Software Engineering Challenges of Deep Learning

Anders Arpteg
Peltarion AB
Stockholm, Sweden
anders@peltarion.com

Björn Brinne
Peltarion AB
Stockholm, Sweden
bjorn@peltarion.com

Luka Crnkovic-Friis
Peltarion AB
Stockholm, Sweden
luka@peltarion.com

Jan Bosch
Peltarion AB
Stockholm, Sweden
jan@peltarion.com

ABSTRACT

Surprisingly promising results have been achieved by deep learning systems in recent years. Many of these achievements have been reached in academic settings, or by large technology companies with highly skilled research groups and advanced supporting infrastructure. For companies without large research groups or advanced infrastructure, building high-quality production-ready systems with deep learning components has proven challenging. There is a clear lack of well-functioning tools and best practices for building deep learning systems. It is the goal of this research to identify what the main challenges are, by applying an interpretive research approach in close collaboration with companies of varying size and type.

A set of eight projects have been selected to describe the potential with this new technology and also to identify what the main challenges are. A set of twelve main challenges has been identified and categorized into the three areas of development, production, and organizational challenges. Furthermore, a mapping between the challenges and the projects is defined, together with selected motivating descriptions of how and why the challenges apply for specific projects.

Compared to other areas such as software engineering or database technologies, it is clear that deep learning is still an immature technology in need of further research and development. The challenges identified in this paper can hopefully be used to guide future research by the software engineering and deep learning communities. Together, we could enable a large number of companies to start taking advantage of the high potential of the deep learning technology.

KEYWORDS

deep learning, machine learning, artificial intelligence, software engineering challenges

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

ACM Reference Format:
Anders Arpteg, Björn Brinne, Luka Crnkovic-Friis, and Jan Bosch. 2018. Software Engineering Challenges of Deep Learning. In *Proceedings of International Conference on Software Engineering (ICSE).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Deep Learning (DL) has received lots of attention in recent years due to its success in areas such as computer vision tasks (e.g., object recognition [31] and image generation [45]) using convolution neural networks (CNN), natural language understanding using recurrent neural networks (RNN) [38] and machine strategic thinking using deep reinforcement learning (DRL) [39]. One of the main differences from traditional machine learning (ML) methods is that deep learning automatically learns how to represent data using multiple layers of abstraction [4, 7]. In traditional machine learning, a significant amount of work has to be spent on "feature engineering" to build this representation manually, but this process can now be automated to a higher degree. Having an automated and data-driven method for learning how to represent data improves both the performance of the model and reduces requirements for manual feature engineering work [24, 32].

Given the recent advances in machine learning, we are also seeing industry starting to increasingly take advantage of these techniques especially in large technology companies such as Google, Apple, and Facebook. Google applies deep learning techniques to the massive amounts of data collected in services such as Google Translator, Android's voice recognition, Google's Street View, and their Search service [28]. Apple's virtual personal assistant Siri offers a variety of services such as weather reports, sports news, and generic question answering by utilizing techniques such as deep learning [16]. As the techniques behind deep learning are becoming increasingly available and easy to use, we are seeing not only large technology companies but also smaller companies in other areas starting to take advantage of deep learning.

1.1 Machine Learning and Software Engineering

Machine learning (ML), and especially deep learning (DL) differs partly from traditional software engineering (SE) in that its behavior is heavily dependent on data from the external world. Indeed, it is in these situations where machine learning becomes useful.

A key difference between ML systems and non-ML systems is that data partly replaces code in an ML system, e.g. when a learning algorithm is used to automatically identify patterns in the data instead of writing hard coded rules. This suggests that data should be tested just as well as code, but there is a lack of best practices for how to do so today. We would like to be able to e.g. compare the distribution of the input data over time, and of the output predictions. This is difficult partly because of the non-deterministic nature of many training algorithms and also because of the dependency to the external world, which can make if difficult to reproduce previous results and compare one distribution to another.

Significant amount of research has been conducted on testing software [30], and also for testing machine learning performance. However, the intersection of SE and ML has not be studied as well [8]. It is not only the correctness of the model that needs testing, but also the implementation of a production-ready ML system [41]. An interesting example is described by Google and their new Quick Access ML system in Google Drive [51]. Even though data was accessed from the same service in training and in serving mode (i.e. making real-time predictions in production), it was accessed through an API in production and in training it was scanning the raw table. The API had some data transformation logic not present in the raw table, which caused a significant performance skew between training and production mode. This is an example of a situation where proper testing is needed, but is challenging to perform today.

Another complicating factor for the development of ML systems is that there may exist culture differences between data scientists and software engineers. For data scientists, too much focus may be placed on model accuracy and potentially disregarding the complexity and code quality of building production-ready systems and supporting data pipelines.

ML systems also lack transparency that makes it hard to understand how a specific part of a system will behave. The degree to which this is true depends on the machine learning method in question. At one end we have algorithms such as logistic regression and Bayesian inference that are rather well understood. DL is at the other extreme end with powerful but complex and poorly understood models [21]. Given the data dependency, it may be insufficient to write tests that mainly verify the functionality of the code. To phrase this in Software Engineering terminology, code tests can only find code bugs, not “data bugs”.

1.2 Big Data and Deep Learning

When deep learning is combined with the recent growth of big data, opportunities and transformative potential exist for sectors such as finance, healthcare, manufacturing, and educational services [11, 34, 36, 43, 49]. In 2011, digital information had grown nine times in volume in just five years and by 2020, its amount in the world will reach 35 trillion gigabytes [19]. The combination of deep learning and big data is considered as “big deals and the bases for an American innovation and economic revolution” [13].

In 2012, the Obama Administration announced a “Big Data Research and Development Initiative” to “help solve some of the Nation’s most pressing challenges” [25]. Consequently, six Federal departments and agencies (NSF, HHS/NIH, DOD, DOE, DARPA, and USGS) committed more than $200 million to support projects that can transform our ability to harness volumes of digital data in novel ways. In May the same year, the state of Massachusetts announced the Massachusetts Big Data Initiative that funds a variety of research institutions [23].

