Database Meets Deep Learning: Challenges and Opportunities

Wei Wang†, Meihui Zhang‡, Gang Chen§,
H. V. Jagadish#, Beng Chin Ooi†, Kian-Lee Tan†
†National University of Singapore ‡ Singapore University of Technology and Design
§Zhejiang University #University of Michigan
†{wangwei, ooibc, tankl}@comp.nus.edu.sg ‡meihui_zhang@sutd.edu.sg
§cg@zju.edu.cn #jag@umich.edu

ABSTRACT

Deep learning has recently become very popular on account of its incredible success in many complex data-driven applications, such as image classification and speech recognition. The database community has worked on data-driven applications for many years, and therefore should be playing a lead role in supporting this new wave. However, databases and deep learning are different in terms of both techniques and applications. In this paper, we discuss research problems at the intersection of the two fields. In particular, we discuss possible improvements for deep learning systems from a database perspective, and analyze database applications that may benefit from deep learning techniques.

1. INTRODUCTION

In recent years, we have witnessed the success of numerous data-driven machine-learning-based applications. This has prompted the database community to investigate the opportunities for integrating machine learning techniques in the design of database systems and applications [28]. A branch of machine learning, called deep learning [20, 30, 16], has attracted worldwide interest in recent years due to its excellent performance in multiple areas including speech recognition, image classification and natural language processing (NLP). The foundation of deep learning was established about twenty years ago in the form of neural networks. Its recent resurgence is mainly fueled by three factors: immense computing power, which reduces the time to train and deploy new models, e.g. Graphic Processing Unit (GPU) enables the training systems to run much faster than those in the 1990s; massive (labeled) training datasets (e.g. ImageNet) enable a more comprehensive knowledge of the domain to be acquired; new deep learning models (e.g. AlexNet [18]) improve the ability to capture data regularities.

Database researchers have been working on system optimization and large scale data-driven applications since 1970s, which are closely related to the first two factors. It is natural to think about the relationships between databases and deep learning. First, are there any insights that the database community can offer to deep learning? It has been shown that larger training datasets and a deeper model structure improve the accuracy of deep learning models. However, the side effect is that the training becomes more costly. Approaches have been proposed to accelerate the training speed from both the system perspective [3, 17, 6, 27, 10] and the theory perspective [9, 45]. Since the database community has rich experience with system optimization, it would be opportune to discuss the applicability of database techniques for optimizing deep learning systems. For example, distributed computing and memory management are key database technologies also central to deep learning.

Second, are there any deep learning techniques that can be adapted for database problems? Deep learning emerged from the machine learning and computer vision communities. Recently, it has been successfully applied to other domains, like NLP [12]. However, few studies have been conducted using deep learning techniques for traditional database problems. This is partially because traditional database problems — like indexing, transaction and storage management — involve less uncertainty, whereas deep learning is good at predicting over uncertain events. Nevertheless, there are problems in databases like knowledge fusion [7] and crowdsourcing [26], which are probabilistic problems. It is possible to apply deep learning techniques in these areas. We will discuss specific problems like querying interface, knowledge fusion, etc. in this paper.

The rest of this paper is organized as follows: Section 2 provides background information about deep learning models and training algorithms; Section 3

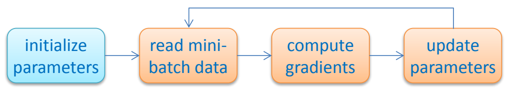

Figure 1: Stochastic Gradient Descent.

discusses the application of database techniques for optimizing deep learning systems. Section 4 describes research problems in databases where deep learning techniques may help to improve performance. Some final thoughts are presented in Section 5.

