A System Analysis of Improvements in Machine Learning

by

Sabin M. Thomas

B. S., Electrical and Computer Engineering
Worcester Polytechnic Institute

M.S., Computer Science
Boston University

Submitted to the System Design and Management Program in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Engineering and Management

at the

Massachusetts Institute of Technology
January 2015 [February 2015]

Signature of Author Signature redacted
Sabin M. Thomas
System Design and Management Program

Certified by Signature redacted
Abel Sanchez, Thesis Supervisor
Senior Lecturer, Engineering Systems Division

Accepted by Signature redacted
Patrick Hale, Director
System Design and Management Program

This page has been intentionally left blank

A System Analysis of Improvements in Machine Learning

by

Sabin M. Thomas

Submitted to the System Design and Management Program in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Engineering and Management

Abstract

Machine learning algorithms used for natural language processing (NLP) currently take too long to complete their learning function. This slow learning performance tends to make the model ineffective for an increasing requirement for real time applications such as voice transcription, language translation, text summarization topic extraction and sentiment analysis. Moreover, current implementations are run in an offline batch-mode operation and are unfit for real time needs. Newer machine learning algorithms are being designed that make better use of sampling and distributed methods to speed up the learning performance.

In my thesis, I identify unmet market opportunities where machine learning is not employed in an optimum fashion. I will provide system level suggestions and analyses that could improve the performance, accuracy and relevance.

Thesis Supervisor:	Dr. Abel Sanchez
Title:	Senior Lecturer, Engineering Systems Division

Acknowledgements

I wish to thank my fellow students, staff and professors of the System Design and Management program for their support in making my Masters program such a wholesome and rewarding experience.

I also thank my advisor, Dr. Abel Sanchez, who offered me invaluable insight and guidance during my thesis research. I also convey my appreciation to Pat Hale who provided valuable curriculum guidance and administrative support personally and through the personnel they direct.

Finally and most importantly, I dedicate this work to my wonderful wife. My success would not have been possible without her love, patience and constant support throughout this experience. I also am thankful to my family for their love and understanding.

Table of Contents

List of Figures

Introduction

Over the last two decades, machine learning has emerged as a requisite in many software applications. The availability of cheap computing and peta-scale data sets coupled with the focus on statistical research in natural language processing (NLP) has ordained a unique marriage that has resulted in many real-world applications such as voice transcription, automatic summarization, machine translation, information extraction.

However, these applications are not optimized at a system level. This lack of system optimization can spawn severely limited systems that fail miserably when there is significant variation in expected dataset size. Research at Microsoft showed that systems that are performant at 1 million words fail miserably at 1 billion words. Conversely, a poor performing system at 1 million words was a strong performer at 1 billion words (Banko and Brill 2001a). These failures translate into unreasonably long processing times for model training completion. These failures also could result in a drop in model relevance or accuracy. To the user of a voice transcription system based on NLP and machine learning, these failures translate into high rates of transcription errors and misrecognition of the user. This culminates in a systematically ineffective voice recognition system for the user.

Additionally, there are system limitations in current NLP implementations. A majority of model learning functions is set up to run in an overnight batch fashion.

The overnight or off-hours nature of this system limitation manifests into an archaic solution. This solution now becomes chronologically imbalanced when compared to the real-time nature of other solutions. For example, current implementations of automatic summarization functionality employed by eDiscovery software requires a batch run mode, where the learning function can vary from taking hours to days.

In the later sections, I will point out these system architectural flaws that have resulted in missed market opportunities in creating a fully featured enterprise Knowledge Management system. This new Knowledge Management system will benefit from system optimizations that will allow it to scale easily to peta-scale data sets. Furthermore, it will be able to train the model via online-learning utilizing streaming and in-place model updates. These methods circumvent the slowness of batch mode model training.

This real-time and high-performing Knowledge Management system has immediate business impact. It has applications in powering eDiscovery legal software, enterprise search, enterprise content management systems, etc. Additionally, this new improved system will realize the untapped market potential in being able to integrate across other enterprise systems such as customer relationship management systems (CRM), cyber security and enterprise workflow systems.

In a testament to this missed potential, Microsoft is ambitiously planning to invest in and launch an upcoming product called Office Graph that will collect behavior and social metadata from all Office 365 applications to enrich the search, discovery, navigation, and filtering of all knowledge among these components.

Literature Review

The prevalence and increasing commercial applicability of machine intelligence is made more evident by looking at the number of firms and startups that are in the field of artificial intelligence, machine learning or big data. Figure 1 depicts this landscape. Shivon Zilis, a Venture Capitalist at the Bloomberg Beta fund, has termed machine intelligence, in an investment context, to be a unifying term for machine learning and artificial intelligence. (Zilis 2014)

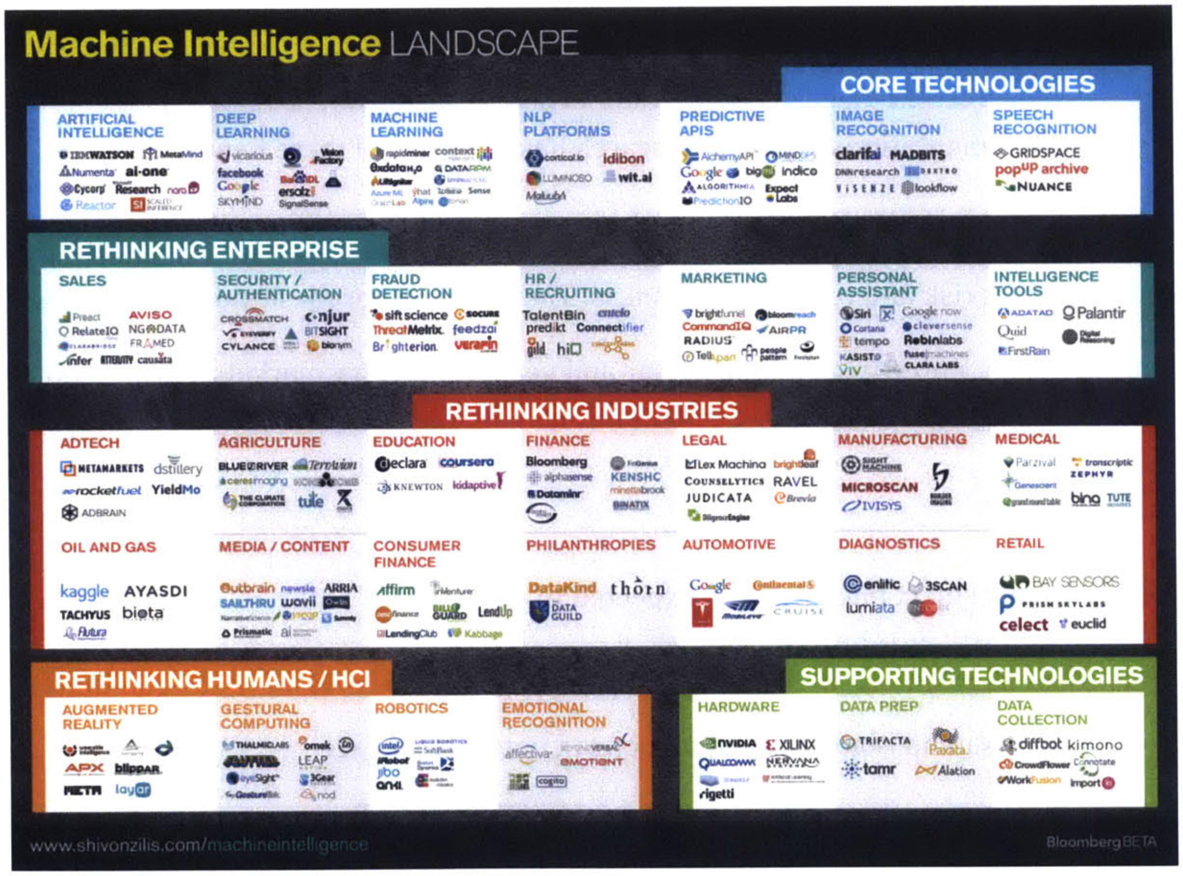

Figure 1 Machine Intelligence Landscape, Source: (Zilis 2014)

There are a number of reasons that can explain the dizzying array of companies that seek to be involved in the machine intelligence landscape. There could be a number of contributing factors but the following at the very least provide some insight into this.

