Scalable Deep Document / Sequence Reasoning with Cognitive Toolkit

Sayan Pathak
Microsoft Research
One Microsoft Way
Redmond, WA 98052 USA

+1 (425) 5387386
sayanpa@microsoft.com

Pengcheng He
Advanced Technology Group
Microsoft China
Beijing China

+86 (10) 59172966
penhe@microsoft.com

William Darling
Microsoft AI and Research
Microsoft Search Technology Center
Munich, Germany

+49 (89) 31766089
wdarling@microsoft.com

ABSTRACT
Deep Neural Networks (DNNs) have revolutionized the way that machines understand language and have allowed us to create models that answer textual questions, translate pairs of languages, and intelligently compare document corpora. At the heart of these successes lie core techniques that fall into the area of sequence understanding. While powerful, dealing with variable-size sequences in DNNs requires deep understanding and experience in creating such networks, which can be daunting to many scientists and engineers. This tutorial will focus on introducing core concepts, end-to-end recipes, and key innovations facilitated by the cross-platform fully open-source Cognitive Toolkit (formerly called CNTK) with superior scalability (up to 1000 GPUs) for very large data corpora. Specifically, we will present tutorials on basic sequence understanding, intermediate sequence-to-sequence translation (both with and without attention), and the advanced Reasoning Network (ReasoNet) which has achieved industry-leading results in reading comprehension.

Keywords
Deep neural networks; Cognitive Toolkit; CNTK; Sequence to Sequence, ReasoNet

1. INTRODUCTION
Deep Neural Networks (DNNs) are powerful models that have achieved state-of-the-art performance on many diverse difficult learning tasks. DNNs are powerful because they can perform arbitrary parallel computation for a modest number of steps. Large DNNs can be trained with supervised backpropagation whenever the labeled training set has enough information to specify the network's parameters. Thus, if there exists a parameter setting of a large DNN that achieves good results (for example, because humans can solve the task very rapidly), supervised backpropagation will find these parameters and solve the problem.

Sequential data poses a challenge for DNNs because they traditionally they have required the dimensionality of the inputs and outputs to be known and fixed. Recurrent neural networks, however, allow DNNs to have loops, and therefore accept data of arbitrary lengths. One problem with RNNs is that as the gradients must be backpropagated through time, they often become infinitesimally small or explode in size. Sutskever et al. [1] have shown that Long Short-Term Memory (LSTM) architectures – a slightly more complex version of RNNs – can solve general sequence-to-sequence problems with the vanishing/exploding gradient problem taken care of. The LSTM's ability to successfully learn on data with long-range temporal dependencies makes it a natural choice for this application due to the considerable time lag between the inputs and their corresponding output. Recently, research has shown how the human brain pays selective attention to certain parts of the enormous amount of information that comes our way by filtering out extraneous data that is not currently of import for the task at hand. One can extend this concept to neural networks. This approach – commonly referred to as "attention" – helps improve on challenging sequence-processing tasks where more simple sequence-to-sequence models fail, and we will cover it extensively in this tutorial. Additionally, we will also walk through the Reasoning Network (ReasoNet) framework which has achieved industry-leading results in reading comprehension.

In this tutorial, we will introduce Microsoft's Cognitive Toolkit, also known as CNTK, to solve many DNN-based algorithms applied to sequence data. We will introduce the audience to solving sequence-to-sequence problems via hands-on tutorials using the toolkit. While there are several other deep learning toolkits, we will explain several key innovations that are made available in CNTK which provide unique advantages over other tools in addition to ease of use, speed, and scalability. CNTK achieves such scalability via advanced algorithms such as 1-bit SGD [3] and block-momentum SGD [4]. We will explain these algorithms in the tutorial, and how they directly lead to increases in productivity. The goal is to enable the audience to build their own sequence-based networks and train them in a distributed manner by the end of the tutorial.

2. Sequence Reasoning
Comprehension of text corpora is predicated by techniques rooted in sequence understanding. In the context of NLP, sequences are defined as a set of words in a corpus where the positional information is modelled and may or may not be of importance in the context of the task being performed. In the context of language translation, the positions of words in a sentence changes the meaning of the text (depending on the language). However, in web query language, positional information may not be of much consequence, at least for very short queries. Regardless of the domain, the deep learning community has found RNNs using LSTM and GRU architectures to be very effective in capturing long-range dependencies between word tokens.