2. BACKGROUND

Deep learning refers to a set of machine learning models which try to learn high-level abstractions (or representations) of raw data through multiple feature transformation layers. Large training datasets and deep complex structures enhance the ability of deep learning models for learning effective representations for tasks of interest. There are three popular categories of deep learning models according to the types of connections between layers [20], namely feedforward models (direct connection), energy models (undirected connection) and recurrent neural networks (recurrent connection). Feedforward models, including Convolution Neural Network (CNN), propagate input features through each layer to extract high-level features. CNN is the state-of-the-art model for many computer vision tasks. Energy models, including Deep Belief Network (DBN) are typically used to pre-train other models, e.g., feedforward models. Recurrent Neural Network (RNN) is widely used for modeling sequential data. Machine translation and language modeling are popular applications of RNN.

Before deploying a deep learning model, the model parameters involved in the transformation layers need to be trained. The training turns out to be a numeric optimization procedure to find parameter values that minimize the discrepancy (loss function) between the expected output and the real output. Stochastic Gradient Descent (SGD) is the most widely used training algorithm. As shown in Figure 1, SGD initializes the parameters with random values, and then iteratively refines them based on the computed gradients with respect to the loss function. There are three commonly used algorithms for gradient computation corresponding to the three model categories above: Back Propagation (BP), Contrastive Divergence (CD) and Back Propagation Through Time (BPTT). By regarding the layers of a neural net as nodes of a graph, these algorithms can be evaluated by traversing the graph in certain sequences. For instance, the BP algorithm is illustrated in Figure 2, where a simple feedforward model is trained by traversing along the solid arrows to compute the data (feature) of each layer, and along the dashed arrows to compute the gradient of each layer and each parameter (W and b).

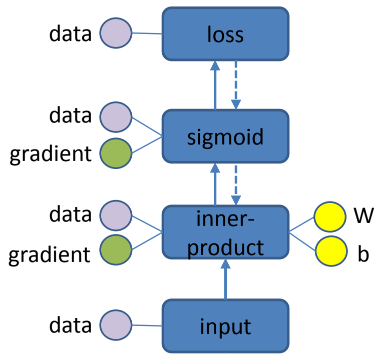

Figure 2: Data flow of Back-Propagation.

3. DATABASES TO DEEP LEARNING

In this section, we discuss the optimization techniques used in deep learning systems, and research opportunities from the perspective of databases. After that, we give a summary of existing deep learning systems on their usage of these techniques.

3.1 Distributed Computing

The database community has a long history of working on distributed environment, ranging from parallel databases [21] and peer-to-peer systems [38] to cloud computing [23]. Distributed training is a natural solution for accelerating the training speed of deep learning models. There are two basic parallelism schemes for distributed training, namely, data parallelism and model parallelism. As shown in Figure 3 (ignore the shaded area), Data parallelism runs multiple worker groups for the BP (CD, BPTT) algorithm, where each worker group runs over a partition of the training dataset. There is a central server updating the model parameters [46, 29] based on parameter gradients. Model parallelism partitions the neural net model into different workers (each worker is assigned a subset of layers), who run the BP (CD, BPTT) algorithm synchronously. We will discuss some research problems relevant to databases arising from distributed training in the following paragraphs.

3.2 Communication

Distributed training typically adopts the parameter server architecture, which uses central servers for receiving parameter gradients and updating pa-

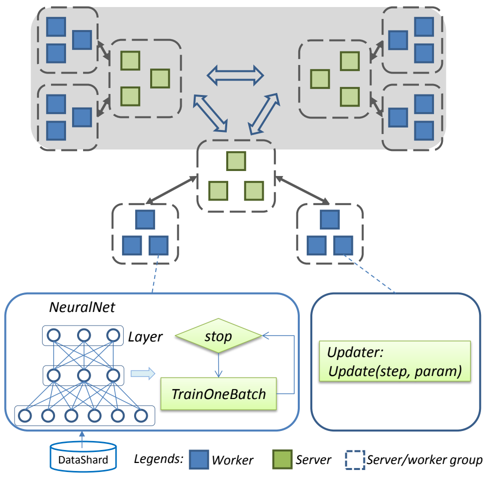

Figure 3: Overview of Apache SINGA.

rameters [40]. The central servers are likely to become the bottleneck, because they have to communicate with all workers for all model parameters. Consequently, it is important to investigate efficient communication protocols for both single-node multiple GPU training and training over a large cluster. Possible research directions include : a) compressing the parameters and gradients for transferring [31]; b) organizing servers in an optimized topology to reduce the communication burden of each single server, e.g., tree structure [14], AllReduce structure [44] (all-to-all connection) and the ring structure as shown in Figure 3; c) using more efficient networking hardware like RDMA [3].

