Speed Up of Recurrent Neural Network Language Models With Sentence Independent Subsampling Stochastic Gradient Descent

Yangyang Shi[1,2], *Mei-Yuh Hwang*[1], *Kaisheng Yao*[1], *Martha Larson*[2]

[1]Online Service Division, Microsoft, China
[2]Intelligent Systems Department, Delft University of Technology, Netherlands
{yangyang.shi,m.a.larson}@tudelft.nl,{mehwang,kaisheng.yao}@microsoft.com

Abstract

Recurrent neural network based language models (RNNLM) have been demonstrated to outperform traditional n-gram language models in automatic speech recognition. However, the superior performance is obtained at the cost of expensive model training. In this paper, we propose a sentence-independent subsampling stochastic gradient descent algorithm (SIS-SGD) to speed up the training of RNNLM using parallel processing techniques under the sentence independent condition. The approach maps the process of training the overall model into stochastic gradient descent training of submodels. The update directions of the submodels are aggregated and used as the weight update for the whole model. In the experiments, synchronous and asynchronous SIS-SGD are implemented and compared. Using a multi-thread technique, the synchronous SIS-SGD can achieve a 3-fold speed up without losing performance in terms of word error rate (WER). When multi-processors are used, a nearly 11-fold speed up can be attained with a relative WER increase of only 3%.

Index Terms: Recurrent neural networks language models, stochastic gradient descent, subsampling, N-best list.

1. Introduction

Statistical language models play a crucial role in applications such as automatic speech recognition, machine translation and spelling correction. They model probability distributions over all possible word sequences in a language. Conventional n-gram language models have dominated automatic speech recognition for years due to their simplicity, good performance, and robustness. However, they suffer in face of sparse data and their ability to capture long-distance dependencies is insufficient.

It has been shown that both of these problems can be addressed by recurrent neural networks language models (RNNLM) [1, 2]. Conventional n-gram language models are discrete in nature, and despite smoothing techniques [3, 4], their ability to generalize remains limited. In contrast, better generalization can be achieved by RNNLMs, which map a discrete word to a point in a continuous space. The n-gram language models predict the next word by conditioning only on two or three previous words. However, in RNNLM, the recurrent connections between input layer and hidden layer, theoretically, can allow the information to cycle for an arbitrary length of time [1].

A critical drawback of RNNLMs is that they achieve their performance at the cost of expensive training. It is well known that it is difficult to use parallel processing to speed up an RNNLM without degrading performance because the SGD algorithm needs to update the weight of the RNNLM word by word. Because of the recurrent structure of an RNNLM, it is necessary to copy the activations of the hidden layer from the previous word to the current input layer. So that the copying process can be supported, RNNLMs are trained sequentially.

In order to make the parallel training of an RNNLM possible, we propose a sentence-independent subsampling stochastic gradient descent method (SIS-SGD) in this paper. SIS-SGD constrains the RNNLM by imposing the condition of sentence independence. The novel contribution of our approach is that it limits the history of a word in a way that is shown to both enable parallelization and also retain the ability of the RNNLM to benefit from information cycling in the network.

This paper is organized as follows. In the next section, we present the relevant related work. In Section 3, we explain the details of the SIS-SGD approach. In Section 4, we use the Penn Treebank and the Wall Street Journal data sets to show its effectiveness. The final section gives the conclusion.

2. Related work

In this section, we discuss the NNLM, RNNLM and SGD algorithms and speed up strategies that have been applied in the literature. Bengio *et al.* [5] proposed a neural probabilistic language model using feed forward neural networks, in which the input is the preceding $n-1$-gram. Their experiments showed that an NNLM can yield lower perplexity than conventional smoothed n-gram language models. Even though an NNLM is easier to speed up by parallel processing [6, 7], compared with RNNLMs, the NNLM has two drawbacks. One is that the preceding context information that the NNLM uses to predict next word is fixed. It is usually constrained to five to ten words, which is smaller than the context used for an RNNLM. NNLMs are shallower than RNNLMs. The recurrent hidden layer effectively equips an RNNLM with multiple hidden layers, which can learn more abstract representations of the input. This effect is confirmed by the fact that an NNLM approaches the performance of an RNNLM when additional hidden layers are added [8].

Most of the previous studies on speeding up NNLMs have focused on reducing the size weight matrix for learning from hidden layer to output layer. In [9], the output of an NNLM was constrained to a short list of most frequent words. Bengio *et al.* [10] used an adaptive importance sampling strategy to reduce the computation. Xu *et al.* [11] also used a subsampling strategy, but converted the multi-class prediction problem to a binary class prediction problem. However, the subsampling strategy is influenced by noisy samples, which make the model training unstable and difficult to adjust. Alternative to subsampling, in [12], it was proposed to use noise contrastive estimation to train NNLM. In [2], Mikolov proposed the class-based RNNLM, which used a simple and stable class trick to factorize the output layer in RNNLM. These methods already reduced the computation used for weight learning between hidden layer and

output layer to less than 1%. This class trick has been adopted in NNLM parallel training by [6]. However, the parallelization of the RNNLM has not been addressed due to the constraints of recurrent neural networks and of SGD. In this paper, we focus on the speed up of class-based RNNLMs via parallelization.