We will start the tutorial by introducing the basic structure of an LSTM-based deep network to classify different words in a text sequence (aka "slot labelling"). With the basics of network building covered, we will introduce sequence-to-sequence models both with and without attention [5]. Finally, we will introduce the ReasoNet model for performing reading comprehension.

WWW 2017 Companion, April 3-7, 2017, Perth, Australia.
ACM 978-1-4503-4914-7/17/04.
http://dx.doi.org/10.1145/3041021.3051103

2.1 Sequence-to-Sequence

Sequence-to-sequence models are achieving state-of-the-art results in fields as diverse as machine translation, text summarization, and syntactic parsing. The framework of sequence-to-sequence networks is extremely powerful yet easily adaptable to different datasets with little or no domain-specific adaptation. Building from the sequence reasoning tutorial, participants will learn the sequence-to-sequence framework of encoder-decoders, where an input sequence is run through an encoder RNN (typically represented by an LSTM) to get a thought vector-space representation of its content, and then run through a second decoder RNN that translates the encoded representation into an output sequence. We will show how to implement basic versions of the model in CNTK along with some recent variations from the literature including, most predominantly, attention, where the decoder learns the specific parts of the encoded input that it should concentrate on while generating each word (or, more generally, label) in the output sequence, and as illustrated below.

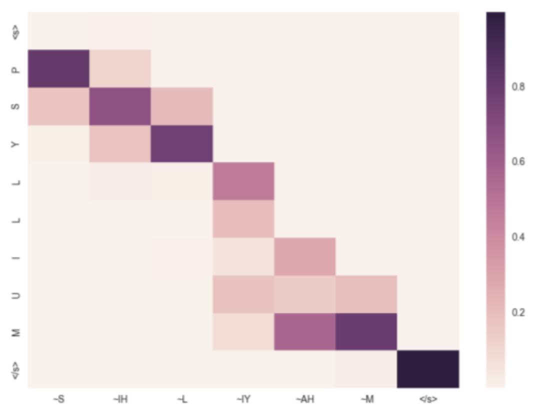

Figure 1. Learned attention window for generating phonemes from a grapheme-to-phoneme translation sequence-to-sequence network with attention.

The initial sequence-to-sequence networks worked by encoding all the information from the input sequence into a single ("thought") vector that the decoder RNN used as a strong context initialization to generate an output sequence. While the results have been impressive, this framework quickly breaks down as the length of the input sequence grows.

For translating long sentences or even paragraphs, holding all of the information in a single vector is untenable, regardless of how much capacity is in the network. With attention models, we can solve this problem in two important ways:

(1) the decoder RNN gets access to all the hidden state information of the encoder (and thereby the input sequence); and,

(2) more importantly, the network learns, given the current context and position of the word it is generating, which states – as a weighted sum – it should use to help it determine the word to generate. In this portion of the tutorial, we will show how CNTK makes this setup both conceptually easy and able to learn in a fast and robust manner.

2.2 Reasoning Network (ReasoNet)

The Reasoning Network (ReasoNet) [6] has achieved industry-leading results in reading comprehension. The ReasoNet model is currently the second-best performing model with ExactMatch (EM) and F1 scores being 73.42 and 81.75, respectively, on the Stanford Question Answering Dataset (SQuAD) [7].

ReasoNet attacks the problem of teaching a computer to read a document and then answer general questions pertaining to the contents of that document. Single-turn reasoning models use attention mechanisms **Error! Reference source not found.** with associated deep learning models to emphasize specific parts of the document which are relevant to the query. However, for many sophisticated comprehension tasks, a human reader often revisits some specific passage or the question to grasp a better understanding of the problem. Recent work has made use of multiple turns to infer the relation between query, document, and answer [9][10][11]. This approach has been demonstrated to produce superior results. We summarize the essence of the original paper by Shen et. Al. [6] in the remainder of this section.

Existing multi-turn models have a fixed number of hops or iterations in their inference, i.e., with predetermined reasoning depth, without regard to the complexity of each individual query or document. However, a human reader may read a document several times and stop when the question in mind has been adequately understood (reaching a certain level of confidence) or terminate after a certain number of tries.