3.3 Memory Management

Memory management is a hot topic in the database community with a significant amount of research towards in-memory databases [36]. Effective memory management not only reduces the memory footprint, but also improves the training speed. Deep learning models are becoming larger and larger, and already occupy a huge amount of memory space. For example, the VGG model [33] cannot be trained on normal GPU cards due to memory size constraints. One approach to reduce memory consumption is to use data types with shorter width, e.g., 16-bit float [5]. Another approach is to share memory space for data items that do not have conflicts. For example, as shown in Figure 2, for each iteration, the data item in the *inner-product* layer will not be used after finishing the forward propagation of the *sigmoid* layer, hence its memory space can be reused by the gradient of the *sigmoid* layer for the time remaining in the current iteration. Data conflicts can be analyzed from a computation graph [2], e.g., using functional programming or declarative programming.

It has been demonstrated that many existing systems suffer from poor scalability caused by inefficient memory access in NUMA machine [40]. For training using GPUs, the data transfer between GPUs and between GPU and CPU incurs significant overhead [39]. Hence, optimization towards data locality and device topology (e.g., GPU connection) is necessary for improving training speed.

3.4 Concurrency and Consistency

Concurrency and consistency are traditional research problems in databases. For distributed training of deep learning models, they also matter. Currently, both declarative programming (e.g., Theano and TenforFlow) and imperative programming (e.g., Caffe and SINGA) have been adopted in existing systems for concurrency implementation. Most deep learning systems use threads and locks directly. Other concurrency implementation methods like actor model (good at failure recovery), co-routine and communicating sequential processes have not been adopted.

Sequential consistency (from synchronous training) and eventual consistency (from asynchronous training) are typically used for distributed deep learning. Both approaches have scalability issues [40]. Recently, there are studies for training convex models (deep learning models are non-linear and non-convex) using a value bounded consistency model [43]. Researchers are starting to investigate the influence of consistency models on distributed training [14]. There remains much research to be done on how to provide flexible consistency models for distributed training, and how each consistency model affects the scalability of the system, e.g., communication.

3.5 Data Storage and Loading

HDFS, prefetching and compression are used in deep learning systems for handling massive training datasets. There is research [11] on improving the efficiency of training by loading a subset of data for each epoch. In particular, "valuable" data instances are sampled with high probability. Sampling is a common technique in databases, which is useful for efficiency optimization, but may affect the accuracy because the model is not trained over all training data.

3.6 Failure Recovery

Databases systems have good durability via logging (e.g., command log) and checkpointing. Current deep learning systems recover the training from crashes mainly based on checkpointing files [10]. In contrast with database systems, which enforce strict

Table 1: Summary of optimization techniques used in existing systems as of March 28, 2016.

	SINGA[27]	**Caffe**[17]	**Mxnet**[2]	**TensorFlow**[10]	**Theano**[1]	**Torch**[4]
Parallelism	★★★	★★	★★	★★★	★★	★★
Communication	★★	★	★★	★	★	★
Consistency	★★	★★	★★	★★	★	★★
Memory Management	★	★	★★	★	★★	★★
Data Storage and Loading	★★	★★	★★★	★	★	★
Failure recovery	★	★	★	★	★	★

consistency in transactions, the SGD algorithm can tolerate a certain degree of inconsistency. Therefore, logging is not a must. How to exploit the SGD properties and system architectures to implement fault tolerance efficiently is an interesting problem. Considering that distributed training would replicate the model status, it is thus possible to recover from a replica instead of checkpointing files. Robust frameworks (or concurrency model) like actor model, could be adopted to implement this kind of failure recovery.