Working with big data adds extra requirements to supporting infrastructure. Instead of simply loading all data in memory on a single machine, special infrastructure may be needed such as Apache Spark [56], Apache Flink [9], or Google DataFlow [2]. Big data is often loosely defined by the three Vs: Volume, Variety, and Velocity [5]. It is not only the volume of data that may require big data techniques, but also the variety of different types and formats of data. This is another strength of deep learning, which can use representation learning to combine different modalities such as images, audio, text, and tabular data. For example, Ngiam et al [42] demonstrated how deep learning techniques could be used to integrate audio and video data by learning an internal representation for each modality. This has also been demonstrated by Google more recently [29]. Given these two internal representations, they could also show how the model could combine features from both modalities in another layer of abstraction. However, that also presents challenges in how to integrate and transform multiple data sources [10]. Given the large number of different formats for images, video, text, and last but not least tabular data, it can be challenging to build a data integration pipeline. Additionally, velocity requirements (i.e. processing time or real-time low latency demands) may also require use of big data techniques such as streaming processing.

This adds numerous challenges to the task of building a big data deep learning system. It is not only the process of extraction, transforming and loading (ETL) data but also that novel distributed training algorithms need to be used. Especially deep learning techniques are not trivially parallelized and again needs special supporting infrastructure. An example is DistBelief [14] that demonstrated training of models with billions of parameters and thousands of CPU cores. They make use of centralized parameter servers and a large number of worker machines that each work on shards of the data. There are other examples that combine distributed deep learning systems with big data systems, such as TensorFlowOnSpark [55].

1.3 Technical Debt

Technical debt, a metaphor introduced by Ward Cunningham in 1992, can be used to account for long term costs incurred by moving quickly in software engineering. Traditional methods used to handle technical debt include refactoring code, improving unit tests, deleting dead code, reducing dependencies, and improving documentation [18]. It has been argued that ML systems have a special capacity for incurring technical debt [47]. Due to the dependency to the external world, ML systems not only have the code level debts but also issues related to changes to the external world.

Data dependencies have been found to build similar debt as code dependencies. For example, consider the case where one data source was not properly normalized. The consumer could have been training on poorly normalized data, but once a bug-fix for the normalization is deployed, the consuming model stops working.

Few tools exists today for evaluating and analyzing data dependencies, especially when compared to static analysis of code using compilers and build systems. One example of a data dependency tool is described in [37], where data sources and features can be annotated.

Being able to reduce technical debt in ML-systems requires understanding of all these problems and that teams are rewarded for being able to build a successful high accuracy ML-system with low technical debt that can be maintained in a sustainable way.

Deep Learning also makes it possible to compose complex models from a set of sub models and potentially reuse pre-trained parameters with so called "transfer learning". This adds additional dependencies on not only the data, but also on external models that may be trained separately and may also change in configuration over time. Additionally, as the internal model may be continuously retrained and updated, having dependencies to external pre-trained models can introduce additional debt that also might degrade overall performance over time. External models add additional technical debt and cost of ownership for ML systems. Dependency debt is noted as one of the key contributors to technical debt in SE [40].

Another challenge with ML systems is when unintended feedback loops exist. To illustrate this problem in a media streaming context, imagine a media recommendation system using collaborative filtering techniques working together with a search system where the user can find new content. The recommender uses historical data such as media played by the user, and the search system tries to identify relevant content. If the functionality of the search system is changed by e.g. creating personalized search results, this will also change the media played by the user. Since the recommender makes use of historical data about what content the user has played, changes in the search functionality also change the behaviour of the recommender system, and vice versa.

When moving an ML system from an initial prototype or from an academic implementation into a production-ready version in an industrial setting, significant amount of work has to be invested in the infrastructure surrounding the ML system. Using the terminology of [35], the code needed to support ML systems can be referred to as "plumbing" and can be significantly larger than the core ML code itself.

It is not uncommon that the supporting code and infrastructure incur significant technical debt, and care should be taken to properly plan for time needed to develop plumbing code. Few high quality tools exists today, but the growth of cloud services and ML platforms are increasing and it is important to carefully choose suitable infrastructure and platforms. This can not only reduce the time needed to implement a production-ready ML system, but also reduce technical debt.

A mature system might end up with 95 % of the code being plumbing and glue code, which mainly connects different ML libraries and packages together [47]. For large companies, it is not uncommon that hundreds or thousands different data pipelines have been developed. An ML system is usually dependent on many different pipelines that may also be implemented in different languages, formats, and infrastructure systems. It is not uncommon for pipelines to change, add and remove fields, or become deprecated. Keeping a deployed production-ready ML system up to date with all these changes can require significant amount of work, and requires supporting monitoring and logging systems to be able to detect when these changes occur.

Another common bad practice in ML systems is to have experimental code paths. This is similar to *dead flags* in traditional software [40]. Over time, these accumulated code paths create a growing debt that will be difficult to maintain [47]. An infamous example of experimental code paths was Knight Capital's system losing $ 465 million in 45 minutes, apparently because of unexpected behavior from obsolete experimental code paths [48].

If we compare ML systems with database systems, it becomes clear that there is a lack of a well-defined abstraction layer in ML systems [58]. Having a declarative interface for how to describe input, output, transformations, training, and predictions in as precise a way as for database systems could provide a significant improvement in abilities for libraries and packages to interoperate. The maturity of ML systems, and especially DL systems, is today far behind that of database systems.

Configuration of ML systems are frequently not given the same level of quality control as software code, even though it can both have an higher impact on the performance of the model and also be large in size [58]. Configuration changes should be reviewed and possibly even tested in the same way as code changes.

Making it easy to make incremental changes and compare configuration to previous experiments is essential to be able to build a high performing ML system in a scientific way. Once again, having a clear abstraction layer would make it easier to write and compare configuration between different models and experiments.

The contribution of this paper is threefold. First, it presents eight cases of machine learning systems for varying use cases and we identify the potential of machine learning and specifically deep learning as a technology. Second, we identify the key software engineering challenges associated with building deep learning systems. Finally, we validate the relevance of these software engineering challenges by providing an overview of the case study systems in which we have experienced these challenges.

The rest of the paper is organized as follows. After the introduction, which provides background information and outlines some of the challenges in the intersection of SE and ML, the research approach have been described. A set of selected real-world projects have then been described, followed by a set of identified main challenges. The projects have also been mapped to identified challenges, together with motivating text of how and where the challenges apply. Finally, the related work section gives additional context and the conclusion section summarizes the findings.

