PBML

logo

The Prague Bulletin of Mathematical Linguistics
NUMBER ??? JULY 2017 1–11

Parallelization of Neural Network Training for NLP with Hogwild!

Valentin Deyringer,[a b] Alexander Fraser,[a] Helmut Schmid,[a]
Tsuyoshi Okita[a]

[a] Centrum für Informations- und Sprachverarbeitung, LMU München
[b] Gini GmbH, München

Abstract

Neural Networks are prevalent in todays NLP research. Despite their success for different tasks, training time is relatively long. We use Hogwild! to counteract this phenomenon and show that it is a suitable method to speed up training Neural Networks of different architectures and complexity. For POS tagging and translation we report considerable speedups of training, especially for the latter. We show that Hogwild! can be an important tool for training complex NLP architectures.

1. Introduction

Many novel Machine Translation (MT) systems make use of Neural Networks (NNs) of different structure. In contrast to other machine learning methods, NNs are able to learn the relevant characteristics of the data independently (Bengio et al., 2013) and thus do not rely on handcrafted features which in turn requires expert knowledge and extensive study of the data basis. Backed by growing amounts of data available and increasing computational power, NNs have achieved remarkable results in different disciplines (Goodfellow et al., 2016). NNs have also proven to perform very well for MT (Cho et al., 2014; Sutskever et al., 2014).

These promising results of adopting NNs for MT and especially their capability of capturing the semantics of phrases (Cho et al., 2014) led to the emergence of a new branch of research referred to as Neural Machine Translation (NMT). This approach addresses the problem of translation with techniques solely based on NNs. A compara-

 Corresponding author: valentin@gini.net
Cite as: Valentin Deyringer, Alexander Fraser, Helmut Schmid, Tsuyoshi Okita. Parallelization of Neural Network Training for NLP with Hogwild!. The Prague Bulletin of Mathematical Linguistics No. ???, 2017, pp. 1–11. doi: HERE-WILL-BE-A-DOI-NUMBER

bly simple system has shown that an NMT system is able to reach near state-of-the-art results and even surpass a matured SMT system (Bahdanau et al., 2014).

A major drawback of NMT systems attenuating the positive findings is the long time needed to train the translation models. The most widely used gradient based optimization algorithms SGD, Adagrad (Duchi et al., 2011) Adadelta (Zeiler, 2012), Adam (Kingma and Ba, 2014) and RMSprop (Tieleman and Hinton, 2012) show good convergence properties for optimizing NNs and can be efficiently implemented by moving the underlying matrix operations to GPUs for heavy parallelization (e.g., with frameworks like theano (Bergstra et al., 2010) or Tensorflow (Abadi et al., 2016)). This approach obtains considerable speedups (Brown, 2014). There are several libraries for programming languages which offer a convenient interface for GPU programming in the context of NNs. Nowadays, almost all real world applications of bigger NN models involve computation on GPUs.

Dependent on quantity of training data and model size, which both generally have a positive effect on the resulting models quality when increased, training NMT systems reportedly still requires several days. Training times of 3 to 10 days are common (Cho et al., 2014; Sutskever et al., 2014; Bahdanau et al., 2014). In consequence, other ways to speed up the training are desirable. Besides from using GPUs, a way to shorten training times is parallelization on a higher level. This is not a trivial task as all of the optimization algorithms mentioned earlier are inherently sequential procedures. Nevertheless, there are generally two distinct approaches to achieve such parallelism, namely model parallelism and data parallelism. These approaches do not restrict the application of GPUs for the underlying matrix calculations and allow making use of the combined strength of several GPUs in a cluster.

The method of model parallelism distributes different computations performed on the same data onto multiple processors. The results are then merged in an appropriate way by a master process which also handles communication between processors as they are dependent on the results computed by the other processors. This technique is well suited for NNs due to their structure and is successfully implemented for the training of NMT models in (Sutskever et al., 2014). However, the work in hand is not concerned with model parallel approaches.

Data parallelism pursues a different approach where the processors perform the same operation on different data. In terms of optimization of NNs, this means that the training data is divided among the processors while shared parameters of the network are updated according to a suitable schedule. Data parallel training of NNs is not a trivial task and the commonly used optimization algorithms for training NNs are inherently iterative. Nevertheless, there are approaches in a data parallel fashion that allow parallelization of NN optimization, one of which is Hogwild! (Niu et al., 2011).