The online stochastic gradient descent (SGD) algorithm [13] has been extensively applied for neural networks, especially for large scale problems. Recently, [14] applied the Map-Reduce for SGD parallelization. Under Map-Reduce, a master machine distributes the computation of gradients to multiple slave machines, and then aggregates all the gradients to perform a global update. However, this algorithm requires many passes of data and many synchronization sweeps for convergence. In [15], the authors proposed to use a full-batch SGD in each slave machine. Each slave machine holds the data necessary to compute one update direction. In the end, the master machine averages all the directions to get a final update direction. In this way, it significantly reduces the cost of communication among processors. However, in each processor full-batch SGD is necessary and, in practice, the resulting performance turned out to be much worse than the SGD. In [16, 17, 18], a similar parallel strategy for the deep neural network learning is used. The parallel SGD proposed in [19] improved the algorithm in [15] by using online SGD in each slave processor rather than the full-batch SGD. In this paper, we use an algorithm similar to [19]: each slave uses SGD to get an update direction and the master aggregates all the directions to perform one iteration of parameter update. One major difference with our SIS-SGD is that we use subsampling instead of partitioning of the training data. Our experimental results reflect the importance of subsampling.

In order to reduce the latency among processors, previous work [20, 21, 6, 22, 23] has proposed a variant parallel SGD algorithm—the asynchronous SGD—under which each slave performs SGD to calculate each direction independently of the others and updates the shared parameters asynchronously. Specifically, each slave updates a delayed parameter vector. In [21] it was shown that the large delay, which was due to the fact that each slave handles a significant subset of the training data, in fact degraded the model quality. In other words, it is better if the slave can send parameter update directions to the master regularly after a small number of SGD steps are performed. In many applications, such overhead is acceptable. However, in RNNLM, the model has a huge number of parameters. The frequent communication of parameters among processors will dilute the gain from parallel processing. The parallelization approach in this paper attempts to minimize the overhead.

3. Sentence Independent Subsampling Stochastic Gradient Descent Algorithm

RNNLMs [1] generally have three layers: an input layer x, a hidden layer h and an output layer y. At each time t, the input vector $x(t)$ is constituted by the current word vector $w(t)$ as well as a copy $h(t-1)$ from the previous hidden neurons. The sigmoid function and softmax function are used as the activation functions in the hidden layer and output layer, respectively. In [2], the output layer consists of two groups of output units. The first group represents the vocabulary; hence there are V output units if V is the vocabulary size (UNK can be included). The second group represents classes; usually there are about 100-200 class units used. Each input word $w(t)$ is mapped to a unique class, which is connected to a constrained, fixed subset of output units in the first group that possibly can be activated, and thus, computed.

In the conventional RNNLM, the matrix weights between different layers are trained using the stochastic gradient descent algorithm. To apply RNNLM to large data sets, we propose SIS-SGD.

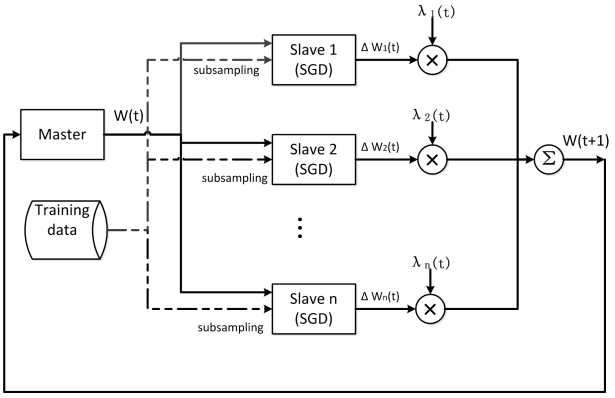

Figure 1: The SIS-SGD algorithm

The SIS-SGD algorithm uses a master/slave scheme to carry out message passing among different threads or processors. As shown in Figure 1, at each iteration t, all slaves start from the same parameters $W(t)$. Each slave randomly samples a subset of the training data as its own training set, and runs regular online SGD. After finishing its own samples, it computes the gradient direction from the initial $W(t)$ to the point at which it converged using its own data set. The directions are then weighted and aggregated by the master to compute the new parameter $W(t+1)$:

$$W(t+1) = W(t) + \frac{\alpha}{n} * \sum_{k=1}^{n} \lambda_k(t) \Delta W_k(t) \qquad (1)$$

where n is the number of slaves. $\Delta W_k(t)$ is the overall update direction that slave k learned from its training subset. $\lambda_k(t)$ is the weight for the direction from slave k.