2 RESEARCH APPROACH

The findings in this paper are based on eight case studies that have been carried out in close collaboration with companies of different sizes and types. The companies range from startup size to large multinational companies with thousands of employees and many millions of active users. The case studies are focused on ML projects of different types, some projects have been in production for more than 10 years, starting from 2005 and forward, and other projects are just in prototype stage. Experience is collected from people directly employed in associated organizations and working with given case studies, taking a participant observer approach and making use of well-established case study methods [17].

The focus of this study is limited to identify challenges specifically related to the intersection of software engineering practices and deep learning applications. The challenges are based on existing research and validated by empirical studies of selected case studies, and the study adopts an interpretive research approach [52]. Software development is seen as an activity conducted by people in organizations with different values, different expectations, and with different strategies. This type of case study research is appropriate to explore real-life challenges where control over the context is not possible and where there is an interest in accessing people's experiences related to case studies and associated organizational structures [52].

3 REAL-WORLD PROJECTS

The set of real-world projects that has been used in this research is described in this section. The projects have been selected so as to collectively represent and exemplify different aspects of challenges, and a mapping between the projects and challenges is presented in a later section.

3.1 Project EST: Real Estate Valuation

Real estate valuation is a cornerstone of financial security for banks. When issuing loans, they need to make sure that the sales price is reasonably close to the market value and they need to know the total value of their collateral securities. Project EST is a long running (> 10 years) neural network based real estate valuation system. It is used in production by multiple Swedish banks.

The data used by the network was historical sales, information about individual properties, demographic information and geographic information. It was developed by a small team of two data scientists and two backend developers. It tied into a pre-existing SQL database system and a large collection of legacy data pipeline scripts running on assorted systems.

3.2 Project BOO: Automated Bookkeeping

Accounting is today a manual process that requires many man hours each month for most companies. It is also an inaccurate process sensitive to human error. Project BOO was started by a vertical AI company that aims to automate bookkeeping with the goal of automatically reading images of receipts. The lack of labelled data led to the development of a photo-realistic receipt generator to create synthetic data.

The initial team consisted of 4 data scientists backed up by a development team of 10 people. The development started from scratch, with no legacy code. Although initial results were promising, the lack of tooling for training, version controlling and deploying deep learning models made large scale deployment and maintenance too costly.

3.3 Project OIL: Predicting Oil and Gas Recovery Potential

When building multi-stage hydraulically fractured horizontal wells in tight gas and oil reservoirs, the main question of importance is how much oil and gas will be ultimately recovered. Many different methods have been proposed and used to get a figure - the Estimated Ultimate Recovery (EUR). Project OIL started by a US oil and gas exploration company investigated the use of deep learning in predicting the EUR based on geological information.

The data consisted of high resolution geological maps combined with a small number of oil/gas wells that had been in production for a while (and where you could deduce the EUR accurately). The project resulted in a decision support tool that could given a coordinate in the Eagle Ford Shale do an EUR prediction for an average multi-stage hydraulically fractured horizontal well. The development team consisted of two senior data scientists/developers.

3.4 Project RET: Predicting User Retention

An important metric for any end-user facing company is user retention, e.g. what percentage of active users will remain active after 28 days or how many users will be active two weeks after registration. Especially the latter question, trying to estimate the percentage of users that will remain active shortly after registration is a difficult given the small amount of available data. Project RET refers to a set of projects built by different teams for this purpose. Even though the media streaming service may have many millions of active users, being able to predict second week retention given only few days of data per user is non-trivial.

A number of data sources were used such as media consumption, client interactions (e.g. mobile application actions), and user demographics. Assuming that there are many millions of active users and that data has to be extracted from multiple data sources, this becomes a big data problem with many terabytes to process. Multiple teams are working with this problem, often for a specific functionality of the service in mind. Each team usually consists of 4 - 10 data scientists and engineers. Predictions are usually tracked in dashboards and can be used to evaluate for example randomized tests of product changes.

3.5 Project WEA: Weather Forecasting

Weather forecasts today relay on physical modelling and are solved primarily using finite element methods. They require very expensive supercomputers, are slow and for many applications inaccurate. Project WEA, performed in collaboration with a national meteorological agency, aimed to use deep learning to do weather forecasts. The ultimate purpose was building better wind turbine generator predictions.

The data, which was large (>1 TB) consisted of satellite images, meteorological observations and topological maps. The data spanned a time period of 12 years. Some of it was easily obtainable while other parts required the physical transport of data tapes. The project team consisted of 3 data scientists, 2 developers and a number of meteorologists.

3.6 Project CCF: Credit Card Fraud Detection

For companies in the gaming industry, credit card fraud can be a big issue due to the possibility of money laundry. If you deposit money with a stolen credit card you can withdraw "winnings" from another account after losing deliberately to yourself or after turning the money round a few times in casino style games. Preventing or detecting fraudulent deposits early on by combining payment details with activities in the games is crucial.

Project CCF was started with the purpose of building a generalized model for this purpose. Significant time was spent on improving the model efficiency of fraud detection and in that way also reducing levels of blocked, risky payments in a company where the detection was based on hand-crafted rule sets. The data consisted of payment request data (method, amount location etc) and customer details (CRM, payment history, activity history). If the payment was approved subsequent analysis also took into account post payment activity such as signs of abnormal play. Responsible for developing the models was a small dedicated analysis team and results were handed over to a different department for integration into the payment processing systems.

3.7 Project BOT: Poker Bot Identification

Online poker grew very quickly in the early 2000s. With a lot of inexperienced players waging real money there was an opportunity to win if you were a good player. However the stakes were quite low on the tables favoured by inexperienced players so you would have to play a lot of games to make a reasonable profit. That means scaling the number of games by automating play using software was attempted by many and successful for some. For the game sites, it was and is a question of customers' trust to keep the games free from these "bots". Project BOT was initiated to develop methods for detecting these bots, and to be able to quickly lock their accounts.

The data used was game play activity (statistics on actions taken given state of the game as seen from each player's perspective) together with game client technical details such as connection ip, hardware fingerprints, player clicking/timing statistics. The models were developed by a small consultancy team (2-3 people) together with internal experts (1-2 people) on part time. Despite promising results the project was cancelled before it was finished due to unpredictability of efficiency of finished product.

