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, including 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 [29]. A branch of machine learning, called deep learning [22, 18], 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 [20]) 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 [5, 19, 9, 28, 11] and the theory perspective [45, 12]. 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. They are 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 [13]. However, few studies have been conducted using deep learning techniques for 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 [10] and crowdsourcing [27], 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 discusses the application of database techniques for

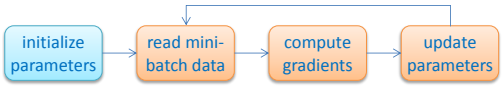

Figure 1: Stochastic Gradient Descent.

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 [22], namely feedforward models (directed 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

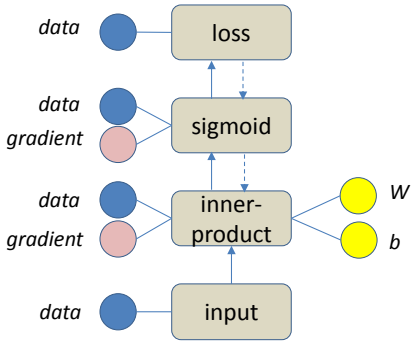

Figure 2: Data flow of Back-Propagation.

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).

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.

3.1 Stand-alone Training

Currently, the most effective approach for improving the training speed of deep learning models is to use Nvidia GPU with the cuDNN library. Researchers are also working on other hardware, e.g. FPGA [21]. Besides exploiting advancements in hardware technology, operation scheduling and memory management are two important components to consider.

3.1.1 Operation Scheduling

Training algorithms of deep learning models typically involve expensive linear algebra operations as shown in Figure 3, where the matrix $W1$ and $W2$ could be larger than $4096*4096$. Operation scheduling is to first detect the data dependency of operations and then place the operations without dependencies onto executors, e.g., CUDA streams and CPU threads. Take the operations in Figure 3 as an example, $a1$ and $a2$ in Figure 3 could be computed in parallel because they have no dependencies. The first step could be done statically based on dataflow graph or dynamically [3] by analyzing the orders of read and write operations. Databases also have this kind of problems in optimizing transaction execution [44] and query plans. Those solutions should be considered for deep learning systems. For instance, databases use cost models to estimate query plans. For deep learning, we may also create a cost

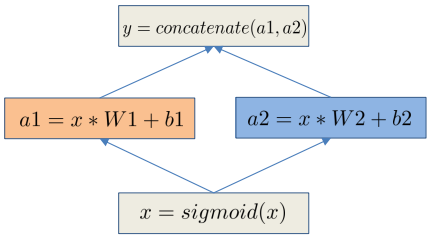

Figure 3: Sample operations from a deep learning model.

model to find an optimal operation placing strategy for the second step of operation scheduling given a fixed computing resources including executors and memory.

3.1.2 Memory Management

Deep learning models are becoming larger and larger, and already occupy a huge amount of memory space. For example, the VGG model [32] cannot be trained on normal GPU cards due to memory size constraints. Many approaches have been proposed towards reducing memory consumption. Shorter data representation, e.g. 16-bit float [7] is now supported by CUDA. Memory sharing is an effective approach for memory saving [3]. Take Figure 3 as an example, the input and output of the *sigmoid* function share the same variable and thus the same memory space. Such operations are called 'in-place' operations. Recently, two approaches were proposed to trade-off computation time for memory. Swapping memory between GPU and CPU resolves the problem of small GPU memory and large model size by swapping variables out to CPU and then swapping back manually[8]. Another approach drops some variables to free memory and recomputes them when necessary based on the static dataflow graph[4].

Memory management is a hot topic in the database community with a significant amount of research towards in-memory databases [35, 46], including locality, paging and cache optimization. To elaborate more, the paging strategies could be useful for deciding when and which variable to swap. In addition, failure recovery in databases is similar to the idea of dropping and recomputing approach, hence the logging techniques in databases could be considered. If all operations (and execution time) are logged, we can then do runtime analysis without the static dataflow graph. Other techniques, including garbage collection and memory pool, would also be useful for deep learning systems, especially for GPU memory management.