- Lowering of computational barriers
- Abundant sources of rich data
- Increasing need for intelligent solutions

These companies are working to deliver complex systems that utilize machine learning implementations. Gartner Research has categorized these benefactor machine learning systems as Smart Machines, a rapidly emerging set of technologies that pose great risk and reward to business. The use cases for these smart machines span consumer, enterprise, public-sector and vertical industry markets. (Brant and Austin 2014)

These complex smart machines are expected to

- Understand problems and their context
- Mimic human reactions to questions in natural language
- Make decisions using probabilistic models
- Predict future states

Gartner Research expects the product cycle for these smart machines to mimic the industry hype cycle graph depicted in Figure 2. The goal of Natural Language Processing (NLP) smart technology is linguistic competence comparable or superior to humans. NLP technologies in its current state fall far short of that goal, but have undergone significant advancement in research since 1950.

NLP is an umbrella category for many different capabilities. Many IT-based linguistic uses are already practical, powerful and commonplace (such as semantic analysis). Its uses go beyond enhanced human-computer interaction to include the ability to "understand" (draw inferences based on) large bodies of ever-changing content. (Brant and Austin 2014)

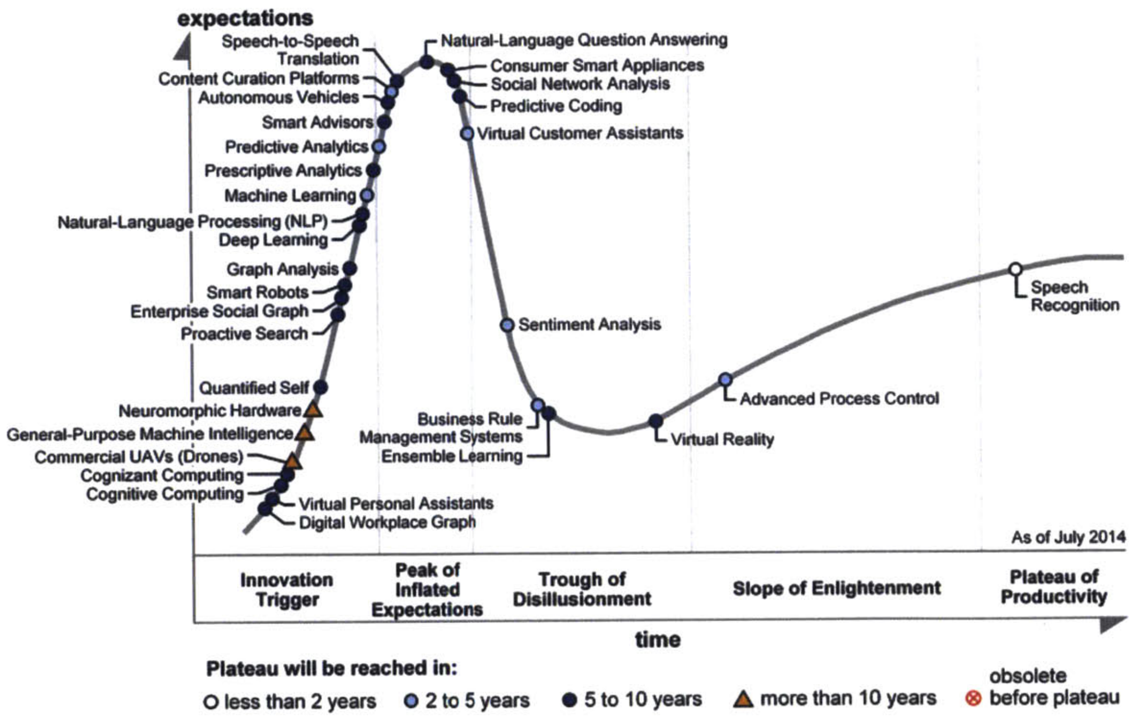

Figure 2 Smart Machines Hype Cycle for 2014 (Source: Gartner)

Architectures for NLP Machine Intelligence

In this section, I will provide an overview of some of the commonly available NLP Machine Intelligence architectures. These architectures involve the use of open source packages and frameworks. They also vary in their use of data ingestion patterns.

Private Cloud

Most vendors in the NLP Machine Intelligence space opt for an on premise setup of their environment. This option provides the most flexibility and additionally has the highest potential for an optimally tuned and performant system. The disadvantages for a private cloud setup are that the vendor will have to in-house many auxiliary functions requisite for managing this cloud – functions such as security and authentication management, network operations, operating system updates and firewall management. These auxiliary functions require appropriate staffing levels and can prove distracting for a vendor that is just starting up in the NLP Machine Intelligence space.

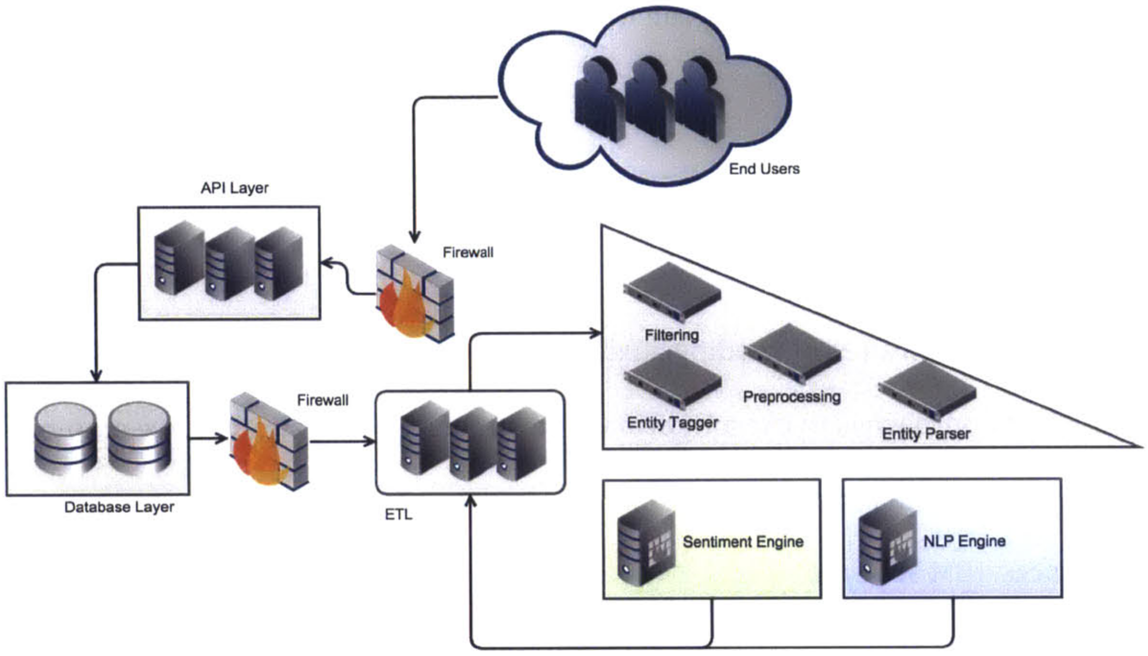

Figure 3 Typical Architecture Setup

In Figure 3, I provide an example of a typical private cloud setup. The reference to NLP and Sentiment engine in the diagram can be any number of currently available open source packages such as MALLET, Stanford NLP and Sentiment engine, Vowpal Wabbit, Apache Mahout.

Additionally, it is easy to glean from Figure 3 the dependence on common plumbing techniques such as the ETL (Extract, Transform, Load) systems that are required for the transfer of data from the application into the NLP/Machine Learning systems for appropriate processing. The ETL system may be used in back propagation if we are to use the results from the NLP/Machine Learning processing back into application logic. This plumbing technique of ETL suffers from not being realtime, as these data transformation processes typically run in batch mode during off-peak hours so as to not stress the systems.

Hybrid Cloud

To alleviate the pressures of needing auxiliary teams to manage a private cloud, a cost-efficient approach is to use a Hybrid Cloud strategy. In this strategy, we employ the NLP and Machine Learning cloud offerings from vendors like Google, Amazon Web Services, IBM Watson.