3.8 Project REC: Media Recommendations

Recommendation systems is a popular field that has increased in importance with media streaming services. Having a high quality recommendation service that can not only understand the users' preference but also specific user contexts can significantly improve the user experience. Traditional techniques such as collaborative filtering works well given that sufficient data exists for both the user and the content. However, it can become problematic when new content is released such as a new movie or song that few users have seen. Being able to quickly recommend new content is important, as users are interested in novel content.

Project REC refers to the work with solving the problem of being able to recommend new content. Using techniques such as deep learning and convolutional networks, it is possible to use e.g. raw media data to learn the user preferences for novel content without having any user data. Having a hybrid system that makes use of collaborative filtering techniques when sufficient user data exist, and deep learning for novel content, brings the best of two worlds. This deep learning project started as a small prototype project, that later was built out as a full production-ready system by a team of data scientists and a large number of engineers.

4 SELECTED CHALLENGES

A number of challenges and examples were presented in the Introduction section. To get a better overview of the challenges in the intersection between machine learning and software engineering, this section presents a list of concisely described challenges. They have been grouped into the three categories: development, production, and organizational challenges.

4.1 Development Challenges

Developing an ML system is fundamentally different from a "normal" SW project in that you leave much of the responsibility for finding a working solution to the computer. It becomes more similar to a research project and is not necessarily easy to split up into predictable modules/blocks. The lack of model transparency and reproducibility makes it hard to identify issues and chase bugs, which in turn makes the effort required to complete the project even more unpredictable.

4.1.1 Experiment Management. During the development of ML models, a large number of experiments are usually performed to identify the optimal model. Each experiment can differ from other experiments in a number of ways and to have a reproducible results, it may be necessary to know the exact version of components such as:

(1) Hardware (e.g. GPU models primarily)
(2) Platform (e.g. operating system and installed packages)
(3) Source code (e.g. model training and pre-processing)
(4) Configuration (e.g. model configuration and pre-processing settings)
(5) Data sources (e.g. input signals and target values)
(6) Training state (e.g. versions of trained model).

Traditional management of software is usually light weight as you only need to track source code files, and their dependencies. Version control for ML is more heavy weight and branches out more, especially given the high level of data dependency common in ML systems. Different versions of data will yield different results, and the input data is often a conglomerate of data from multiple heterogeneous data sources. It has been argued that one of the most difficult dimensions to keep track of is the data dimension, and the cost and storing of versioned data can be very high [40].

Futhermore, different versions of the model are created during training of the deep learning model, each with different parameters and metrics that need to be properly measured and tracked. With the addition of data dependencies and a high degree of configuration parameters, it can be very challenging to properly maintain ML systems in the long run. Also, it is not uncommon to perform hyperparameter tuning of models, potentially by making use of automated meta-optimization methods that generate hundreds of versions of the same data and model but with different configuration parameters [22]. Deep learning can also add the requirement of specific hardware, on top of software. For example, switching to a new GPU version with only integer instead of floating point computations can lead to different results.

It is essential to keep track of these versions and metrics, to be able to compare and reproduce experiments in a scientific and data-driven way.

4.1.2 Limited Transparency. Software engineering is based on the principle of reducing a complex system into smaller, simpler blocks. Whenever possible, it is desirable to group blocks into different levels of abstraction that have similar conceptual meaning.

Although deep learning systems essentially do that automatically, it is very difficult to know exactly how it is performed or predict what the abstraction layers will look like once the model has been trained. Furthermore, it is difficult to isolate a specific functional area or obtaining a semantic understanding of the model. This can only be performed with approximated methods [3].

The great advances that have been made in fields such as computer vision and speech recognition, have been accomplished by replacing a modular processing pipeline with one large neural network that is trained end-to-end [33]. In essence, transparency is traded for accuracy. This is an unavoidable reality, and if the problem that you want to solve is simple enough to be explained in symbolic logic, then you don't need a complex system to solve it. However, if the problem is complex and the model is inherently irreducible then an accurate explanation of the model will be as complex as the model itself [21].

4.1.3 Troubleshooting. A major challenge in developing deep learning systems is the difficulty to estimate the results before a system has been trained and tested. Furthermore, our poor understanding of the inner working of a complex neural network makes it difficult to debug them in a traditional way. In a neural network, the structure combines the functional parts and the memory and can also be distributed across multiple machines. Furthermore, using libraries such as TensorFlow [1] potentially combined with big data frameworks such as Apache Spark [56] makes it difficult to troubleshoot and debug problems in the code. As they both have a lazy execution graph, where the code is not executed in imperative order of the code, it can be very difficult to troubleshoot bugs using traditional software engineering tools. Other frameworks such as PyTorch [12] do not have the lazy execution graph problem but have other issues such as less mature code quality.

Even if it was possible to step through the source code or set a break point, manual evaluation is in practice often impossible as it would involve the inspection of millions of parameters. Compared to traditional software engineering, a small bug in the source code may not be detected at neither compile- nor run-time. During training, the only information that the developer or data scientist may have is a global error estimate. As large neural networks usually take days and often weeks to train and there is no way of guaranteeing that a certain performance level will ever be reached, the effort required to build them can become very large [10].

4.1.4 Resource Limitations. As mentioned in the introduction, working with data that requires distributed system adds another magnitude of complexity compared to single machine solutions. It is not only the volume of the data that may require a distributed solution, but also computational needs for extracting and transforming data, training and evaluating the model, and/or serving the model in production. For deep learning systems, it can also be limited GPU memory that requires special techniques to split the model across multiple GPUs.

Working with distributed systems, both for data processing such as Apache Spark [56] and deep learning training such as Distributed TensorFlow or TensorFlowOnSpark [1, 55], add complexity in a number of dimensions. It requires not only additional knowledge and time to operate them, but also additional management and cost of associated hardware and software.

Another significant challenge introduced by making use of distributed systems is increased difficulty in troubleshooting problems. We are also seeing an increased use of low latency streaming systems in production, which are even more immature than corresponding batch systems.

4.1.5 Testing. Machine learning systems requires testing of software used for building data pipelines, training models, and serving in production. Given the high data dependency of ML systems, data also needs to be tested. However, few data testing tools exist today especially compared to software testing. A frequent pattern seen when testing data is to make use of a small sample of the full dataset. It is challenging to provide a sample that include all the edge cases that may exist in the full dataset. Also, as the external world is dynamic and changes over time, new edge cases will continue to appear later in time.