ReasoNet tries to mimic the inference process of human readers. With a question in mind, ReasoNet reads a document repeatedly, each time focusing on different parts of the document until a satisfactory answer is found or formed. Moreover, unlike previous approaches using fixed numbers of hops or iterations, ReasoNet introduces a termination state in the inference. This state can decide whether to continue the inference to the next turn after digesting intermediate information, or to terminate the whole inference when it concludes that existing information is sufficient to yield an answer. The number of turns is dynamically modeled by both the document and the query, and will be learned automatically according to the difficulty of the problem.

One of the significant challenges ReasoNet faces is how to design an efficient training method, since the termination state is discrete and not connected to the final output. This prohibits the canonical back-propagation method from being directly applied to train ReasoNet. Inspired by [12][13], this challenge is tackled by a novel deep reinforcement learning method called Contrastive Reward (CR) to successfully train ReasoNet.

The challenge for CNTK (and most other deep learning toolkits) to implement Contractive Rewards is that the derivative of the loss cannot be directly computed based on the loss formula via Chain-Rule.

$$R = \sum_{t,a} \pi_{t,a} (\frac{r_{t,a}}{b} - 1)$$

where,

$$b = \sum_{t,a} \pi_{t,a} r_{t,a}$$

is referenced in the formula as a **constant**. Otherwise the backward **derivative of loss will always be zero**. Thus, we introduce a new operator, **ConstantRef**, in CNTK to tackle that challenge. With **ConstantRef**, we can just take the forward output of **b** in the computation of the loss while keeping it as a constant during backward propagation.

Unlike traditional reinforcement learning optimization methods using a global variable to capture rewards, CR utilizes an instance-based reward baseline assignment. Using **ConstantRef** enables us to factor part of the forward value in the loss computation.

Experiments show the superiority of CR in both training speed and accuracy. Finally, by accounting for a dynamic termination state during inference and applying the proposed deep reinforcement learning optimization method, ReasoNet can achieve the state-of-the-art results in machine comprehension datasets, including unstructured CNN and Daily Mail datasets, and a proposed structured Graph Reachability dataset.

We will show how to implement this powerful deep learning framework in CNTK and how to extend it to achieve impressive results in reading comprehension.

3. CNTK Overview

CNTK was originally designed for speech processing tasks but quickly developed into a full-featured deep learning toolkit. It was moved to GitHub [2] under the MIT License in 2016 and has evolved even further since then. Recently-released CNTK 2.0 provides support for both C++ and Python APIs and includes numerous examples and high-level building blocks. This toolkit was the key for Microsoft Research's recent breakthrough in speech recognition by reaching human parity in conversational speech recognition [14]. It has been extensively used internally at Microsoft for image, text, and speech data with each area benefiting from the built-in scalability.

There are certainly many deep learning toolkits already widely used in the deep learning community, including Caffe [15], Cognitive Toolkit (CNTK) [2], MxNet [16], TensorFlow [17] Theano [18], Torch [19], etc. However, we argue that CNTK has unique advantages over these toolkits in the combination of ease of use, speed, and scalability.

Recently, a study done by the Hong Kong Baptist University summarizes the performance of different deep learning toolkits [20]. Note the version of CNTK used in this analysis is 1.72 which is a much older version compared to the latest, and faster, 2.0 release. Here is a brief excerpt from their summary:

1. With a single GPU platform, Caffe, CNTK, and Torch perform better than MXNet and TensorFlow on FCNs; MXNet is outstanding in CNNs, especially on very large networks, while Caffe and CNTK also achieve good performance on smaller CNN networks; for LSTM, CNTK obtains excellent time efficiency, which is up to 5-10 times better than other toolkits.
2. With the parallelization of data during training, all the multi-GPU versions have a considerably higher throughput and the convergent speed is also accelerated. CNTK performs better scaling on FCN and AlexNet, while MXNet and Torch are outstanding in scaling CNNs.
3. CNTK outperforms Tensorflow in all categories.

4. CNTK Learners (Optimizers)

Performance is a key aspect of deep learning toolkits, especially with ever increasing network complexity and exponential growth in data sizes. However, to facilitate such large-scale computations, one needs to move computing to the cloud where any improvement in computing efficiency results in equivalent cost savings.