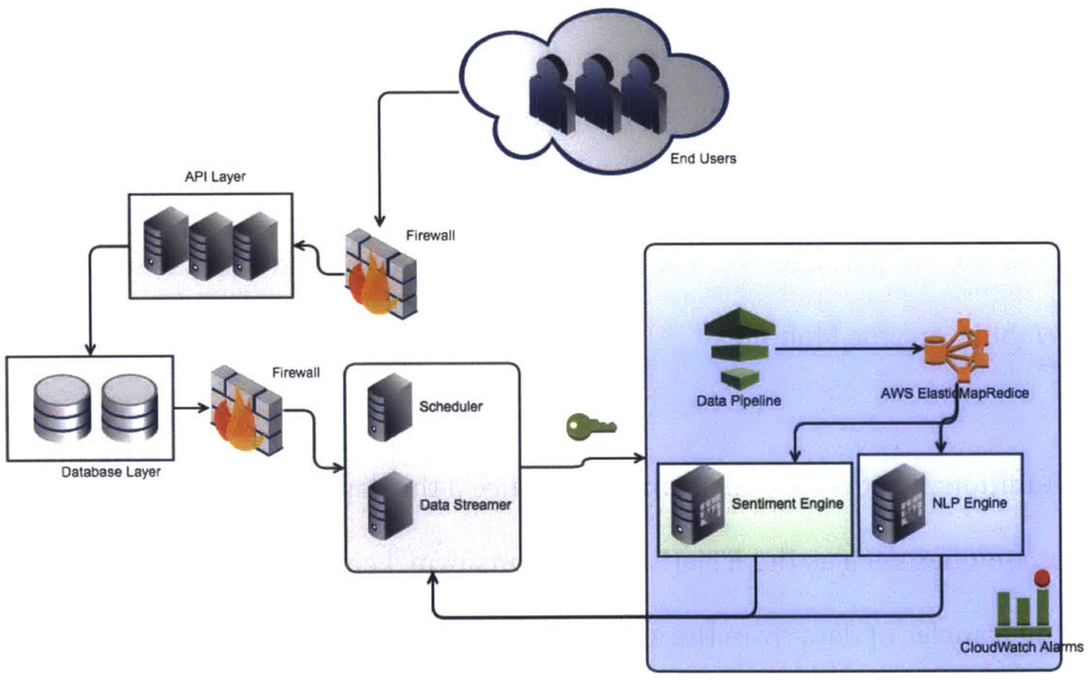

Figure 4 Hybrid Cloud Example using AWS

In Figure 4, I depict a sample hybrid cloud setup using components from Amazon Web Services such as AWS Elastic Map Reduce framework along with AWS Data pipeline, CloudWatch Alarms and Identity Access Management. These components can easily be spun up and down as per demand and is very elastic to the nature of the needs.

IBM Watson, Google Cloud and Microsoft Azure have similar offerings that allow for this API based invocation of NLP and machine learning processing.

This hybrid cloud approach offers the benefits of elasticity and low maintenance, which is particularly advantageous for startups with small engineering teams.

General Problems with current NLP Machine Intelligence

Machine learning offers immeasurable benefits and aids in building complex systems. However, there is a dangerous precedent that is played out with the rapid adoption of machine learning for existing systems. This precedent is that machine learning enabled systems are not scalable, run too slow, consume too much memory and are not built to handle even minute variations outside their operating range. These issues with machine learning systems can be categorized (Sculley et al.)

- boundary erosion
- entanglement
- hidden feedback loops
- undeclared consumers
- data dependencies
- changes in the external world
- system level anti-patterns.

One example of a problematic implementation is a current state of the art NLP classifier technology that works very well on a limited tree bank, typically less than a million words, but begins to fail when the same machine learning system is scaled up to a 1 billion word training corpus.

Michele Banko and Eric Brill at Microsoft Research conducted a series of experiments to evaluate the performance of confusion set disambiguation machine learning algorithms for a simple English grammar checker. Confusion set disambiguation is one of a class of natural language problems involving disambiguation from a relatively small set of alternatives based upon the string context in which the ambiguity site appears (Banko and Brill 2001b).

In all the machine learning approaches studied by Banko and Brill for this experiment, the confusion disambiguation problem is described as follows: Given a specific confusion set (e.g. {to,two,too}), all occurrences of confusion set members in the test set are replaced by a marker; everywhere the system sees this marker, it must decide which member of the confusion set to choose. (Banko and Brill 2001b)

Problems in Classification Accuracy

In these experiments, the learning curves for various machine learning algorithms were studied. The machine learning algorithms employed were

- winnow
- perceptron
- naïve Bayes
- simple memory-based learner

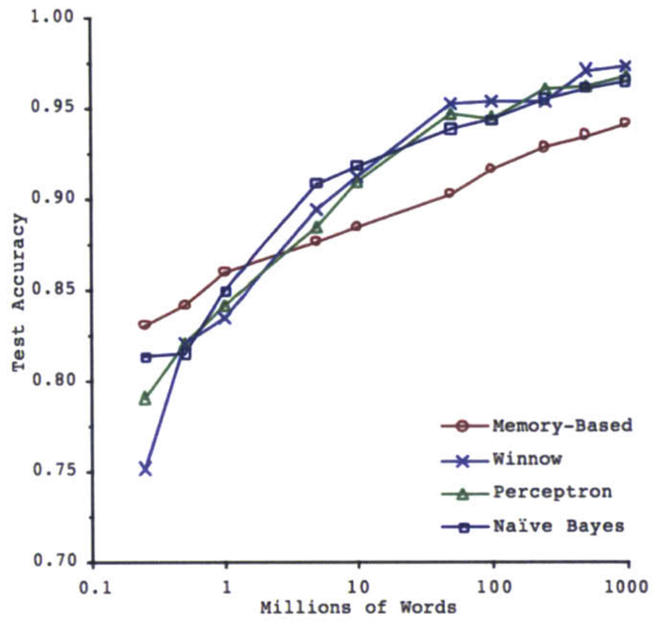

Figure 5 Learning Curves for Confusion Set Disambiguation, Source: (Banko & Brill, 2001)

Figure 5 clearly depicts what is described as non-ideal for a scalable algorithm. With increase in training corpus size from less than a million words to a 1 billion words, the accuracy increases for each implementation. This is non-ideal because in log-linear graph like we this, we should expect to see asymptotic increase in accuracy, as depicted in Figure 6, which we clearly do not see.

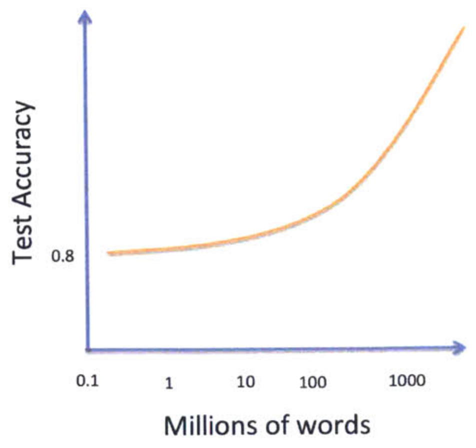

Figure 6 Ideal Learning Curve for scalable Confusion Set Disambiguation

This shows us that these machine learning implementations suffer from being optimized for a narrow training corpus range (~ 1million words) and are not able to cope with any increase in training corpus size. This represents a significant complexity in being able to find real world applicability for these algorithms.

Problems with Memory bounds

Ignoring the problems in classification accuracy, it was also demonstrated by Banko and Brill that these machine learning implementations also suffered from memory bound issues when scaling to very large data sets.

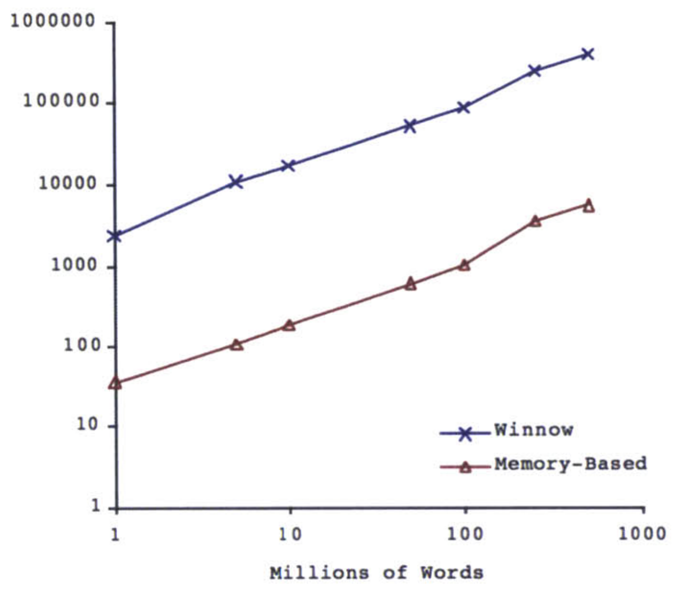

Figure 7 Memory Size vs Training Corpus Size, Source: (Banko and Brill, 2001)

In Figure 7 we see the bounds we hit in memory when trying to increase the corpus size for the afore mentioned machine learning implementations for confusion set disambiguation. This again represents a departure from real world applicability of these machine learning approaches in systems where RAM comes at a premium. To answer this issue, we would need to look into enabling compression on these machine learning algorithms.