As mentioned in the introduction, the non-deterministic nature of many training algorithms makes testing of models even more challenging. Another challenge can be the data processing and model serving in production mode can differ in implementation compared to training or testing mode, causing a training-serving skew in model performance [51]. Having the proper tests in place then becomes crucial.

4.2 Production Challenges

Particularly for deep learning, it is important to take advantage of the newest hardware. With this follows more frequent updates to state-of-the-art software and managing dependencies becomes a big issue. Making sure you detect problems introduced by changing behaviour of dependencies - including data sources that have been modified - requires careful and clever monitoring. More interestingly, since an ML system is often used to try to influence users the model might change the reality it tries to understand, hence creating a feedback loop.

4.2.1 Dependency Management. Traditional software engineering typically builds on the assumption that hardware is at best a non-issue and at worst a static entity that has to be taken into consideration. Deep learning systems are primarily trained on GPUs as they provide a 40-100x speedup over classic CPUs. The market is currently dominated by NVIDIA's hardware and CUDA computation platform. For the past 5 years significantly improved and changed GPUs are released 1-2 times per year.

As each step increases the performance significantly (beyond Moore's law) [20] and hardware performance directly translates to reduced training time (or better results for equivalent training time) there is a great incentive for the software to tightly follow the hardware development.

The deep learning software platforms are continuously updated on a weekly and sometime daily basis and the updates typically result in noticeable improvements. This works well for academic research and for developing proofs of concept but can cause considerable issues for production-ready systems. Unlike other machine learning methods, deep learning often scales directly with model size and data amount. As training times can be very long (typically a few days or up to a week) there is a very strong motivation to maximize performance by using the latest software and hardware.

Changing the hardware and software may not only cause issues with being able to maintain reproducible results, they may also incur significant engineering costs with keeping software and hardware up to date.

4.2.2 *Monitoring and Logging.*

Building a toy example ML system or even an offline research prototype is easy compared to amount of work required to build a production-ready ML system [8]. In real-world ML applications beyond toy examples, it can become very difficult to cover all the edge cases that may occur once a model has been deployed in production. It is also common that people fail to recognize the effort needed to maintain a deployed ML system over time [47].

An ML system may be retrained frequently and thus change behavior autonomously. As the behavior of the external world changes, the behavior of the ML system can suddenly change without any human action in "control" of the system. In this situation, unit testing and integration tests are valuable but not sufficient to validate the performance of the system. Old thresholds that may have been manually assigned may no longer be valid given drifts in the data from the external world.

Live monitoring of the system performance can help, but it may not be obvious what metrics to monitor. It may not be sufficient to monitor accuracy metrics of validation and test datasets, but also metrics such as distribution of the predictions, and also because of input data and up-stream data providers. As an example from a large media streaming company, the monitoring of an ML system showed no signs of any issues even though one of the upstream datasets had become deprecated. The system had a built-in fallback to revert back to older versions of the data in case current day was missing. This works well for temporary issues when just one or two days of data are missing, but when six months of data is missing, the system is not performing at a satisfactory level. It took more than six months before this dependency problem was detected and fixed. With additional monitoring of the upstream datasets, this issue could have been detected much earlier.

4.2.3 *Unintended Feedback Loops.*

A machine learning system is by definition always open ended as it is driven by external data. In practice this means that when dealing with complex models, there is no way of separating the invariant parts of the model from the ones formed by the data. No matter how carefully you construct the model and test the code that defines it, its final performance will always be heavily dependent on the external data [47].

Furthermore, especially in models deployed in a big data context (as machine learning systems often are), there is a risk of creating unintended feedback loop where over time you adapt your real-world system to your model rather than the other way around. Imagine having a widely used real estate price prediction system. When such as system becomes sufficiently popular, the predictions can easily become a self-fulfilling prophecy. As known from control theory, apart from controlled special cases, feedback loops are inherently unstable [6].

4.2.4 *Glue Code and Supporting Systems.*

An unfortunate property of ML systems, and especially deep learning systems, is that only a small part of the system deals with the model. In a production-ready system, only 5 % of the code may deal with the model and the rest is "glue code" that interacts with supporting systems and glues all libraries and systems together [47]. Also, as the number and the efficiency of different cloud providers continue to increase, an increasing percentage of the code interacts with external systems that is out of the control of the designer.

Keeping the glue code up to date, and keeping up with changes in cloud services, can introduce unexpected challenges in a product-ready system. That said, the benefits of making use of cloud services should not be understated. By not having to build and manage all services yourself, significant reductions in time to production can be achieved. However, it also means less control over external systems and it can introduce unexpected work with glue code and supporting systems.

4.3 Organizational Challenges

To put an ML model into production usually requires collaboration between many different teams with very different ideas of what is important and with quite different expectations on outcomes. A data scientist might be "pragmatic" about their code as long as it achieves desired results in a controlled environment, whereas members of the engineering teams care a lot more about maintainability and stability. Since creating an ML model is in many respects a research project, it is difficult to plan and to fit it into a long-term company roadmap. It requires a predictability that is often not possible to have in ML projects.

4.3.1 *Effort Estimation.*

The reductionist modular SE design of a non ML project makes it a lot easier to estimate the time and resources required to complete it. In an ML project, the goal might also be well defined but it is unclear to what extent a learned model will achieve that goal, and an unknown number of iterations will be needed before results reach acceptable levels. This is in the nature of any research project. It is also usually not possible to decrease scope and run the project in a time based setting with a predefined delivery date since you might not have achieved even close to acceptable results by then.

Instead, an ML project will be defined by set scope, i.e. model efficiency goals. An added complication is the lack of transparency inherent in many ML models. Algorithms such as logistic regression and Bayesian inference are rather well-understood. DL is at the other extreme end with powerful but complex and poorly understood models [21]. Since there is in many cases no easy way to understand how a model works it is also very difficult to understand how it should be modified to reach better results.