In this aspect, there are two optimizations that are critical to CNTK's superior scalability. Both are related to reducing communication costs between worker nodes. These are:

(1) 1-bit quantized Stochastic Gradient Descent (SGD) [3]; and

(2) Block momentum (using incremental block training) [4]

4.1 1-bit SGD Algorithm

There are different degrees of parallelism one can explore such as data parallelism, model parallelism, and layer parallelism. 1-bit SGD exploits data parallelism. The algorithm revolves around taking small samples of the minibatch data during training. Deep models consist of multiple layers (say several layers of fully connected projections, convolutions, recurrences, or a combination of these). Typically, each worker (GPU node in a GPU cluster) computes the sub-gradient for a given layer and for the minibatch sample assigned to the node. The sub-gradients across the different worker nodes are pooled in an All Reduce operation. However, this introduces a choke point which in the case of 1-bit SGD is overcome by distributing the sub-gradients aggregation across all the nodes. By distributing the gradients across all the nodes, each worker node can aggregate a subset of sub-gradients from other nodes. Repeating the process yields the sub-gradients aggregation across all the worker nodes.

While all the workers are efficiently used, we have increased the communication between the worker nodes. One way to reduce the overhead is to reduce the payload for each communication. As the name suggests, 1-bit SGD achieves this by quantizing the gradient to a single bit and carrying the quantization error over to the next minibatch. We will show how one can greatly leverage this optimization while training large models.

4.2 Block momentum

While 1-bit SGD reduces the payload to be communicated between worker nodes during each minibatch, Block Momentum operates at the block level and splits within each block. Blocks are defined as chunks of non-overlapping data partitions of the training set and each block is further partitioned into splits. There are two iterative stages within the block momentum algorithm: (a) intra-block parallel optimization (IBPO); and (b) block wise model-update filtering (BMUF). The details [4] will be covered in the tutorial.

4.2.1 IBPO

In this step, we randomly select a block of unprocessed data, and generate the block-wise model-update. Within each block the data is further partitioned into *N* splits and distributed to N different workers (e.g., GPU cards in a GPU cluster). These workers run in parallel to optimize local models with their own split of data. By leveraging data-parallelism within the block, the intra-block optimization can be conducted with different parallel algorithms. Finally, an aggregated model for a given block can be obtained by averaging optimized local models across the different sub-blocks provided by each of the different workers. from same initial model $\boldsymbol{W_g}(t-1)$. Each local model is optimized by mini-batch based SGD for one epoch of the split. Please be noted that local model optimization can also be conducted by 1-bit SGD, ASGD, etc, so that each worker can leverage multiple GPU cards to achieve better scalability. Finally, an aggregated model denoted as $\overline{\boldsymbol{W}}(t)$can be obtained by averaging N optimized local models.

4.2.2 BMUF

Instead of treating the output of IBPO, the aggregated model for each block is used to globally reach the final model directly, which is model averaging strategy. We treat global model updates as a block-level stochastic optimization process and propose a Blockwise Model-Update Filtering (BMUF) technique to stabilize the learning process. The method involves calculating the model update for each block using a technique like SGD with momentum trick except that the model update for the current iteration is blended with a weighted product of the previous iteration.

Instead of treating the $\overline{\boldsymbol{W}}(t)$ as the finetuned global model directly, which is the strategy of model averaging, we treat global model update operation as a block-level stochastic optimization process and propose a Blockwise Model-Update Filtering (BMUF) technique to stabilize the learning process.

First, we calculate the model-update vector resulting from current data block by subtract initial model from the aggregated model:

$$\boldsymbol{G}(t) = \overline{\boldsymbol{W}}(t) - \boldsymbol{W}_{\boldsymbol{g}}(t-1)$$

Then calculate global model-update vector, which is a weighted sum of $\boldsymbol{G}(t)$ and previous global model update vector:

$$\Delta(t) = \zeta_t \boldsymbol{G}(t) + \eta_t \Delta(t-1)$$

This formulation is similar with SGD with momentum trick, so we call ζ_t block learning rate and η_t block momentum. ζ_t and η_t can be set automatically by an empirical formulation. Then we update global model by:

$$\boldsymbol{W}(t) = \boldsymbol{W}(t-1) + \Delta(t)$$

Inspired by Nesterov momentum trick, we generate initial model for next data block by

$$\boldsymbol{W}_{\boldsymbol{g}}(t) = \boldsymbol{W}(t) + \eta_{t+1}\Delta(t)$$

Broadcast $\boldsymbol{W}_{\boldsymbol{g}}(t)$ to each worker and repeat IBPO and BMUF until all data blocks are processed, which is called one sweep. We can fine-tune the model by several sweeps until a stopping criterion is satisfied and obtain the final global model.