3.7 Optimization Techniques in Existing Systems

Existing systems more or less use the aforementioned techniques for optimizing their performance. We compare the usage of the optimization techniques in existing systems in Table 1. It is not a comprehensive comparison as some optimization techniques are implemented by third-party libraries, which may not be covered. For instance, Twitter added Multi-GPU training for Torch[4][1], and Platoon is working on Multi-GPU for Theano[1] [2]. All systems adopt data parallelism for distributed training. SINGA [27] and TensorFlow (TF)[10] also support model parallelisim. Asynchronous data transferring is enabled in all systems. Different communication topologies are used by different system for transferring parameters and gradients, e.g., Mxnet [2] uses AllReduce and Caffe [17] uses a tree topology. Currently, SINGA provides more communication topologies for users to select, including P2P and AllReduce across multiple nodes. No systems can automatically tune the communication topology based on hardware and other context. Consequently, users have to configure the topology manually. All systems except Theano support both synchronous training (sequential consistency) and asynchronous training (eventual consistency). Platoon added asynchronous training for Theano. Mxnet enables data structures to share memory if they do not have conflicts which can be derived from the dataflow graph. TF and Theano have similar graph, hence could be able to implement this technique. For the time being, TF is not fully optimized in terms of memory and speed[3]. The recent version of Theano (v0.8) includes CnMeM[4] for optimized GPU memory management. Data prefetching is enabled by all systems. SINGA and Mxnet have integrated with HDFS. Data compression for image data is enabled in Mxnet and Caffe. Failure recovery is simply implemented by checkpointing.

[1]`https://blog.twitter.com/2016/distributed-learning-in-torch`
[2]`https://github.com/mila-udem/platoon`

4. DEEP LEARNING TO DATABASES

Deep learning applications, such as computer vision and natural language understanding, may appear very different from database applications. However, the core idea of deep learning, known as feature (or representation) learning, is applicable to a wide range of applications. Intuitively, once we have effective representations for entities, e.g., images, words, table rows or columns, we can compute entity similarity, perform clustering, train prediction models, and retrieve data with different modalities [42, 41] etc..

Recent studies have shown that deep learning has good performance in natural language processing [12], learning structured output [35, 37] and predicting the relationship between entities (e.g., words) [34]. These techniques can indeed be directly used in some database applications. We shall highlight a few below.

4.1 Query Interface

Natural language query interfaces have been attempted for decades [22], because of their great desirability, particularly for non-expert database users. However, it is challenging for database systems to interpret (or understand) the semantics of natural language queries. Recently, deep learning models, including CNN and RNN, have achieved state-of-the-art performance for NLP tasks. Moreover, RNN

[3]`https://github.com/soumith/convnet-benchmarks/issues/66`
[4]`https://github.com/NVIDIA/cnmem`

has been shown to be able to learn structured output [35, 37]. It is interesting to apply deep learning models e.g. RNN, for parsing natural language queries (reading word by word) to generate SQL queries. The challenge is that deep learning models like RNN, typically require a large amount of (labeled) training samples, whereas it is not easy to collect such a dataset with each natural language query associated with a matching SQL query. One possible solution is to train a baseline model with a small dataset, and gradually refining it with users' feedback. For instance, users could help correct the generated SQL query, and these feedback essentially serve as labeled data for subsequent training.

4.2 Query Plan

Query plan optimization is a traditional database problem. Most current database systems use complex heuristic algorithms to generate the query plan. According to [15], each query plan of a parametric SQL query template has an optimality region. As long as the parameters of the SQL query are within this region, the optimal query plan does not change. In other words, query plans are in-sensitive to small variations of the input parameters. Therefore, we can learn a query planner which learns from a set of pairs of SQL queries and optimal plans to generate (similar) plans for new (similar) queries. For instance, we can learn a RNN model that accepts the SQL query elements and meta-data (like buffer size and primary key) as input, and generates a tree structure [37] representing the query plan. We can also combine reinforcement learning (like AlphaGo [32]) to train the model on-line based on the execution result e.g., performance in terms of execution time.