Problems with CPU performance

A popular tool used for machine learning is Weka (Waikato Environment for Knowledge Analysis). Weka is a popular suite of machine learning written in Java, and was developed at the University of Waikato in New Zealand and is free software available under the GNU license. (Hall et al. 2009)

The ease of Weka is apparent in its GUI implementation, however, when attempting to use it for larger data sets, there are limitations that require tweaking of the Java VM heap. Even after making these tweaks, there are still limitations in expanding the data sets to larger sizes.

An NLP implementation test using Weka was conducted where a Webdocs dataset with 1.7 million transactions over 5.2 million unique items was processed. The Weka Apriori algorithm finds sets of items that appear frequently in transactions. To achieve 10% support, Weka continued processing for **3 days** at which point the implementation was terminated. An unverified implementation of the Apriori algorithm by a StackOverflow.com user "mvarshney" claims to have achieved processing in **4hr 24 min.** In this implementation, the user "mvarshney" attributed this massive speedup by use of Tries data structures instead of Weka's implementation in hashtables (Mvarshney). Additionally, "mvarshney"'s implementation claims to have achieved a CPU-bound improvement by taking 39 minutes to run on a single 8-core machine and taking 6min 30sec on 8 machines, representing a 400% improvement.

This example is one of many problems that can be found when attempting to apply stock machine learning algorithms and toolset against much larger data sets than what they were intended for before running into processor and memory limitations.

Current State of Machine Learning Optimizations

Machine learning has benefited from increased research both from academia and from industry. These two approaches represent different angles of optimization that can be combined together to manifest in powerful and pragmatic applications.

In the next few sections, I will evaluate these optimizations.

Distributed Deep Networks

Google has a vested interest in furthering industry research on Natural Language Processing, Machine Learning and Text Analytics. Google introduced the Google Search Appliance (now labeled "Google Search for Work") in 2002 in its first foray into the Enterprise Search and Enteprise Content Management (ECM) sector.

State of the art performance in text processing traditionally has been achieved through the use of deep learning and unsupervised feature learning. It has been observed that increasing the scale of deep learning by increasing the number of training examples, the number of model parameters, or both, can drastically improve the classification accuracy. (Dean et al. 2012)

Until very recently, the use of GPUs provided a significant advance in boosting computational power needed for crunching through unsupervised feature learning and deep learning implementations. However, a theoretical limit was reached when attempting to use GPUs with modestly sized data sets. A known limitation of the GPU approach is that the training speed-up is small when the model does not fit in GPU memory (typically less than 6 gigabytes). To get around this limitation, researches often reduced the size of the training data or parameters so that CPU-to-GPU transfers are not a bottleneck. (Dean et al. 2012)

However, reducing the size of the training data or parameters to get around the CPU-to-GPU transfer limitation worked well for small problems (ex: for acoustic modeling for speech recognition), they are less attractive for problems with a large number of examples and dimensions (ex: high resolution images or paragraph inference). (Dean et al. 2012)

Google's research scientists and engineers were able to combat this limitation by using large-scale clusters of machines to distribute training and inference in deep networks. They developed a software framework called DistBelief that enables model parallelism within a machine via multithreading and across machines via message passing. The DistBelief framework manages the details of parallelism, synchronization and communication. (Dean et al. 2012)

In addition to model parallelism, the DistBelief framework also supports data parallelism where multiple replicas of a model are used to optimize a single objective. The Google engineers also implemented two new methods for large-scale distributed feature learning: (Dean et al. 2012)

- Downpour SGD
- Sandblaster L-BFGS

Distbelief Framework

DistBelief supports distributed computation in neural networks and layered graphical models. The user defines the computation that takes place at each node in each layer of the model, and the messages that should be passed during the upward (alternatively forward or feedforward) and downward (alternatively backward or backprop) phases of computation. In the case of a neural network, 'upward 'is synonymous with 'feedforward' while 'downward' is synonymous with 'backprop'. In a Hidden Markov Model, they are alternatively called 'forward' and 'backward' (Dean et al. 2012)

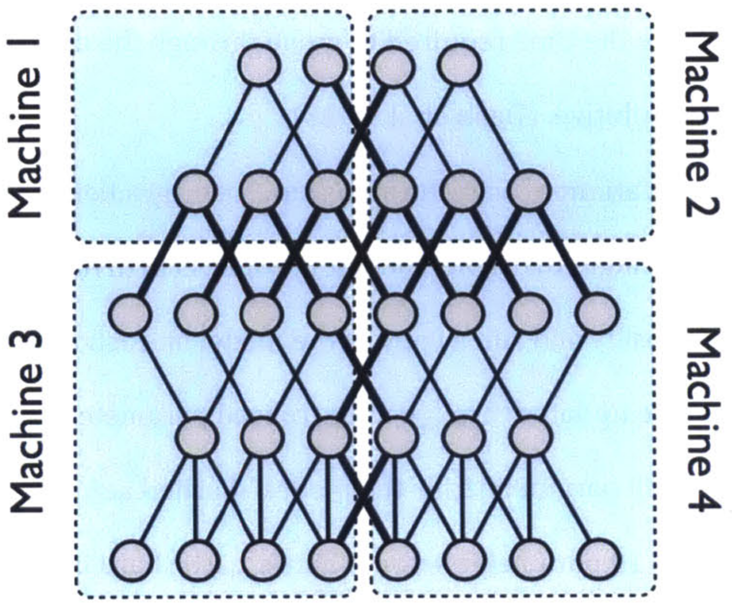

Figure 8 Model Parallelism in DistBelief framework

Error! Reference source not found. depicts the model parallelism with a five layer deep neural network with local connectivity partitioned across 4 machines. Only those nodes with edges that cross partition boundaries will need to have their state transmitted between machines. Even in cases where a node has multiple edges crossing a partition boundary, its state is only sent to the machine on the other side of that boundary once. Within each partition, computation for individual nodes will be parallelized across all available CPU cores.

Downpour SGD

Stochastic Gradient Descent (SGD) is perhaps the most commonly used optimization procedure for training deep neural networks. Unfortunately, the traditional formulation of SGD is inherently sequential making it impractical to apply to very large data sets where the time required to move through the data in an entirely serial fashion is prohibitive. (Dean et al. 2012)

Downpour SGD is a variant of asynchronous stochastic gradient descent that uses multiple copies of a single DistBelief model. The idea is to divide the training data into a number of subsets and run a copy of the model on each of these subsets. The models communicate updates through a centralized parameter server that keeps the current state of all parameters for the model, sharded across many machines. An example is if we had 10 parameter server shards, each shard is responsible for storing and applying updates to 1/10th of the model parameters. (Dean et al. 2012)

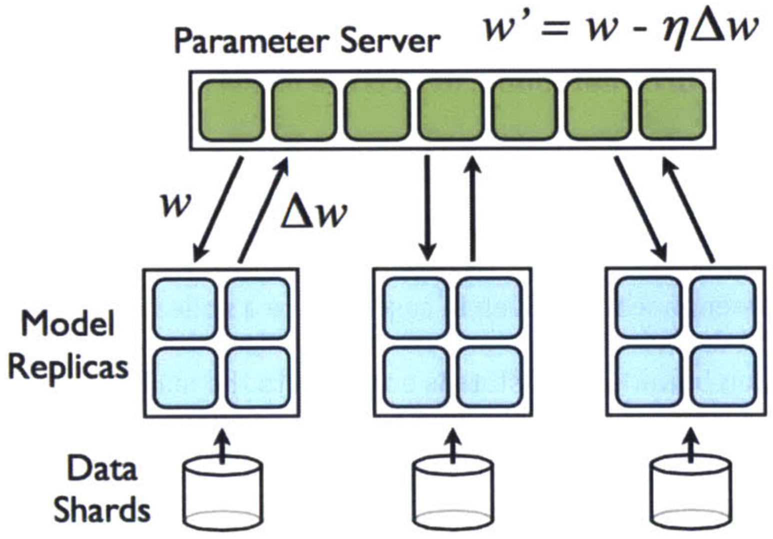

Figure 9 Downpour SGD System. Source (Dean et al. 2012)

The asynchronous and parallel aspect of this implementation arise from two features

- the model replicas run independently of each other
- the parameter server shards also run independently of one another.