4.3 Discussion

1-SGD uses minibatch level parallelism and BMUF uses Block level parallelism with a momentum like update trick to overcome scale out challenges of simple model averaging, a wide variety of deep learning models can benefit. Due to the synchronization at mini-batch level, 1-bit is more sensitive to the I/O latency (because once a worker slows down due to I/O, the overall training speed of one-mini-batch slows down). BMUF on the other hand synchronizes at block level, thus its speed is less sensitive to burst I/O latency. However, 1-bit SGD can work as local model optimizer for BMUF for optimal scalability across multiple server / multiple GPU distributed computing environment.

5. Tutorial Session

In this tutorial, we assume the audience is familiar with the basics of deep learning. The session will focus specifically on text-based modeling of sequences. We encourage the audience to come prepared with the latest CNTK version installed on their machines, which can be done by following the instructions on the github site [2]. Tutorial details will be updated and archived. We will be using both slideware and Jupyter Python notebooks. The audience is expected to be familiar with Python and the Jupyter notebooks.

ACKNOWLEDGMENTS

We would like to thank Frank Seide, Principal Researcher at Microsoft Research and CNTK architect, Redmond USA and Qiang Huo, Principal Research Manager at Microsoft Research Asia (MSRA), Beijing, China for sharing their research materials on 1-bit SGD and BMUF algorithms, respectively. Additionally, we would like to thank Kai Chen from MSRA for BMUF algorithm summarization.

6. REFERENCES

[1] Sutskevar, I., Vinyals, O., and Le, Q.V.. "Sequence to sequence with neural networks," https://arxiv.org/pdf/1409.3215.pdf, 2014

[2] Cognitive Toolkit (formerly CNTK), https://github.com/Microsoft/CNTK/wiki

[3] Seide, F., Fu, H., Droppo, J., Li, G., and Yu, D., "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs," in *Proceedings of Interspeech*, 2014.

[4] K. Chen and Q. Huo, "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering," In *Proceedings of ICASSP*, 2016.

[5] Luong, M.T., Pham H. and Manning, C.D. Effective approaches to attention based neural machine translation. https://arxiv.org/abs/1508.04025.

[6] Shen, Y., Huang, P., Gao, J., and Chen, W., "ReasoNet: Learning to stop reading in machine comprehension," https://posenhuang.github.io/papers/reasonet_iclr_2017.pdf.

[7] The Stanford Question Answering Dataset (SQuAD), https://rajpurkar.github.io/SQuAD-explorer/

[8] Bahdanau, D., Cho, K., and Bengio. Y. "Neural machine translation by jointly learning to align and translate," *in Proceedings of the International Conference on Learning Representations*, 2015.

[9] Hill, F., Bordes, A., Chopra, S. and Weston. J., "The Goldilocks principle: Reading children's books with explicit memory representations," in *Proceedings of the International Conference on Learning Representations*, 2016.

[10] Dhingra, B, Liu, H., Cohen, W.W. and Salakhutdinov, R. "Gated-attention readers for text comprehension," *CoRR, abs/1606.01549*, 2016.

[11] Sordoni, A., Bachman, P., and Bengio, Y, "Iterative alternating neural attention for machine reading," CoRR, abs/1606.02245, 2016.

[12] Williams. R.J., "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, 8(3-4):229–256, 1992.

[13] Mnih, V., Heess, N., Graves, A et al., "Recurrent models of visual attention," In *Advances in Neural Information Processing Systems*, pp. 2204–2212, 2014.

[14] Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M., Stolcke, A., Yu, D., and Zweig, G., "Achieving Human Parity in Conversational Speech Recognition," https://arxiv.org/abs/1610.05256.

[15] Caffe, https://github.com/BVLC/caffe.

[16] MxNet, https://github.com/dmlc/mxnet

[17] Tensorflow, https://github.com/tensorflow/tensorflow

[18] Theano, https://github.com/Theano/Theano

[19] Torch, https://github.com/torch/torch7/wiki/Cheatsheet

[20] Shi, S, Wang, Q., Xu., P., and Chu, X., "Benchmarking state-of-the-art deep learning software tools," https://arxiv.org/pdf/1608.07249v6.pdf

[21] Tutorial session titled, Scalable deep document / sequence reasoning with Cognitive Toolkit https://github.com/Microsoft/CNTK/wiki/WWW-2017-Tutorial