide impressive floating-point performance, but also consume less power than traditional processors.

Why does this matter for the enterprise? Because the TPU reduces the cost and time required to train models. Even though Google has no plans to sell TPUs, this capability is being made available via Google's Cloud offerings. Further, there are some intriguing options for Google to optimize across both software and hardware given that the company controls the entire stack. Google is not the only company in this space; Nervana, a small company making custom silicon for accelerating deep learning, was purchased by Intel in August of 2016.

Google Cloud TPU and CloudML

Cloud TPU is a Google Cloud service offering currently in alpha that gives users the ability to perform training and inference of machine learning models using the second-generation TPUs. You can connect to the Cloud TPUs from both standard and custom virtual machine types, and the offering is also fully integrated with other Google Cloud Platform offerings including Google Compute Engine and BigQuery.[4] This is the most direct way for the enterprise to take advantage of Google's TPUs. Google also exposes TPUs indirectly through some of the functionality of the Cloud Machine Learning Engine (Cloud ML).

3 Patrick Kennedy, "Google Cloud TPU Details Revealed," STH, May 17, 2017, *https://www.servethehome.com/google-cloud-tpu-details-revealed/*.

4 Cloud TPUs - ML accelerators for TensorFlow | Google Cloud Platform.

Summary

The question for any enterprise adopting deep learning is how will it integrate into the organization's existing workflows and data pipelines. The TensorFlow data pipeline is composed of three stages: (1) data preparation, (2) model training, and (3) model serving and inference. All three see substantial support from both the TensorFlow library directly and the emerging ecosystem. This data pipeline is very similar to the traditional machine learning pipeline found in the enterprise with one notable difference; model training can be substantially more time and resource intensive for deep learning models. The ecosystem attempts to remedy this situation with support for multiple GPUs and even Google's own TPUs.

Conclusion

This report attempted to introduce deep learning and frame core questions around its use and the use of TensorFlow for the enterprise. We first looked at the case for deep learning, especially when compared to more traditional machine learning techniques, and examined some of the state-of-the-art applications for which deep learning technologies have been used. If your organization is interested in using audio, video, image, or free-text data, deep learning is worth exploring. Next, we examined various frameworks for deep learning, with particular attention paid to libraries focused on the enterprise, including Microsoft's Cognitive Toolkit, MXNet, and Deeplearning4J. Finally, we surveyed the TensorFlow library and the existing ecosystem to understand how various components complement the core capabilities of the library and assist with both network training and inference. We hope that this overview helps decision makers and technology leaders within the enterprise navigate the growing world of deep learning.

About the Authors

Sean Murphy is the co-CEO of PingThings, Inc., an AI-focused startup bringing advanced data science and machine learning to the nation's electric grid. After earning dual undergraduate degrees with honors in mathematics and electrical engineering from the University of Maryland College Park, Sean completed his graduate work in biomedical engineering at Johns Hopkins University, also with honors. He stayed on as a senior scientist at the Johns Hopkins University Applied Physics Laboratory for over a decade, where he focused on machine learning, high-performance and cloud-based computing, image analysis, and anomaly detection. Switching from the sciences into an MBA program, he graduated with distinction from Oxford. Using his business acumen, he built an email analytics startup and a data sciences consulting firm. Sean has also served as the chief data scientist at a series A-funded healthcare analytics company and the director of research and instructor at Manhattan Prep, a boutique graduate educational company. He is the author of multiple books and several dozen papers in multiple academic fields.

Allen Leis is an experienced data engineer and sometimes data scientist located in Washington D.C. While his former life entailed developing web systems for the US Navy, US Senate, and nonprofit organizations, he is currently devoted to the technological "wild west" of data analytics and machine learning. Allen is presently working as a consultant and software engineer for a variety of data science startups in order to bootstrap their data ingestion, wrangling, and machine learning efforts. When not heroically solving big data engineering and distributed computing problems, he can usually be found teaching for Georgetown University's Data Science certificate program or indulging in his hobby of computer science graduate courses at the University of Maryland.