4.3.2 *Privacy and Safety.*

The lack of understanding of the internal workings of a large neural network can have serious implications for privacy and safety. The knowledge in a neural network is stored in a distributed way across the weights of the network. Although we know that specialization occur in specific regions of the neural network, its exact mechanism is poorly understood. Thus, it is very difficult for designers to control where and how information is stored. It also not uncommon for companies to have

Table 1: Challenges faced in selected projects

Category	Challenge	EST	BOO	OIL	RET	WEA	CCF	BOT	REC
Dev	Experiment Management	X	X	X	X	X		X	X
	Limited Transparency	X		X		X	X		X
	Troubleshooting	X			X	X			X
	Resource Limitations	X	X		X	X			X
	Testing	X	X	X	X	X	X	X	X
Prod	Dependency Management	X	X		X	X	X		X
	Monitoring and Logging				X	-	-	-	
	Unintended Feedback Loops	X							X
	Glue Code and Infrastructure	X			X	X			X
Org	Effort Estimation	X	X		X		X	X	X
	Privacy and Safety	X	X				X	X	
	Cultural Differences	X	X	X	X		X	X	

"X": the project has clearly experienced associated challenge
"-": the challenge is not applicable for associated project

terms-of-service agreements with their end users that prevents them from using raw data as direct input to a machine learning model, but instead have to make use of anonymized and/or aggregated statistics of the user data [51]. This can not only reduce the performance on the model, but also make tasks such as data exploration and troubleshooting problems more difficult.

Although the information in a model is obscured and there is no trivial way of transforming it back to humanly readable information, it is not impossible to do so and there it is difficult to protect sensitive information. There has been some work to preserve the safety and privacy of data, e.g. differential privacy [15], k-anonymity [50], and encrypted networks that make use of homomorphic encryption to keep sensitive data safe [54]. However, more work is needed to preserve privacy and safety of sensitive datasets while still being able to efficiently perform data exploration, develop models, and troubleshoot problems.

4.3.3 Cultural Differences. Building a production-ready ML system usually involves a collaboration between people with different roles. An initial prototype may be built by data scientists that focus mainly on model performance and building a working proof-of-concept. Transforming a prototype into a production-ready system that also interacts with existing backend and frontend systems usually requires a significantly larger effort. This normally includes collaboration with for example backend engineers, UX designers, and product owners.

It is not uncommon that the culture, skills, and interest areas differs between these people. For example, data scientists may be lacking in software engineering skills and understanding of good engineering practices. UX designers that have a good understanding of how to optimize the user experience, may also have a different culture and ways of working that can introduce challenges in working together to develop a production-ready ML system [51].

Another challenge for larger organizations can be the physical and organizational structure, i.e. collaboration between people working in different teams that may also be physically seated in different locations can introduce challenges. For example, data scientists and data engineers may be working in a centralized structure or they may be embedded in cross-functional teams, and there can be advantages and disadvantages with both approaches.

5 PROJECT CHALLENGE MAPPINGS

Given selected projects and identified challenges, a mapping between the two is provided in table 1. The top row in the table lists the three letter acronyms for respective project, e.g. EST refers to "Project EST: Real Estate Valuation". An "X" means that the project has clearly experienced the specific challenge. A "-" (dash) means that the challenge is not applicable for the specific project, e.g. because the project was never deployed. To further motivate and explain the mappings, see respective motivating text below. Due to the limited space in this paper, we are unable to describe every mapping in the table, but motivating highlights for how projects are related to challenges are provided.

5.1 Experiment Management

As with most ML projects, conducting some form of experiment is crucial. However, some projects conduct more experiments than other, and problems may occur when trying to compare not only the results but also the reasons for some experiment outperforming other experiments. For example, in the Real Estate Valuation (EST) project, a large number of experiments were conducted, and the code behind pre-processing, training, and evaluation of the models was continuously improved.

As common during software development, a number of code refactorings were also performed to improve the quality of the source code without changing the functionality. During one of these code refactorings, a shuffle flag was accidentally switched from false to true. Although it is in many situations a common practice to shuffle training data, this caused a significant decrease of the model performance. Since it happened accidentally during a code refactoring, the obvious but erroneous conclusion was that it was something in the refactoring that caused the performance drop. Several days of work was lost to identify and resolve this shuffle problem. This illustrates one of the challenges with understanding differences between experiments, just understanding potential code modifications between experiments is hard, especially if they influenced the model performance or not. Being able to efficiently compare software, platform environment, configurations, and potential changes in the data is a significant and very important challenge.

5.2 Limited Transparency

In the OIL project, a number of very expensive engineering decisions had to be made based on the results. Since the system was not perfect and the data could be flawed the engineers responsible for the exploration boring wanted to know why the neural network made a certain decision. Given that it is a nearly black box model, no direct simple explanation could be made. Sensitivity analysis was used to estimate the impact of the various geological parameters, but that could not take into account when data was locally of poor quality. Subsequently the engineers were highly reluctant to risk drilling a bore hole (cost > $ 1M) without additional corroboration.

5.3 Troubleshooting

One of the principal components of the weather prediction system (WEA) was an autoencoder that compressed weather data. Early in the testing, the resolution of the reconstructed output was of very poor quality. There was no immediately obvious explanation and several hundreds of experiments with different neural network models was done over a period of two weeks.

Because of the lack of any usable debugging tools for deep neural networks, it took a long time to identify the cause of the error. It turned out to be a pooling (subsampling) operation that was too aggressive leading to a loss of resolution before the data was actually encoded.

5.4 Resource Limitations

There can be many reasons for need of special techniques to handle resource limitations such as lacking memory size (CPU or GPU), too long training time, or low-latency serving needs. The most common reason is probably the need for distributed data processing, to preprocess and prepare the data before training the model.

For example, the media recommendations service has both a collaborative filtering part, and a deep learning part. The deep learning part learns how to recommend novel media that have never or a only a few times been seen by a user, and needs to process raw media data. This involves processing data at petabyte scale, and also train a rather computationally expensive convolutional neural network. This resource demands are extensive, involving many days processing on hundreds of machines. This is an example of having both a big data processing challenge and distributed deep learning training problem.

5.5 Effort Estimation

Although the goals of an ML project can be well-defined, there is no way of guaranteeing when a model achieving the desired goal will be reached. Because it is very difficult to predict what model structure and parameter settings will lead to good results you try theories out by running a series of experiments until you reach a good enough solution. During the development of Project BOT the business owners grew impatient and decided to cancel it even though it had proceeded quite well according to the ML team. Until you have a working model it is very hard to claim you have accomplished anything of value for the client, and being unable to set a final delivery date their process for evaluating projects may lead them to shut it down despite promising intermediate results.