Hogwild! is an instance of a data parallel approach where updates to the global parameters are applied without locks. In this work we will show that Hogwild! can

be successfully applied to train NNs for NMT as well as for POS tagging. The main contribution of this work is the implementation of this algorithm for theano.[1]

The final results suggest that fitting NMT models with this asynchronous optimization technique has the potential to speed up the training process. It is found that Hogwild! is well suited for parallelized training of NMT models. As a secondary finding, an additional experiment shows that the same algorithms can be applied to NNs of various structures.

2. Approach

In SGD and descendant algorithms, updates are calculated with parameters estimated in the previous time step. Therefore these algorithms are sequential in nature. While basically applying the same update rule as standard SGD, in Hogwild!, separate updates for different batches of data are calculated on each working node based on parameters shared among all working nodes. These shared parameters are read and written to without any locks which usually are used to avoid simultaneous read/write operations on the same data in parallelized programs. As a result, the parameters possibly lack some updates computed on other processors that are yet to be applied and occasional overwrites may occur. However, assuming sparsity in the parameters updated for each training example, Niu et al. (2011) show that these downsides have negligible impact on the training procedure. With the results presented in Section 5, we demonstrate that this algorithm is also successfully applicable to NN training.

We implemented Hogwild! for the Theano framework using Python's multiprocessing module. After initializing the weights and defining the model's computational graph, several worker processes are spawned and local copies of the graph are compiled for each. This is necessary due to Theano functions not being thread safe. The subprocesses read batches of training data from a queue and when a new batch of data is processed, the globally shared variables are read and updates are calculated accordingly. These updates are then sent back to be applied to the shared parameters. In accordance with the update scheme of Hogwild! the shared parameters are read and written to without any locking. For more detail we refer the interested reader to our source code.

Especially in the case of using GPUs, data transfer to and from device memory may slow down training. However, in our experiments we did not find this to have a strong impact. Rather, due to Theano's GPU capabilities it is easy to utilize GPUs as working nodes and benefit from their strengths for matrix calculations.

[1] Our implementation of Hogwild! for Theano can be found at http://github.com/valentindey/async-train.

3. Related Work

Introducing the algorithm, Niu et al. (2011) compare Hogwild! to a version thereof with locking and the asynchronous optimization strategy presented by Langford et al. (2009) and demonstrate that Hogwild! obtains improved speed for several problems. The positive findings make it a natural choice for us to apply this algorithm to problems of NLP. However the examined tasks in the original paper only give little indication for applicability of Hogwild! for optimization of NNs as they are mostly used, especially in NLP.

The single machine C implementation of word2vec released as part of the work of Mikolov et al. (2013) also uses lock-free updates in the style of Hogwild!.[2] This method is clearly useful in this setting, as the problem typically is very sparse with large vocabulary sizes and only a few words affected at each update. Training word embeddings this way is based on a relatively simple NN architecture. We train more complex models with Hogwild! without such a clear notion of sparsity, and our approach allows us to use Hogwild! with flexibly defined complex NN architectures.

Feng et al. (2016) present an evaluation of different optimization algorithms on question answering tasks. Among other algorithms, implementations of EASGD/ EAMSGD (Zhang et al., 2015) and Downpour SGD (Dean et al., 2012) are evaluated. While showing promising results for parallelized gradient based optimization, their study lacks comparison with Hogwild! which we find is a suitable method for optimizing NNs.

Building on Hogwild!, Noel and Osindero (2014) introduce an optimization technique working on computing clusters like multiple CPU cores, multiple GPUs, or several machines. They implemented this for the Caffe framework (Jia et al., 2014) and show brief benchmarks on the MNIST (Lecun et al., 2009) and ImageNet (Deng et al., 2009) data sets, depicting promising results for applications using NNs. Inter alia, we take these findings as basis for porting Hogwild! to NLP problems.