3.2 Distributed Training

Distributed training is a natural solution for accelerating the training speed of deep learning models. The parameter server architecture [9] is typically used, in which the workers compute parameter gradients and the servers update the parameter values after receiving gradients from workers. There are two basic parallelism schemes for distributed training, namely, data parallelism and model parallelism. In data parallelism, each worker is assigned a data partition and a model replica, while for model parallelism, each worker is assigned a partition of the model and the whole dataset. The database community has a long history of working on distributed environment, ranging from parallel databases [23] and peer-to-peer systems [37] to cloud computing [25]. We will discuss some research problems relevant to databases arising from distributed training in the following paragraphs.

3.2.1 Communication and Synchronization

Given that deep learning models have a large set of parameters, the communication overhead between workers and servers is likely to be the bottleneck of a training system, especially when the workers are running on GPUs which decrease the computation time. In addition, for large clusters, the synchronization between workers can be significant. 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 transmission [30]; b) organizing servers in an optimized topology to reduce the communication burden of each single node, e.g., tree structure [15] and AllReduce structure [42] (all-to-all connection); c) using more efficient networking hardware like RDMA [5].

3.2.2 Concurrency and Consistency

Concurrency and consistency are critical concepts 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 explored.

Sequential consistency (from synchronous training) and eventual consistency (from asynchronous training) are typically used for distributed deep learn-

Table 1: Summary of optimization techniques used in existing systems as of July 2016.

	SINGA	Caffe	Mxnet	TensorFlow	Theano	Torch
1. operation scheduling	✓	x	✓	-	-	x
2. memory management	d+a+p	i	d+s	p	p	-
3. parallelism	d + m	d	d + m	d + m	-	d + m
4. consistency	s+a+h	s/a	s+a+h	s+a+h	-	s

1. x: not available: ✓: available 2. d: dynamic; a: swap; p: memory pool; i: in-place operation; s: static;
3. d: data parallelism; m: model parallelism; 4. s: synchronous; a: asynchronous; h:hybrid; -: unknown

ing. Both approaches have scalability issues [38]. Recently, there are studies for training convex models (deep learning models are non-linear and non-convex) using a value bounded consistency model [41]. Researchers are starting to investigate the influence of consistency models on distributed training [15, 16, 2]. 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, including communication overhead.

3.2.3 Fault Tolerance

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 [11]. However, frequent checkpointing would incur vast overhead. In contrast with database systems, which enforce strict consistency in transactions, the SGD algorithm used by deep learning training systems 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.

3.3 Existing Systems

A summary of existing systems in terms of the above mentioned optimization aspects is listed in Table 1. Many researchers have extended Caffe [19] with ad hoc optimizations, including memory swapping and communication optimization. However, the official version is not well optimized. Similarly, Torch [6] itself provides limited support for distributed training. Mxnet[3] has optimization for both memory and operations scheduling. Theano [1] is typically used for stand-alone training. TensorFlow [11] has the potential for the aforementioned static optimization based on the dataflow graph.

We are optimizing the Apache incubator SINGA system [28] starting from version 1.0. For stand-alone training, cost models are explored for runtime operation scheduling. Memory optimization including dropping, swapping and garbage collection with memory pool will be implemented. OpenCL is supported to run SINGA on a wide range of hardware including GPU, FPGA and ARM. For distributed training, SINGA (V0.3) has done much work on flexible parallelism and consistency, hence the focus would be on optimization of communication and fault-tolerance, which are missing in almost all systems.

4. DEEP LEARNING TO DATABASES

Deep learning applications, such as computer vision and NLP, 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 [40, 39] etc. We shall highlight a few deep learning models that could be adapted for database applications below.