4.3 Data and Knowledge Fusion

Data fusion is important for data-driven applications. Recently, database researchers have moved on to knowledge fusion for knowledge-base (graph) construction [7]. Deep learning [34, 25] has been applied to knowledge-base related problems. With the advances of word vector models and RNN models in NLP, it is likely that deep learning can help more in related areas. First, deep learning models can be developed to extract entities from web pages (e.g., sentences). Second, deep learning models can be trained to predict relationship of entities. We can then enhance the knowledge-base using extracted entities and relationships.

4.4 Spatial and Temporal Data

Spatial and temporal data are common data types in database systems [13, 8], and are commonly used for trend analysis, progression modeling and predictive analytics [19]. Spatial data is typically mapped into rectangular blocks for modeling spatial relationships, e.g, locality. All blocks together compose an image. CNN model is effective for extracting spatial relationships between nearby pixels in an image, and thus is suitable for processing spatial data. For instance, if we have the real-time location data (e.g., GPS data) of moving objects, we could learn a CNN model to capture the density relationships of nearby areas for predicting the traffic congestion for a future time point. When temporal data is modeled as features over a time matrix, deep learning models, e.g. RNN and CNN (1-D convolution on the time dimension), can be designed to model time dependency and predict the occurrence in a future time point. A particular example would be disease progression modeling [24] based on historical medical records, where doctors would want to estimate the onset of certain severity of a known disease.

5. CONCLUSION

In this paper, we have discussed databases and deep learning. Databases have many techniques for optimizing system performance, while deep learning is good at learning effective representation for data-driven applications. We note that these two "different" areas share some common techniques for improving the system performance, such as memory optimization and parallelism. We have discussed some possible improvements for deep learning systems using database techniques, and research problems applying deep learning techniques in database applications. Let us not miss the opportunity to contribute to the existing challenges ahead!

6. ACKNOWLEDGEMENT

We would like to thank Divesh Srivastava for his valuable comments. This work is supported by A*STAR project 1321202073.

7. REFERENCES

[1] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[2] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.

[3] A. Coates, B. Huval, T. Wang, D. J. Wu, B. C. Catanzaro, and A. Y. Ng. Deep learning with COTS HPC systems. In *ICML*, pages 1337–1345, 2013.

[4] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In

BigLearn, NIPS Workshop, number EPFL-CONF-192376, 2011.

[5] M. Courbariaux, Y. Bengio, and J.-P. David. Low precision arithmetic for deep learning. *arXiv preprint arXiv:1412.7024*, 2014.

[6] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In *NIPS*, pages 1232–1240, 2012.

[7] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, K. Murphy, S. Sun, and W. Zhang. From data fusion to knowledge fusion. *PVLDB*, 7(10):881–892, 2014.

[8] X. L. Dong and W.-C. Tan. A time machine for information: looking back to look forward. *Proceedings of the VLDB Endowment*, 8(12):2044–2045, 2015.

[9] J. C. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[10] M. A. et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[11] J. Gao, H. Jagadish, and B. C. Ooi. Active sampler: Light-weight accelerator for complex data analytics at scale. *arXiv preprint arXiv:1512.03880*, 2015.

[12] Y. Goldberg. A primer on neural network models for natural language processing. *CoRR*, abs/1510.00726, 2015.

[13] C. Guo, C. S. Jensen, and B. Yang. Towards total traffic awareness. *ACM SIGMOD Record*, 43(3):18–23, 2014.

[14] S. Gupta, W. Zhang, and J. Milthorpe. Model accuracy and runtime tradeoff in distributed deep learning. *arXiv preprint arXiv:1509.04210*, 2015.

[15] J. R. Haritsa. The picasso database query optimizer visualizer. *Proceedings of the VLDB Endowment*, 3(1-2):1517–1520, 2010.

[16] Y. B. Ian Goodfellow and A. Courville. Deep learning. Book in preparation for MIT Press, 2016.

[17] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[19] S. Laxman and P. S. Sastry. A survey of temporal data mining. *SADHANA, Academy Proceedings in Engineering Sciences*, 31:173198, April 2006.