The model parallelism results in significant speed up as described in the performance section. These parallelism features also bring robustness to this implementation that is not present in the synchronous implementation.

The synchronous implementation however was bound to a single point of failure where if one machine failed, the entire training process is delayed. However, in Downpour SGD, this robustness in parallelism prevents it from having a single point of failure.

The learning rate is further improved by the use of the Adagrad adaptive learning rate procedure. This adaptive learning rate procedure is implemented in each parameter server which offers advantages in robustness over using a single fixed learning rate. The use of Adagrad extends the maximum number of model replicas that can productively work simultaneously. Through a practice known as "warmstarting" model training, where only a single model replica is used for the training before introducing the other model replicas, stability concerns are virtually eliminated. (Dean et al. 2012)

Sandblaster L-BFGS

Sandblaster is an optimization framework especially applicable to batch methods for training in deep networks. Jeff Dean, and others at Google introduced this framework and implemented Limited Memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) on this framework.

The central idea to the Sandblaster optimization framework is distributed parameter storage and manipulation. The core of the L-BFGS algorithm resides in a coordinator process, which does not have direct access to the model parameters. The coordinator issues commands drawn from a small set of operations (dot product, scaling, coefficient-wise addition, multiplication) that can be performed by each parameter server shard independently, with the results being stored locally on the same shard. The history cache of L-BFGS is also stored on the parameter server shard on which it was computed. This allows running large models in the range of billions of parameters, without incurring the overhead of sending all the parameters and gradients to a single central server.

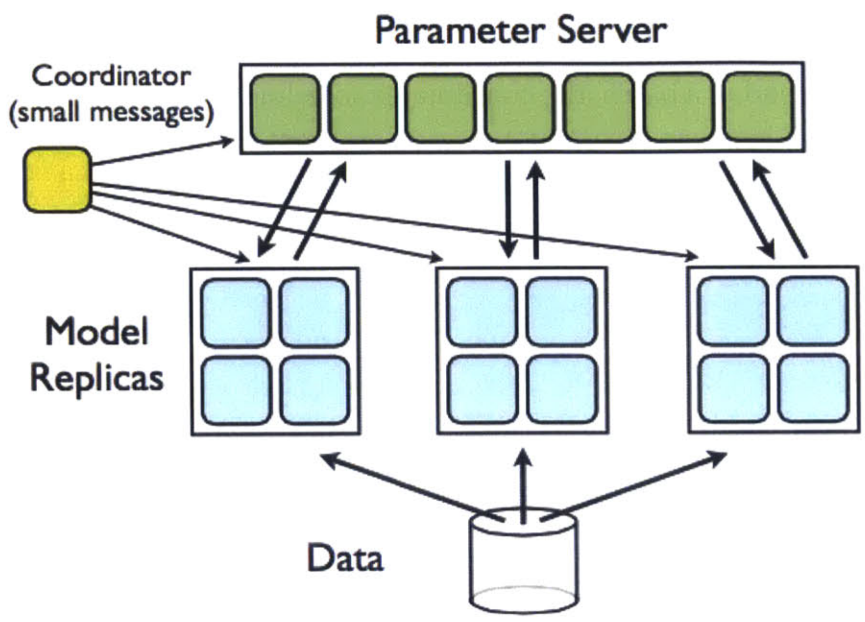

Figure 10 Sandblaster L-BFGS System. Source: (Dean et al. 2012)

Some implementations of L-BFGS attempt to parallelize the training step by distributing data to many machines and by having each machine compute the gradient on a specific subset of data examples. The gradients are then sent back to the central server. These typical implementations suffer from having to wait for the slowest machine in the farm to respond back with the computed gradient. This bottleneck thus makes it impossible for such an implementation to scale well to large shard clusters.

The Sandblaster framework distinguishes itself from this scalability issue by employing a load-balancing scheme. The coordinator assigns each of the N model replicas a portion of work, that is much smaller than 1/Nth the total size of a batch, and assigns model replicas new portions whenever they are free. In this approach, faster model replicas do more work than slower replicas. To further manage slow model replicas at the end of a batch, the coordinator schedules multiple copies of the outstanding portions and uses the result from whichever model replica finishes first. (Dean et al. 2012)

Performance

The team at Google conducted a series of experiments to validate the parallelism benchmark. Here they measured the mean time to process a single mini-batch for simple SGD training as a function of the number of machines used in a single model instance. The average training speed up is measured as the ratio of the time taken using only a single machine to the time taken using N. (Dean et al. 2012)

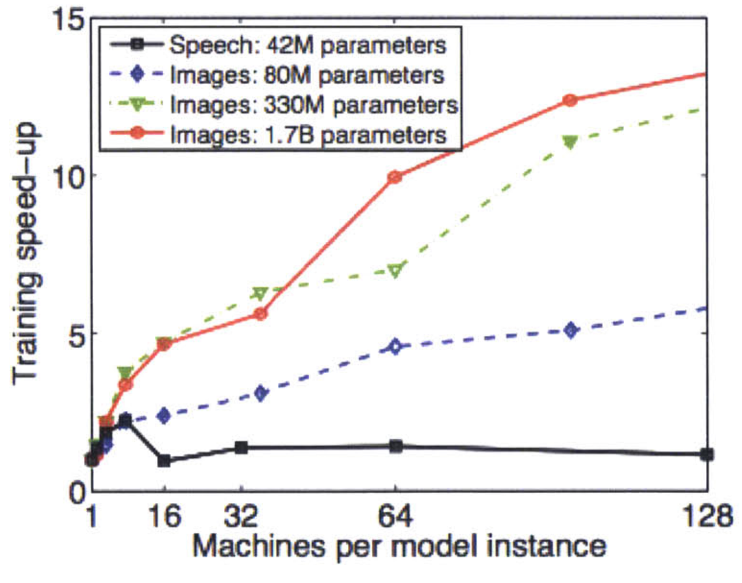

Figure 11 Training Speedup for DistBelief framework, Source: (Dean et al. 2012)

Figure 11 shows the DistBelief framework impact on training speed up for four different networks (speech and three image deep network) as a function of machines allocated to a single DistBelief model instance. Models with more parameters benefit more from the use of additional machines that do models with fewer parameters.(Dean et al. 2012)

Additional experimentation for gauging performance in the speedup of the DistBelief framework using Sandblaster L-BFGS and Downpour SGD were conducted. The improvements were benchmarked for the two following criteria

- Training Accuracy Speed Up
- Classification Accuracy Speed up

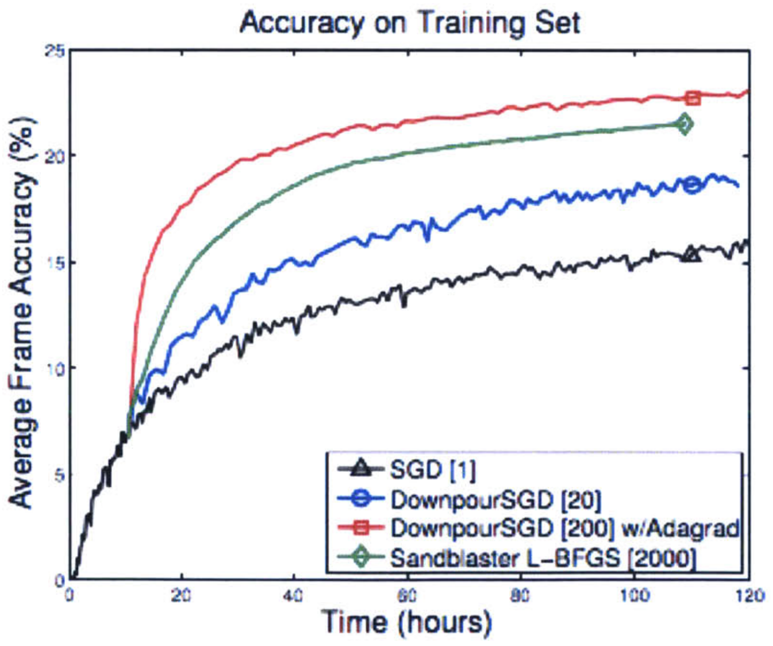

Figure 12 Classification Accuracy on training data as a function of training time,

Source: (Dean et al. 2012)

Figure 12 shows improvement in classification accuracy for each of the four deep networks. The ideal scenario involves obtaining the maximum test set accuracy in the minimum amount of training time, regardless of computing resources. For this experiment, the team at Google, kept the computational resource requirements stable. We do not consider the impact of computation requirements such as run on a 20 core machine versus an 8 core machine, or whether the training was conducted on a vanilla server or on a GPU using CUDA.