5.6 Privacy and Safety

With machine learning and deep learning specifically, the business value of various data modalities (e.g. tabular, text, or images) continue to increase. With increased use of data, there is also a high risk of an increased abuse of data. Also, we are seeing an increase in regulation regarding use of data, for example in upcoming European General Data Protection Regulation [26].

This adds challenges for companies to ensure the privacy of the users, as for example in the User Retention project. As part of the terms of service agreement (ToS) with the users of the service, user data needs to be anonymized and encrypted. This includes any information that might be used to reverse engineer the identify of the user, e.g. gender, age, and city. Even though this information would be useful to have and could improve the performance of the model, it is encrypted and can be difficult to make use of when building the system.

5.7 Cultural Differences

As mentioned in the description of the challenge 4.3.3, there are many examples of how people with different skills and culture needs to interact. One example of interacting between data scientists and product owners can be found in the User Retention project. One purpose of being able to predict second week user retention in this project, is to be able to compare user experience in randomized tests, i.e. A/B tests.

A product owner has many responsibilities, including being able to push changes to production in time. As it may take time to e.g. implement sufficient logging and to analyze the results, a product owner may be unwilling to spend sufficient time implementing logging. Without proper logging in place, there will not be sufficient data to make accurate retention predictions. This can result in long discussions between data scientists and product owners, that may not only have different goals but also different mindsets in how decisions should be made.

6 RELATED WORK

A number of dimensions have been explored to understand the challenges in ML systems. Aspects and risk factors of hidden technical debt for ML systems were explored in [47], including risk factors such as boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, configuration issues, and changes in the external world.

Traditional software engineering practices has shown the value of clear abstraction boundaries using e.g. encapsulation and modular design. However, ML systems with complex models can erode these boundaries partly due to the dependency to the external world. ML systems also tend to entangle signals and parameters together. The CACE principle states that Changing Anything Changes Everything. A model is sensitive to not only changes in data from the external world, but also e.g. hyperparameters for the model such as number of layers, learning rate, and regularization parameters.

With the addition of transfer learning where complex models are composed of sub models, this can lead to “correction cascades” where a small improvements in a dependency may lead to a decrease in performance of the consuming system.

The problem of having undeclared consumers of ML systems, where consumers silently use the output of the system, is similar to what is referred to as “visibility debt” in traditional SE [40]. This tight undeclared coupling of systems can lead to use in ways that are intended and poorly understood. Additionally, these undeclared consumers can introduce hidden feedback loops between the systems.

A scoring system for production-readiness of ML systems was proposed in [8]. Scoring 0 points means it is more of a research prototype than a production-ready system. Score 12+ points means there are exceptional levels of automated testing and monitoring. They focus on the ML specific tests needed for a production-ready system such as:

(1) Tests for features and data, including distributions of the data, correlation between features, and that relevant features are produced both in training and serving mode.
(2) Tests for model development, that code review is performed, impact of tunable hyperparameters, and performance against a simpler baseline model.
(3) Tests for ML infrastructure, including reproducibility of training, integration tests of the full ML pipeline, and that models can be rolled back to previous version in production.
(4) Tests for ML monitoring, including upstream feature instability, training and service features computes same values, and training and serving latency and throughput.

ML applications are highly data-driven, and another field that is also highly data-driven is the database field. Wang et. al reviews challenges and opportunities with combining the fields of deep learning and databases [46, 53]. As the database community has rich experience working with system optimization, the deep learning community could benefit from taking advantage of database techniques to accelerate training speed. Especially the fields of distributed computing and memory management are central to both the database and deep learning community [1, 14, 27].

Another field closely related to machine learning and deep learning is that of big data. Chen et. al review challenges and perspectives in the intersection of these two fields [10]. The area of big data offers opportunities but also significant engineering challenges.

7 CONCLUSIONS

As we have seen in recent years, companies such as Google, Microsoft, and Facebook are considering artificial intelligence and specifically deep learning to become a standard component of their products across the line [44, 57]. However, there are many challenges with building production-ready systems with deep learning components, especially if the company does not have a large research group and highly developed supporting infrastructure.

The main goal of this research is to identify and outline main software engineering challenges with building systems with deep learning components. Eight projects were described to exemplify the potential of making use of the machine learning and specifically the deep learning technology. For these projects, the main problematic areas and issues with developing these systems were identified. To clarify these problematic areas in more detail, a set of twelve challenges were identified and described in the areas of: development, production, and organizational challenges. The main focus of this paper is not to provide solutions but to outline problem areas and in that way help guide future research. In addition, the outlined challenges also provide guidance on potential problem areas for companies interested in building high quality deep learning systems.

One clear conclusion of this work is that although the deep learning technology has achieved very promising results, there is still a significant need of further research into and development in how to easily and efficiently build high-quality deep learning systems. Traditional software engineering has high quality tools and practices for reviewing, writing test, and debugging code. However, they are rarely sufficient for building production-ready systems containing deep learning components. If the software engineering community, together with the deep learning community, could make an effort finding solutions to these challenges, the power of the deep learning technology could be made available not only to researchers and large technology companies but the vast majority of companies around the world.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[2] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, et al. 2015. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1792–1803.

[3] Yoshua Bengio. 2012. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. 17–36.

[4] Yoshua Bengio and Samy Bengio. 2000. Modeling high-dimensional discrete data with multi-layer neural networks. In *Advances in Neural Information Processing Systems*. 400–406.

[5] Mark A Beyer and Douglas Laney. 2012. The importance of 'big data': a definition. *Stamford, CT: Gartner* (2012), 2014–2018.

[6] Léon Bottou, Jonas Peters, Joaquin Quiñonero-Candela, Denis X Charles, D Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Simard, and Ed Snelson. 2013. Counterfactual reasoning and learning systems: The example of computational advertising. *The Journal of Machine Learning Research* 14, 1 (2013), 3207–3260.

[7] Y-lan Boureau, Yann L Cun, et al. 2008. Sparse feature learning for deep belief networks. In *Advances in neural information processing systems*. 1185–1192.

[8] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D Sculley. 2016. What's your ML Test Score? A rubric for ML production systems. In *Reliable Machine Learning in the Wild-NIPS 2016 Workshop*.

[9] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015).

[10] Xue-Wen Chen and Xiaotong Lin. 2014. Big data deep learning: challenges and perspectives. *IEEE access* 2 (2014), 514–525.