[20] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[21] M. L. Lee, M. Kitsuregawa, B. C. Ooi, K.-L. Tan, and A. Mondal. Towards self-tuning data placement in parallel database systems. In *ACM SIGMOD Record*, volume 29, pages 225–236. ACM, 2000.

[22] F. Li and H. Jagadish. Constructing an interactive natural language interface for relational databases. *PVLDB*, 8(1):73–84, 2014.

[23] F. Li, B. C. Ooi, M. T. Özsu, and S. Wu. Distributed data management using mapreduce. *ACM Comput. Surv.*, 46(3):31:1–31:42, 2014.

[24] D. R. Mould. Models for disease progression: New approaches and uses. *Clinical Pharmacology & Therapeutics*, 92(1):125–131, 2012.

[25] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *arXiv preprint arXiv:1503.00759*, 2015.

[26] B. C. Ooi, K. Tan, Q. T. Tran, J. W. L. Yip, G. Chen, Z. J. Ling, T. Nguyen, A. K. H. Tung, and M. Zhang. Contextual crowd intelligence. *SIGKDD Explorations*, 16(1):39–46, 2014.

[27] B. C. Ooi, K.-L. Tan, S. Wang, W. Wang, Q. Cai, G. Chen, J. Gao, Z. Luo, A. K. H. Tung, Y. Wang, Z. Xie, M. Zhang, and K. Zheng. SINGA: A distributed deep learning platform. In *ACM Multimedia*, 2015.

[28] C. Ré, D. Agrawal, M. Balazinska, M. I. Cafarella, M. I. Jordan, T. Kraska, and R. Ramakrishnan. Machine learning and databases: The sound of things to come or a cacophony of hype? In *SIGMOD*, pages 283–284, 2015.

[29] B. Recht, C. Re, S. J. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, pages 693–701, 2011.

[30] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[31] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *INTERSPEECH*, pages 1058–1062, 2014.

[32] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[33] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[34] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In *NIPS*, pages 926–934, 2013.

[35] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[36] K.-L. Tan, Q. Cai, B. C. Ooi, W.-F. Wong, C. Yao, and H. Zhang. In-memory databases: Challenges and opportunities from software and hardware perspectives. *ACM SIGMOD Record*, 44(2):35–40, 2015.

[37] O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton. Grammar as a foreign language. *arXiv:1412.7449*, 2014.

[38] Q. H. Vu, M. Lupu, and B. C. Ooi. *Peer-to-peer computing*. Springer, 2010.

[39] W. Wang, G. Chen, H. Chen, T. T. A. Dinh, J. Gao, B. C. Ooi, K.-L. Tan, and S. Wang. Deep learning at scale and at ease. *arXiv:1603.07846*, 2016.

[40] W. Wang, G. Chen, T. T. A. Dinh, J. Gao, B. C. Ooi, K.-L. Tan, and S. Wang. SINGA: Putting deep learning in the hands of multimedia users. In *ACM Multimedia*, 2015.

[41] W. Wang, B. C. Ooi, X. Yang, D. Zhang, and Y. Zhuang. Effective multi-modal retrieval based on stacked auto-encoders. *PVLDB*, 7(8):649–660, 2014.

[42] W. Wang, X. Yang, B. C. Ooi, D. Zhang, and Y. Zhuang. Effective deep learning-based multi-modal retrieval. *The VLDB Journal*, pages 1–23, 2015.

[43] J. Wei, W. Dai, A. Qiao, Q. Ho, H. Cui, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing. Managed communication and consistency for fast data-parallel iterative analytics. In *SoCC*, pages 381–394, 2015.

[44] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun. Deep image: Scaling up image recognition. *CoRR*, abs/1501.02876, 2015.

[45] M. D. Zeiler. Adadelta: An adaptive learning rate method. *arXiv:1212.5701*, 2012.

[46] S. Zhang, A. Choromanska, and Y. LeCun. Deep learning with elastic averaging SGD. *CoRR*, abs/1412.6651, 2014.