The NMT systems marian[3] and OpenNMT[4] comprise implementations of asynchronous update strategies similar to Hogwild!. As marian is written in C++ and OpenNMT is built with the lua framework torch it is of interest to have a point of reference for a system implemented with Theano in python. Additionally, we are able to attain better speedup properties when increasing the number of used GPUs compared to the benchmarks listed on the website of marian.

[2]Their published results used the DistBelief framework (Dean et al., 2012) which follows a different parallelization paradigm since one of the goals is to overcome the constraint of having only small RAM in GPUs.

[3]https://marian-nmt.github.io

[4]http://opennmt.net

4. Tasks

POS Tagging Part of speech tagging is one of the most fundamental problems in NLP and can also be used to improve machine translation systems (Ueffing and Ney, 2003).

We study a German POS Tagging task using a self-defined NN model. Similar to Ling et al. (2015), our model represents words by the concatenation of a word embedding and a character-based word representation. The latter is computed with a bidirectional LSTM (BiLSTM) from the word's character sequence. Characters occurring only once and words occurring less than 10 times are replaced by a special symbol UNK. The forward/backward character LSTM processes the word suffix/prefix of length 10. The suffix/prefix is padded with padding symbols if the word length is below 10. The final states of the forward and backward LSTMs and the word embedding are concatenated. The resulting sequence of word representations is processed by a second BiLSTM whose forward and backward states are concatenated at each position. Each positional representation obtained in this way is linearly projected to an output layer with a softmax activation function over possible POS tags.

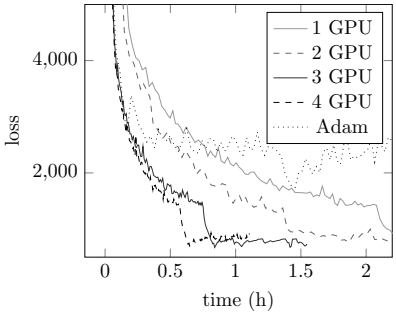
We use character embeddings of size 100, word embeddings of size 800, a character BiLSTM of size 400 for both directions and a deep word BiLSTM of size 800 for both directions with two layers of equal size. The training maximizes the log-likelihood of the correct tags. We decrease the initial learning rate of 0.03 by 0.0135 after each epoch. The character BiLSTMs are processed in parallel for all input words, but otherwise no batch processing is applied.

We trained our model on the German TIGER corpus (Brants et al., 2002) which is annotated with fine-grained POS tags that include additional annotations (e.g., number, gender, and case for nouns). We train on 40472 sentences and evaluate our models on 5,000 sentences held out from training.

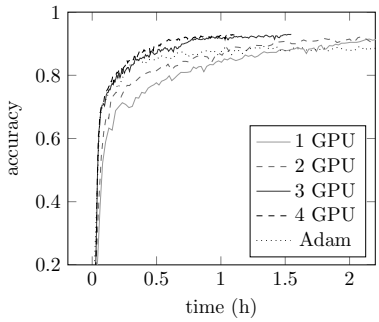
Neural Machine Translation NMT systems working on the sentence level make use of the so-called encoder-decoder architecture which transforms input sentences into vector representations via an encoder RNN and decodes the target sentences with a decoder RNN (Sutskever et al., 2014; Cho et al., 2014).

The NMT model used for this work is based on the dl4mt material[5]. It uses gated recurrent units (GRUs) as introduced by Cho et al. (2014) with 1,000 units for both, the encoder and the decoder, and applies an attention mechanism (Bahdanau et al., 2014). All data is tokenized in a preprocessing step with the tokenizing script from Moses (Koehn et al., 2007). The 15,000 most common words of the source and target languages are mapped to embeddings of size 100. All other words are treated as unknown and mapped to a shared embedding. We ignore sentences longer than 50

[5]The original dl4mt code can be found at https://github.com/nyu-dl/dl4mt-tutorial and the code for our adapted version lives at https://github.com/valentindey/pnmt