4.1 Query Interface

Natural language query interfaces have been attempted for decades [24], 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 have achieved state-of-the-art performance for NLP tasks [13]. Moreover, RNN has been shown to be able to learn structured output [34, 36]. As one solution, we can apply RNN models for parsing natural language queries to generate SQL queries, and refine it using existing database approaches. For instance, heuristic rules could be applied to correct grammar errors in the generated SQL queries. The

challenge is that a large amount of (labeled) training samples is required to train the model. 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 Plans

Query plan optimization is a traditional database problem. Most current database systems use complex heuristic and cost models to generate the query plan. According to [17], 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 insensitive to small variations of the input parameters. Therefore, we can train a query planner which learns from a set of pairs of SQL queries and optimal plans to generate (similar) plans for new (similar) queries. To elaborate more, 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 [36] representing the query plan. Reinforcement learning (like AlphaGo [31]) could also be incorporated to train the model on-line using the execution time and memory footprint as the reward. Note that approaches purely based on deep learning models may not be very effective. In particular, the training dataset may not be comprehensive to include all query patterns, e.g. some predicates could be missing in the training datasets. To solve these problems, a better approach would be to combine database solutions and deep learning.

4.3 Crowdsourcing and Knowledge Bases

Many crowdsourcing [43] and knowledge base [10] applications involve entity extraction, disambiguation and fusion problems, where the entity could be a row of a database, a node in a graph, etc. With the advancements of deep learning models in NLP [13], it is opportune to consider deep learning for these problems. In particular, we can learn representations for entities and then do entity relationship reasoning [33] and similarity calculation using the learned representations.

4.4 Spatial and Temporal Data

Spatial and temporal data are common data types in database systems [14], and are commonly used for trend analysis, progression modeling and predictive analytics. Spatial data is typically processed by mapping moving objects into rectangular blocks. If we regard each block as a pixel of one image, then deep learning models, e.g., CNN, could be exploited to extract the spatial locality between nearby blocks. 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, can be designed to model time dependency and predict the occurrence in a future time point. A particular example would be disease progression modeling [26] based on historical medical records, where doctors would want to estimate the onset of certain severity of a known disease.

5. CONCLUSIONS

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 exciting challenges ahead!

6. ACKNOWLEDGEMENT

We would like to thank Divesh Srivastava for his valuable comments. This work is supported by the National Research Foundation, Prime Minister's Office, Singapore, under its Competitive Research Programme (CRP Award No. NRF-CRP8-2011-08). Meihui Zhang is supported by SUTD Start-up Research Grant under Project No. SRG ISTD 2014 084.

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] J. Chen, R. Monga, S. Bengio, and R. Józefowicz. Revisiting distributed synchronous SGD. *CoRR*, abs/1604.00981, 2016.

[3] 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.

[4] T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost. *CoRR*, abs/1604.06174, 2016.

[5] 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.

[6] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

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

[8] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing. Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In *EuroSys*, page 4. ACM, 2016.

[9] 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.

[10] 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.

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

[12] 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.

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

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

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

[16] S. Hadjis, C. Zhang, I. Mitliagkas, and C. Ré. Omnivore: An optimizer for multi-device deep learning on cpus and gpus. *CoRR*, abs/1606.04487, 2016.

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

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

[19] 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.

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

[21] G. Lacey, G. W. Taylor, and S. Areibi. Deep learning on fpgas: Past, present, and future. *CoRR*, abs/1602.04283, 2016.

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

[23] 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.

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

[25] 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.

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] D. Silver and et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

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

[33] 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.

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

[35] 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.

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

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

[38] 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.

[39] 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.

[40] 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.

[41] 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.

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

[43] T. Wu, L. Chen, P. Hui, C. J. Zhang, and W. Li. Hear the whole story: Towards the diversity of opinion in crowdsourcing markets. *PVLDB*, 8(5):485–496, 2015.

[44] C. Yao, D. Agrawal, G. Chen, Q. Lin, B. C. Ooi, W. F. Wong, and M. Zhang. Exploiting single-threaded model in multi-core in-memory systems. *IEEE Trans. Knowl. Data Eng.*, 2016.

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

[46] H. Zhang, G. Chen, B. C. Ooi, K. Tan, and M. Zhang. In-memory big data management and processing: A survey. *IEEE Trans. Knowl. Data Eng.*, 27(7):1920–1948, 2015.