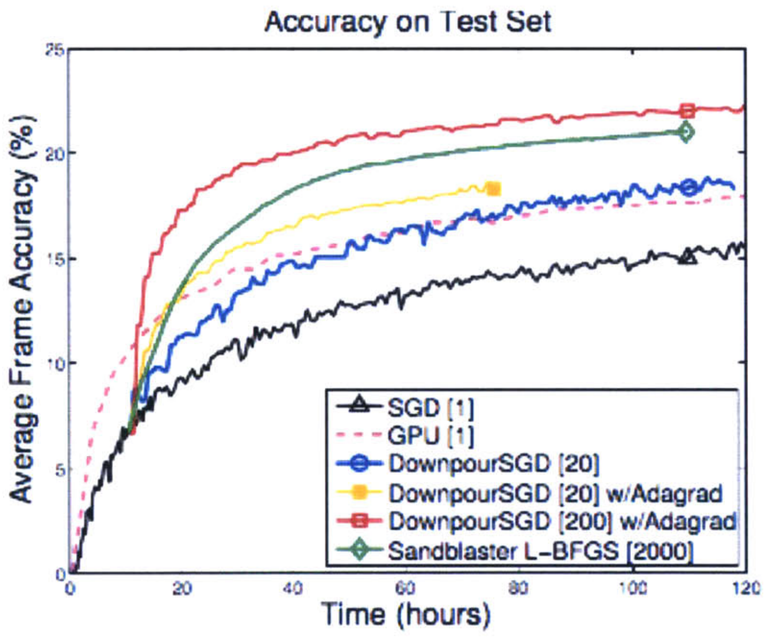

Figure 13 Classification accuracy on test data as a function of training time, Source:

(Dean et al. 2012)

Figure 13 depicts classification performance for a test data set. Conventional single replica SGD (black curves) is the slowest to train. Downpour SGD with 20 model

Error! Reference source not found. depicts classification performance for a test data set. Conventional single replica SGD (black curves) is the slowest to train. Downpour SGD with 20 model replicas (blue curves) shows a significant improvement. Downpour SGD with 20 replicas plus Adagrad (orange curve) is modestly faster. Sandblaster L-BFGS using 2000 model replicas (green curves) is considerably faster yet again. The fastest, however, is Downpour SGD plus Adagrad with 200 model replicas (red curves). Given access to sufficient CPU resourses, both Sandblaster L-BFGS and Downpour SGD with Adagrad can train models substantially faster than a high performance GPU.

Application within Google

Google was successfully able to use this DistBelief system to train a deep network 30x larger than previously reported, and achieves state-of-the-art performance on ImageNet, a visual object recognition task with 16 million images and 21,000 categories. Google was also able to demonstrate that these same techniques dramatically accelerate the training of a more modestly- sized deep network for a commercial speech recognition service. Google's focus is primarily on large neural networks, however, the framework and underlying algorithms are applicable to any gradient-based machine learning algorithm.

Distributed Asynchronous Online Learning

In the previous section, I evaluated Google's distributed framework with adaptive learning optimizations. In this section, I will highlight research that has combined distributed computing with asynchronous or "online" learning.

Modern NLP models are notoriously difficult and expensive to train. Two recent developments have led to major improvements in NLP models. (Gimpel, Das, and Smith 2009)

- Online learning algorithms

  These algorithms update the parameters of a model more frequently, processing only one or a small number of training examples, called a "mini" batch between updates.

- Distributed computing

  Training of the model is divided among multiple CPUs for faster processing between updates.

Online learning algorithms are optimized to offer fast convergence rates and scalability to large datasets. Distributed computing however is a more natural fit for algorithms that require a lot of computation to be done between updates. Typically, distributed online learning has been done in a synchronous setting, meaning that a mini-batch of data is divided among multiple CPUs, and the model is updated when they have all completed processing. (Gimpel, Das, and Smith 2009)

Synchronous frameworks have the drawback that they are only able to benefit from parallelism within one mini-batch iteration. Additionally, empirical research seems to suggest that online methods only converge faster than batch algorithms when using very small mini-batches. Thus using a synchronous framework for online learning will not offer much benefit.(Gimpel, Das, and Smith 2009)

In Gimpel, Das and Smith's paper on distributed asynchronous online learning, they focused on asynchronous algorithms, where multiple mini-batches are processed simultaneously, each using potentially different and stale parameters. The key advantage of an asynchronous framework is that it allows processors to remain in near constant use, preventing them from wasting cycles awaiting other processors to complete their portion of the current mini-batch. In this way, asynchronous algorithms allow more frequent parameter updates, which speeds up convergence.

Natural Language processing tasks such as named entity recognition and unsupervised parts-of-speech tagging have been evaluated with in Gimpel, Das and Smith's research, and have shown to converge much faster when using a distributed asynchronous framework with online learning. (Gimpel, Das, and Smith 2009)

Streaming for large scale NLP

In the earlier sections, I discussed some challenges with current machine learning implementations where an explosion in data sets has caused issues with CPU performance as well as memory bounds that slow down the learning rate of a machine learning model, or render the model ineffective.

In research by Goyal, Daume and Venkatasubramanian, a streaming algorithm paradigm was explored to be able to handle large amounts of data for NLP applications such as speech recognition, spelling correction and information extraction for use within eDiscovery, Enterprise Search and Enteprise Content Management (ECM) solutions.

Traditional approaches for NLP tasks have involved long computational times along with large memory requirements. For example, one traditional experiment required 1500 machines for a day to compute the relative frequencies of n-grams from 1.8TB of web data (Goyal, Daumé III, and Venkatasubramanian 2009). The resulting language model comprised 300 million unique n-grams. This memory requirement for 300 million unique n-grams along with the computational horsepower required to compute this model is a missed opportunity in current eDiscovery, Enterprise Search and ECM applications.

Some optimization techniques have been proposed in the past that involved applying either entropy pruning or count pruning to reduce the memory requirement for a large language model, such as the 300 million unique n-gram model. However, employing these pruning approaches to reduce the size of the language model results in two difficulties when the order of the language model increases, say from 1.8TB to 1.8PB of web data. (Goyal, Daumé III, and Venkatasubramanian 2009)

- The computation time to build the database of counts increases rapidly.
- The initial disk storage required to maintain these counts, prior to applying the pruning for compression is enormous.

The streaming method solves both these problems. Instead of estimating a large model and then applying pruning to compress it, the approach directly estimates a small model. This is done by using a deterministic streaming algorithm that computes the approximate frequency counts of frequently occurring n-grams. Experiments have shown that directly storing approximate counts of frequent 5-grams compared to using pruning gives equivalent business application performance, while dramatically reducing the memory usage and avoid the pre-computation of a large model. (Goyal, Daumé III, and Venkatasubramanian 2009)

Other Optimizations

There are many other optimizations well suited to the specific type of implementation of NLP machine learning algorithms.

Latent Dirichlet allocation is a machine learning technique that is used for text summarization in NLP. In Latent Dirichlet Allocation, we can use Gibbs sampling as an approximation to representing a collapsed Dirichlet distribution. This sampling method is advantageous as it reduces the computing time in needing to keep this distribution updated, and additionally reduces the overall memory footprint, allowing it to be embedded into applications easier. (Porteous et al. 2008)

Applications to Missed Opportunities

The above system improvements to machine learning have immense business applicability to the realm of eDiscovery, enterprise Search and Enterprise Content Management(ECM). In the following sections, I posit how applying these optimizations can result in huge upside potential for current vendors in the ECM and eDiscovery markets.