(a) decrease of loss during training



(b) accuracy on data held out from training

Figure 1: Training POS tagging models

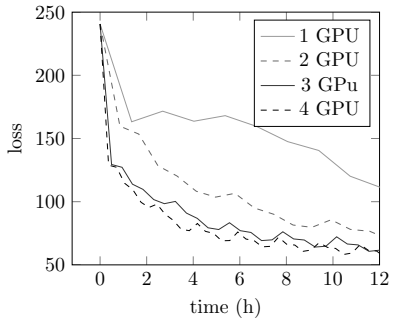
words. We use about 1.6 million sentences from the French/English Europarl corpus (Koehn, 2005) for training and an additional 10,000 sentences from the same corpus held out from training to monitor the training procedure (e.g. for early stopping). We maximize the log-likelihood of the training data with an initial learning rate of 0.1 that is not decreased throughout training. Gradients larger than 1.0 are clipped. For our NMT experiments, we use a batch size of 64.

5. Experimental Results

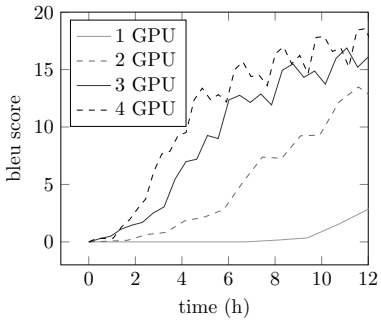
POS Tagging We trained the POS tagging model described in the previous section for 2 epochs on a machine running Ubuntu 16.04 equipped with Nvidia GeForce GTX 1080 units. As Figure 1 shows, this short training period is sufficient to show the performance gains through applying Hogwild!.

For comparison we also train the model with Adam on one GPU with the commonly used default hyperparameter values recommended by Kingma and Ba (2014).

Figure 1 shows the trajectories of training error and accuracy on the held-out data during training. There is a clear speedup when increasing the number of working nodes, and training with Hogwild! quickly becomes superior to training with Adam. The notches of training error in Figure 1a result from the decrease of learning rate after the first epoch. It is interesting to see that Adam fails to decrease the loss during training from a certain point on, probably oscillating around a local minimum while the models trained with Hogwild! decrease the loss further. The common strategy of initial training with an advanced optimization algorithm like Adam followed by SGD



(a) decrease of loss for a subset of the Europarl training data



(b) BLEU score for the Europarl data held out from training

Figure 2: Training NMT models

number of GPUs	time (h)	train loss	BLEU score held out data
1	26.89	68.4441	14.42
2	16.56	64.5328	14.46
3	9.24	64.1065	14.32
4	6.96	64.2586	14.40

Table 1: performance of NMT models after 90,000 updates

for fine tuning (e.g. Sercu et al., 2016) is thus no longer necessary when using Hogwild! and near optimal parameters are still found faster than when only using SGD.

Machine Translation The previous findings support the use of Hogwild! for the somewhat smaller problem of POS tagging where training is relatively fast. In the case of the more complex problem of NMT where training times of several days are common (Bahdanau et al., 2014; Sutskever et al., 2014; Cho et al., 2014) we see an even more marked improvement due to parallelization with Hogwild!.

The numbers in Table 1 exemplify the achieved speedup by listing the times needed for 90,000 updates and the respective model performance for different levels of concurrency. As expected, the time required to perform a fixed number of updates is decreasing approximately linearly dependent on the number of working nodes, i.e., GPUs. This indicates little to no influence of the negative side effects of Hogwild! discussed in Section 2. The models' performance in terms of BLEU scores is almost

unchanged and we can train competitive models in substantially shorter time. Applying Hogwild! for this problem in our eyes shows the most gain as this kind of model usually takes a very long time to train. However Figure 2 also suggests that there is an upper bound to the gains through parallelization which is almost reached when working on four GPUs, the same as with the POS tagging model.

Summary of NLP Results With Hogwild! we can speed up the training for both of the problems we considered, including different kinds of models. We can also see that using more GPUs has less impact on the training progress with each increment, which indicates an upper bound for positive effects from increased concurrency. Using three or four GPUs works very well for appropriately sized models and complex problems.

6. Conclusion

We have shown that Hogwild! is useful for training NNs of different architectures faster. It is meaningful to train models on multiple GPUs and CPU cores. The adverse effects of Hogwild! discussed in Section 2 are at a negligible level for the stated problems and show that the algorithm is suitable for training NNs for NLP tasks. We find that running Hogwild! on three to four GPU devices gives viable results for POS tagging and NMT.