[11] Cheng-Tao Chu, Sang K Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Kunle Olukotun, and Andrew Y Ng. 2007. Map-reduce for machine learning on multicore. In *Advances in neural information processing systems*. 281–288.

[12] PyTorch core team. 2017. PyTorch. http://pytorch.org/. (2017). [Online; Accessed 07-August-2017].

[13] E Crego, G Munoz, and F Islam. 2013. Big data and deep learning: Big deals or big delusions? Business. (2013).

[14] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large scale distributed deep networks. In *Advances in neural information processing systems*.

1223–1231.

[15] Cynthia Dwork. 2008. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*. Springer, 1–19.

[16] Amir Efrati. 2013. How deep learning works at Apple. (2013).

[17] Kathleen M Eisenhardt. 1989. Building theories from case study research. *Academy of management review* 14, 4 (1989), 532–550.

[18] Martin Fowler and Kent Beck. 1999. *Refactoring: improving the design of existing code*. Addison-Wesley Professional.

[19] John Gantz and David Reinsel. 2010. The Digital Universe Decade. *Are You Ready* (2010).

[20] David Geer. 2005. Taking the graphics processor beyond graphics. *Computer* 38, 9 (2005), 14–16.

[21] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 249–256.

[22] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. 2017. Google Vizier: A Service for Black-Box Optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1487–1495.

[23] K. Haberlin, B. McGilpin, and C. Ouellette. 2012. Governor Patrick Announces New Initiative to Strengthen Massachusetts? Position as a World Leader in Big Data. (2012).

[24] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[25] White House. 2012. Obama administration unveils "big data" initiative: Announces $200 million in new R&D investments. (2012).

[26] P Hustinx. 2013. EU Data Protection Law: The Review of Directive 95/46/EC and the Proposed General Data Protection Regulation. https://edps.europa.eu/sites/edp/files/publication/14-09-15_article_eui_en.pdf. (2013). [Online; Accessed 20-August-2017].

[27] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 675–678.

[28] Nicola Jones. 2014. The learning machines. *Nature* 505, 7482 (2014), 146.

[29] Lukasz Kaiser, Aidan N Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. 2017. One Model To Learn Them All. *arXiv preprint arXiv:1706.05137* (2017).

[30] Upulee Kanewala and James M Bieman. 2014. Testing scientific software: A systematic literature review. *Information and software technology* 56, 10 (2014), 1219–1232.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[32] Quoc V Le. 2013. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 8595–8598.

[33] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[34] Jimmy Lin and Alek Kolcz. 2012. Large-scale machine learning at twitter. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 793–804.

[35] Jimmy Lin and Dmitriy Ryaboy. 2013. Scaling big data mining infrastructure: the twitter experience. *ACM SIGKDD Explorations Newsletter* 14, 2 (2013), 6–19.

[36] James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela H Byers. 2011. Big data: The next frontier for innovation, competition, and productivity. (2011).

[37] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. 2013. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1222–1230.

[38] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model.. In *Interspeech*, Vol. 2. 3.

[39] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[40] J David Morgenthaler, Misha Gridnev, Raluca Sauciuc, and Sanjay Bhansali. 2012. Searching for build debt: Experiences managing technical debt at Google. In *Proceedings of the Third International Workshop on Managing Technical Debt*. IEEE Press, 1–6.

[41] Chris Murphy, Gail E Kaiser, and Marta Arias. 2007. An Approach to Software Testing of Machine Learning Applications. In *SEKE*. 167.

[42] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. 2011. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*. 689–696.

[43] Biswanath Panda, Joshua S Herbach, Sugato Basu, and Roberto J Bayardo. 2009. Planet: massively parallel learning of tree ensembles with mapreduce. *Proceedings of the VLDB Endowment* 2, 2 (2009), 1426–1437.

[44] Rajesh Parekh. 2017. Designing AI at Scale to Power Everyday Life. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 27–27.

[45] Alec Radford, Luke Metz, and Soumith Chintala. 2016. Unsupervised representation learning with deep convolutional generative adversarial networks. *ICLR* 1, 3 (2016).

[46] Christopher Ré, Divy Agrawal, Magdalena Balazinska, Michael Cafarella, Michael Jordan, Tim Kraska, and Raghu Ramakrishnan. 2015. Machine Learning and Databases: The Sound of Things to Come or a Cacophony of Hype?. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 283–284.

[47] D Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems*. 2503–2511.

[48] Securities and E. Commission. 2013. SEC Charges Knight Capital With Violations of Market Access Rule. (2013).

[49] Alexander Smola and Shravan Narayanamurthy. 2010. An architecture for parallel topic models. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 703–710.

[50] Latanya Sweeney. 2002. K-anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10, 5 (2002), 557–570.

[51] Sandeep Tata, Alexandrin Popescul, Marc Najork, Mike Colagrosso, Julian Gibbons, Alan Green, Alexandre Mah, Michael Smith, Divanshu Garg, Cayden Meyer, et al. 2017. Quick Access: Building a Smart Experience for Google Drive. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1643–1651.

[52] Geoff Walsham. 1995. Interpretive case studies in IS research: nature and method. *European Journal of information systems* 4, 2 (1995), 74.

[53] Wei Wang, Meihui Zhang, Gang Chen, HV Jagadish, Beng Chin Ooi, and Kian-Lee Tan. 2016. Database Meets Deep Learning: Challenges and Opportunities. *ACM SIGMOD Record* 45, 2 (2016), 17–22.

[54] Pengtao Xie, Misha Bilenko, Tom Finley, Ran Gilad-Bachrach, Kristin Lauter, and Michael Naehrig. 2014. Crypto-nets: Neural networks over encrypted data. *arXiv preprint arXiv:1412.6181* (2014).

[55] Yahoo. 2017. TensorFlowOnSpark. https://github.com/yahoo/TensorFlowOnSpark/. (2017). [Online; Accessed 20-August-2017].

[56] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2–2.

[57] Blaise Zerega. 2017. AI Weekly: Google shifts from mobile-first to AI-first world. https://venturebeat.com/2017/05/18/ai-weekly-google-shifts-from-mobile-first-to-ai-first-world/. (2017). [Online; Accessed 20-August-2017].

[58] A. Zheng. 2014. The challenges of building machine learning tools for the masses. In *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*.