Enterprise Content Management

The worldwide market for ECM software grew by 8.6% in 2013, to a revenue total of $5.1 billion, which indicates that ECM technologies continue to attract more users and deliver value to enterprises. (Gilbert et al. 2014)

Creative and varied uses of ECM tools and services abound as organizations move well beyond basic uses, such as for secure file storage in organized libraries, to tackle deeper business challenges that need strong and flexible process capabilities. This has led organizations increasingly to regard ECM as an environment for solutions that meet a range of business needs, from departmental requirements to more complex enterprise requirements. In addition to focusing on solutions to meet this demand, more ECM vendors are offering cloud-based environments, mobile interfaces and social capabilities to meet the market's needs. (Gilbert et al. 2014)

Gartner Research expects to see a generational "makeover" with ECM moving further away from its roots in networked back-office environments. Gartner Research posits the concept of "content in context" which will be key to the ECM market's evolution as enterprises increasingly need content to be delivered in a personalized fashion — to the right people, at the right time, on the right devices, and in the context of particular business processes or needs.

This "content in context" market opportunity represents the biggest blind spot for the leading vendors in the ECM space such as Microsoft, IBM, EMC and others.

By applying the optimizations for large scale Streaming NLP, as discussed earlier, these ECM solutions can easily be extended to ingest multiple corpuses of information such as Customer Experience Data. Additionally, the distributed asynchronous online learning optimizations, also discussed earlier, allow for these ECM solutions to continually update their learning models in real time. These combinations provide easy extensibility for ECM solutions to fill this untapped market void for "content in context" type solutions.

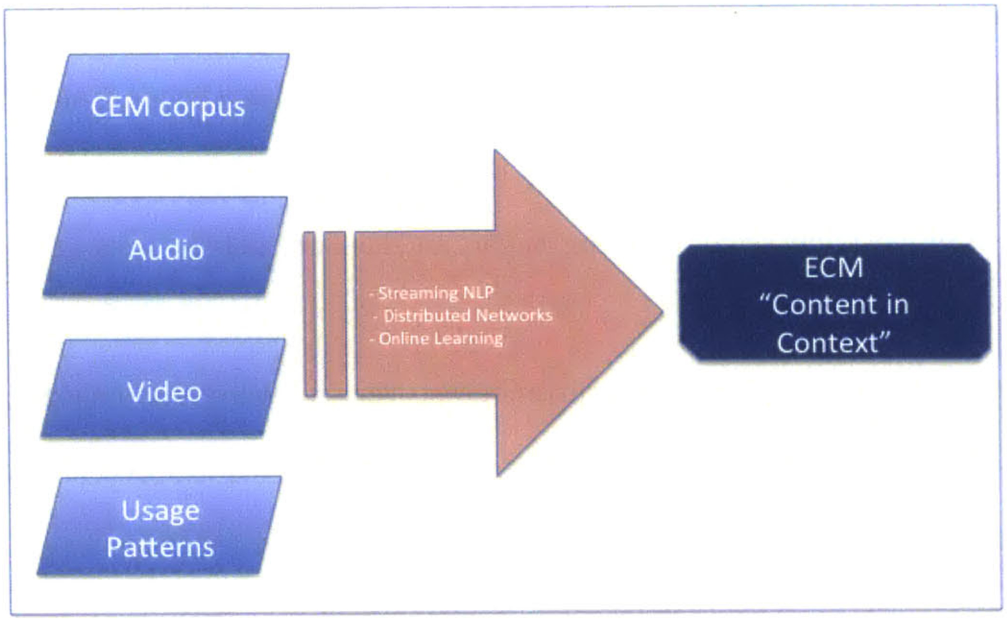

Figure 14 ECM "Content in Context"

eDiscovery

Gartner forecasts that revenue in the enterprise e-discovery software market will grow from $1.8 billion in 2014 to $3.1 billion in 2018. Double-digit revenue growth of approximately 15% is expected because of increasing volumes of litigation and regulatory investigation, and the ever-expanding amount of ESI that must be searched in support of these activities. Corporations continue to move from relying on service providers for the identification, preservation, collection and processing of data to managing the discovery process themselves, in-house. (Zhang, Logan, and Landers 2014)

According to Gartner, there are a number of adjacent software markets, including information governance, EIA, enterprise content management, file analysis, enterprise search and data analytics. Gartner expect that vendors in these areas will extend their offerings to include e-discovery functions and that vendors already in the e-discovery market will add further capabilities, for example, from the content analytics or workflow sectors. (Zhang, Logan, and Landers 2014)

However, an unaddressed need by current eDiscovery software vendors is true integration with enterprise cyber security solutions. Such an integrated solution would offer real time event trigger based data collection and review.

This is a missed opportunity in the eDiscovery market because of the technical complexity that this integrated solution would need to overcome. Aside from the challenge of real time scanning of every definable event (emails, texts, chats, web page views, access requests), this solution would also have to contextualize associated content with this event, such as relevant body of the text, chat, voicemail; relevant content in the web page, attachment and relevant documents in stored in the ECM solution.

All this complexity becomes a great fit for the machine learning optimizations discussed in earlier sections. Use of distributed deep networks along with the large scale streaming capabilities, this solution could easily tap into any source of these events and additionally contextualize using the less expensive memory representations of models using sampling and other optimizations. Additionally, the machine learning models could easily be updated in a distributed, online and asynchronous fashion.

This cyber security incident triggered eDiscovery solution now becomes all too realizable technically, and is sure to gain a large slice of the $3.1 billion eDiscovery market.

Summary

In this research I have identified unmet market opportunities in the Enterprise Content Management and eDiscovery market space. I have also identified where most of today's NLP applications fall short of being an optimal and scalable system solution. I have also evaluated recent improvements in machine learning approaches to NLP. These optimizations provide the missing link to be able to fill the void of unmet market opportunities.

The Future

In the past few years there has been an increasing number of NLP applications to ameliorate communication between human and machine. This communication is increasingly required to be further refined and enhanced and to emulate human conversational patterns. The big data phenomenon has also created a rising demand for robust NLP technologies that can extract and process human conversation on a real time basis. Certain verticals such as healthcare and the financial capital markets have seen increased demand for these technologies. Furthermore, the rising amounts of social media content having rich unstructured information about customer's perception and brand value has further encouraged the use of NLP technologies in various applications. (Marketsandmarkets.com 2013)

Currently, rule based NLP, statistical NLP and hybrid NLP solutions are leading forms of NLP solutions. There is a much stronger need for innovation in this market. Organizations are increasingly spending on building far more effective and innovating processing engines that can be used with new technologies, such as machine to machine communication and multimodal data receptions.(Marketsandmarkets.com 2013)

The healthcare industry is seeing an emergent need for NLP applications, with the most dominant need being computer assisted bill coding in healthcare. The usage of NLP can also be customized based on functionality types, response requirements and business language followed in the industry. For example, the healthcare sector has already established standards such as ICD-9 and ICD-10.

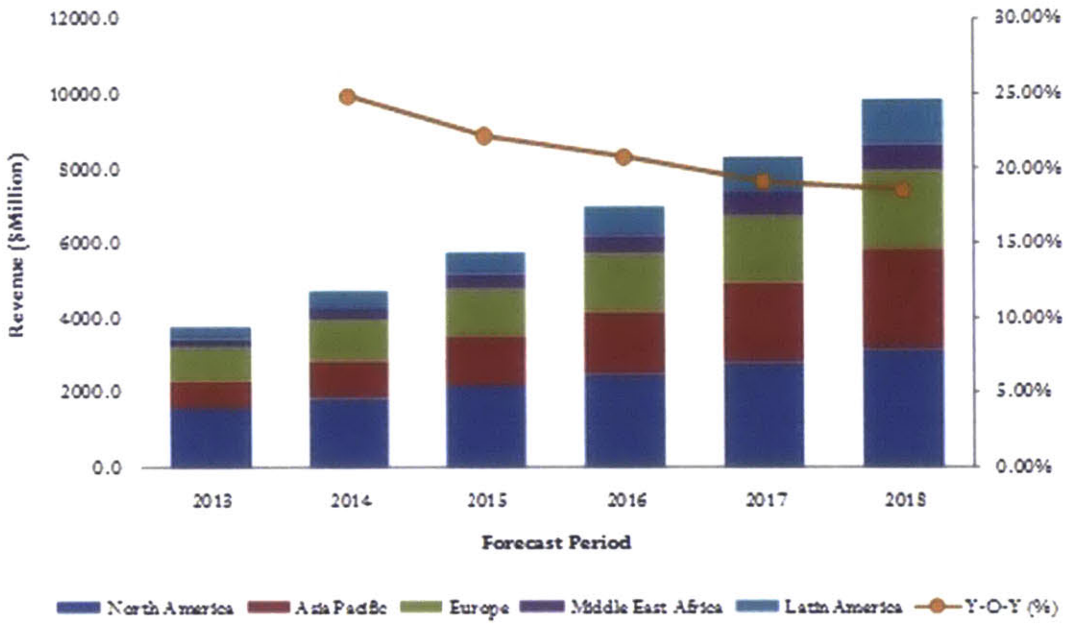

Figure 15 NLP Market and Y-o-Y Growth (Source: MarketsAndMarkets.com)