With the release of our source code, we provide the means to easily use Hogwild! for other systems implemented in Theano. The seq2seq module in Tensorflow provides the GPU parallelization in a layerwise manner automatically when the LSTM consists of multiple layers. Hogwild! can be deployed on top of this setting easily as well.

Recently typical GPU environments have changed drastically. The Nvidia PASCAL architecture provides bigger graphics memory at a cheaper price point. This means that setups with four GPUs (or even eight or sixteen) are becoming widely accessible, an interesting contrast with massive CPU parallelization (Dean et al., 2012). In the computer vision community, parallel GPU architectures like the server we used are heavily used, while in the NLP community they are rare. Our results show that the NLP community should more strongly consider training with multiple parallel GPUs.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644402 (HimL). This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 640550).

Bibliography

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

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.

Bengio, Yoshua, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence, 35(8):1798–1828, 2013.

Bergstra, James, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A CPU and GPU math compiler in Python. In Proc. 9th Python in Science Conf, pages 1–7, 2010.

Brants, Sabine, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. The TIGER treebank. In Proceedings of the workshop on treebanks and linguistic theories, volume 168, 2002.

Brown, Larry. Accelerate Machine Learning with the cuDNN Deep Neural Network Library, 2014. URL https://devblogs.nvidia.com/parallelforall/accelerate-machine-learning-cudnn-deep-neural-network-library. [Online; accessed 2016-07-14].

Cho, Kyunghyun, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014.

Dean, Jeffrey, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In Advances in neural information processing systems, pages 1223–1231, 2012.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, pages 248–255. IEEE, 2009.

Duchi, John, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul): 2121–2159, 2011.

Feng, Minwei, Bing Xiang, and Bowen Zhou. Distributed deep learning for question answering. In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, pages 2413–2416. ACM, 2016.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep Learning. Book in preparation for MIT Press, 2016. URL http://www.deeplearningbook.org.

Jia, Yangqing, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for

fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia, pages 675–678. ACM, 2014.

Kingma, Diederik and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Koehn, Philipp. Europarl: A parallel corpus for statistical machine translation. In MT summit, volume 5, pages 79–86, 2005.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions, pages 177–180. Association for Computational Linguistics, 2007.

Langford, John, Alex J Smola, and Martin Zinkevich. Slow learners are fast. Advances in Neural Information Processing Systems, 22:2331–2339, 2009.

Lecun, Yann, Corinna Cortes, and Christopher JC Burges. The MNIST database of handwritten digits, 2009. 2009. URL http://yann.lecun.com/exdb/mnist.

Ling, Wang, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. Finding function in form: Compositional character models for open vocabulary word representation. arXiv preprint arXiv:1508.02096, 2015.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.

Niu, Feng, Benjamin Recht, Christopher Re, and Stephen Wright. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In Advances in Neural Information Processing Systems, pages 693–701, 2011.

Noel, Cyprien and Simon Osindero. Dogwild! - Distributed Hogwild for CPU & GPU. In NIPS Workshop on Distributed Machine Learning and Matrix Computations, 2014.

Sercu, Tom, Christian Puhrsch, Brian Kingsbury, and Yann LeCun. Very deep multilingual convolutional neural networks for LVCSR. In Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on, pages 4955–4959. IEEE, 2016.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Advances in neural information processing systems, pages 3104–3112, 2014.

Tieleman, Tijmen and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2), 2012.

Ueffing, Nicola and Hermann Ney. Using pos information for statistical machine translation into morphologically rich languages. In Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 1, pages 347–354. Association for Computational Linguistics, 2003.

Zeiler, Matthew D. ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701, 2012.

Zhang, Sixin, Anna E Choromanska, and Yann LeCun. Deep learning with elastic averaging SGD. In Advances in Neural Information Processing Systems, pages 685–693, 2015.

V. Deyringer, A. Fraser, H. Schmid, T. OkitaParallelization of NN Training for NLP with Hogwild! (1-

Address for correspondence:
Valentin Deyringer
valentin@gini.net
Gini GmbH
Prannerstraße 10, D-80333 Munich, Germany