Recently there have been mergers and acquisitions of newer companies in the NLP market. Google's $500 million purchase of AI company DeepMind in January of 2014, Facebook announcing the set up of its new Artificial Intelligence lab, IBM's Watson supercomputer now working on deep learning and Yahoo's recent acquisition of photo analysis startup LookFlow to lead its new deep learning group (Shu 2014). Many emerging players are also entering into the market with unique capabilities and innovative products and solutions. With a focus on providing solutions via the cloud as a Software as a Service platform, these emerging players are responsible for the growing adoption of acceptance of NLP technologies in the market. (Marketsandmarkets.com 2013)

The future for NLP technologies is bright and can be generalized into categories for growth.

- Recognition technologies such as Interactive Voice Response (IVR), Optical Character Recognition (OCR) and pattern and image recognition
- Operational technologies such as auto bill coding for healthcare, categorization and classification technologies
- Analytics such as speech and text analytics

Figure 15 depicts the projected year on year forecast according to MarketsAndMarkets.com for NLP technologies. The market is expected to grow from $3,787.3 million in 2013 to $9,858.4 million in 2018, which represents a compound annual growth rate of 18% from 2013 to 2018. (Marketsandmarkets.com 2013)

References

Banko, Michele, and Eric Brill. 2001a. "Mitigating the Paucity-of-Data Problem: Exploring the Effect of Training Corpus Size on Classifier Performance for Natural Language Processing." *Computational Linguistics*, 2–6.

———. 2001b. "Scaling to Very Very Large Corpora for Natural Language Disambiguation." *ACL '01 Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, 26–33. doi:10.3115/1073012.1073017.

Brant, Kenneth, and Tom Austin. 2014. *Hype Cycle for Smart Machines, 2014*.

Dean, Jeffrey, Greg S Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V Le, Mark Z Mao, et al. 2012. "Large Scale Distributed Deep Networks." *NIPS 2012: Neural Information Processing Systems*, 1–11.

Gilbert, Mark, Karen Shegda, Kenneth Chin, Gavin Tay, and Hanns Koehler-Kruener. 2014. *Magic Quadrant for Enterprise Content Management 2014*. http://www.gartner.com/document/2854918?ref=QuickSearch&sthkw=Enterprise Search&refval=146584442&qid=3383562ca70c380611fb839a06fcab3a.

Gimpel, Kevin, Dipanjan Das, and Noah A Smith. 2009. "Distributed Asynchronous Online Learning for Natural Language Processing." *Computational Linguistics*, 213–22. http://www.aclweb.org/anthology/W10-2925.

Goyal, A, H Daumé III, and S Venkatasubramanian. 2009. "Streaming for Large Scale NLP: Language Modeling." *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 512–20. doi:10.3115/1620754.1620829.

Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. 2009. "The WEKA Data Mining Software." *ACM SIGKDD Explorations* 11: 10–18. doi:10.1145/1656274.1656278.

Marketsandmarkets.com. 2013. *Natural Language Processing (NLP) Market [IVR, OCR, Pattern Recognition, Auto Coding, Text Analytics, Speech Analytics, Machine Translation, Information Extraction, Question Answer, Report Generation] - Worldwide Market Forecast & Analysis (2013–2018)*. doi:TC 1289.

Mvarshney. "Too Slow or out of Memory Problems in Machine Learning." http://stackoverflow.com/questions/15419411/too-slow-or-out-of-memory-problems-in-machine-learning-data-mining.

Porteous, Ian, David Newman, Alexander Ihler, Arthur Asuncion, Padhraic Smyth, and Max Welling. 2008. "Fast Collapsed Gibbs Sampling For Latent Dirichlet Allocation." In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 569. doi:10.1145/1401890.1401960.

Sculley, D, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. *Machine Learning: The High-Interest Credit Card of Technical Debt.* http://static.googleusercontent.com/media/research.google.com/en/us/pubs/archive/43146.pdf.

Shu, Catherine. 2014. "Google Acquires Artificial Intelligence Startup DeepMind For More Than $500M." *TechCrunch*. http://techcrunch.com/2014/01/26/google-deepmind/.

Zhang, Jie, Debra Logan, and Garth Landers. 2014. *Magic Quadrant for eDiscovery Software 2014. Gartner Research.* http://www.gartner.com/document/2773517?ref=QuickSearch&sthkw=eDiscovery&refval=146585495&qid=f9cc7bda72e8d6162ed0e2163075a53d.

Zilis, Shivon. 2014. "Current State of Machine Intelligence." https://medium.com/@shivon/the-current-state-of-machine-intelligence-f76c20db2fe1.

Appendix A – Downpour SGD Client PseudoCode

Source - (Dean et al. 2012)

Algorithm 1.1: DOWNPOURSGDCLIENT($\alpha, n_{fetch}, n_{push}$)

```
procedure STARTASYNCHRONOUSLYFETCHINGPARAMETERS($parameters$)
  $parameters \leftarrow$ GETPARAMETERSFROMPARAMSERVER()

procedure STARTASYNCHRONOUSLYPUSHINGGRADIENTS($accruedgradients$)
  SENDGRADIENTSTOPARAMSERVER($accruedgradients$)
  $accruedgradients \leftarrow 0$

main
  global $parameters, accruedgradients$
  $step \leftarrow 0$
  $accruedgradients \leftarrow 0$
  while $true$
    do
      if ($step$ mod $n_{fetch}$) $== 0$
        then STARTASYNCHRONOUSLYFETCHINGPARAMETERS($parameters$)
      $data \leftarrow$ GETNEXTMINIBATCH()
      $gradient \leftarrow$ COMPUTEGRADIENT($parameters, data$)
      $accruedgradients \leftarrow accruedgradients + gradient$
      $parameters \leftarrow parameters - \alpha * gradient$
      if ($step$ mod $n_{push}$) $== 0$
        then STARTASYNCHRONOUSLYPUSHINGGRADIENTS($accruedgradients$)
      $step \leftarrow step + 1$
```

Appendix B – Sandblaster L-BFGS Pseudocode

Source = (Dean et al. 2012)

Algorithm 1.2: SANDBLASTERLBFGS()

```
procedure REPLICA.PROCESSPORTION($portion$)
  if ($!hasParametersForStep$)
    then $parameters \leftarrow$ GETPARAMETERSFROMPARAMSERVER()
  $data \leftarrow$ GETDATAPORTION($portion$)
  $gradient \leftarrow$ COMPUTEGRADIENT($parameters, data$)
  $localAccruedGradients \leftarrow localAccruedGradients + gradient$

procedure PARAMETERSERVER.PERFORMOPERATION($operation$)
  $PerformOperation$

main
  $step \leftarrow 0$
  while $true$
  do
    comment: PS: ParameterServer
    $PS.accruedgradients \leftarrow 0$
    while ($batchProcessed < batchSize$)
    do
      for all ($modelReplicas$) comment: Loop is parallel and asynchronous
        if ($modelReplicaAvailable$)
          then
            REPLICA.PROCESSPORTION($modelReplica$)
            $batchProcessed \leftarrow batchProcessed + portion$
        if ($modelReplicaWorkDone$ and $timeToSendGradients$)
          then
            SENDGRADIENTS($modelReplica$)
            $PS.accruedGradients \leftarrow PS.accruedGradients + gradient$
    COMPUTELBFGSDIRECTION($PS.Gradients, PS.History, PS.Direction$)
    LINESEARCH($PS.Parameters, PS.Direction$)
    PS.UPDATEPARAMETERS($PS.parameters, PS.accruedGradients$)
    $step \leftarrow step + 1$
```