In neural network training, the model benefits more from new patterns in the training data. In this paper, we apply the following heuristic method to determine the weight λ_k.

$$\lambda_k(t) = \frac{ln(E_k(t))}{\sum_{l=1}^{n}(ln(E_l(t)))}, \qquad (2)$$

where E_k is the entropy of the subset training data in slave k. We adopt the common practice of identifying new patterns of training data by high entropy. The directions calculated on the basis of these new patterns are given more weight in the final update direction.

3.1. Sentence Independence

SIS-SGD operates under the assumption of sentence independence. In a standard RNNLM, the prediction of next word is based on the history information including previous sentences. In fact, however, an infinite history is unnecessary in many applications. For example, in voice search, there is not necessarily a strong relationship among sentences. At the same time, although theoretically information in an RNNLM can cycle for an arbitrary length of time, in practice, the span of the back-propagation-through-time algorithm that is used to learn the weights, is limited. The study in [2] shows that after 4 to 5 steps, back propagation through time has very little benefit.

Further, due to the random sampling at each slave, sentence dependency has no real relevance. Section 4.2 will verify that assuming sentence independence has a barely noticeable effect on the the perplexity of the trained model. Although in this work we use sentence-level sampling, we expect our results to be generalizable to different units of approximately the same size, e.g., utterances.

3.2. Running SGD inside each Slave

As we mentioned before in SIS-SGD, each slave updates its model replica by the standard online SGD algorithm. It does not communicate with the master until it finishes training on its own subset of samples, and in this way avoids frequent messaging between slaves and master. The master waits for all slaves to finish one iteration of training. Within each slave, the direction is effectively computed via SGD, rather than mini-batch SGD or batch SGD.

3.3. Subsampling

An important principle enforced by the subsampling in SIS-SGD is that the subsampled sentences from each slave overlap with each other and that the union of all subsamples cover the original entire training set. Introducing overlaps instead of using parallelization regularizes parallel SGD. After each iteration of parameter update at the master, we re-shuffle the whole training data set and continue to do subsampling.

In SIS-SGD, the use of subsampling instead of disjoint partitioning has two motivations: One factor is determined by the special structure of RNNLM. In RNNLM, both the weight matrix from input word to hidden layer and hidden layer to output word, are sparse and huge. Partitioning leads to a situation in which each slave confronts a sparse data problem in producing an effective direction from its own subset. The other factor is the diversity of the directions from each slave. Each partition of the training data can be quite different from each other, which means that the directions from different partitions can conflict with each other. The aggregation of these diverse directions from disjoint partitions can result in a very small update for the parameter learning. The performance observed during our initial exploratory experimentation was so severely degraded in the case of disjoint partitions, that we did not consider the possibility further in the main experiments.

3.4. Practical Tricks for Robustness

In SIS-SGD, we use a smaller learning rate for the lower layer weight update [24]. Fundamentally, neural networks can be viewed as a combination of many different regressions. Theoretically weights from different layers should use different learning rates. However, in practice, only one learning rate is used for all the weights. In fact, the higher layer usually has a larger gradient than the lower layer. A smaller learning rate on the lower layer avoids divergence during training.

To accelerate the training of SIS-SGD, momentum [25] is applied in the master weight update in SIS-SGD. The basic approach is to interpolate current weight updates with the weight update history. The idea is that if the current updates are close to the historical updates, the momentum will increase the current updates. Otherwise, current updates will be reduced by the historical weight updates. This trick makes the training tolerant to noise and also speeds up training.

4. Experiments

4.1. Data Set

To evaluate the proposed SIS-SGD algorithm, we use two different data sets. The first one is Penn Treebank text data set (referred to as PTB). We use section 00-20 (972K word tokens) for training, section 21-22 (77K word tokens) for validation during the training, section 23-24 (86K word tokens) for testing. The vocabulary size for our experiment is 10K. We will measure only the perplexity on the Penn Treebank data set.

Table 1: The perplexity and WER results of the sentence dependent (DEP) RNNLM and sentence independent (INDEP) RNNLM reported on PTB and WSJ. RAND means that the training sequences were randomized.

model	PPL TEST	WER(%) DEV	WER(%) EVAL
PTB-RNN-DEP	138.7	-	-
PTB-RNN-INDEP	136.6	-	-
PTB-RNN-DEP-RAND	137.4	-	-
PTB-RNN-INDEP-RAND	135.3	-	-
WSJ-RNN-DEP	138.3	11.36	15.48
WSJ-RNN-INDEP	144.0	11.17	15.54
WSJ-RNN-DEP-RAND	139.4	10.88	15.66
WSJ-RNN-INDEP-RAND	140.6	11.15	15.69

The second data set is WSJ, for which we use 100-best speech recognition list from the DARPA WSJ'92 and WSJ'93 data sets, as used by [1, 26]. In the 100-best list set, 333 sentences are used as development data for tuning the interpolation of language models score and acoustic model score (DEV). The rest, 465 sentences, are used for evaluation (EVAL). The oracle WER for development data and evaluation data are 6.1% and 9.5%, respectively. The training corpus contains 37M words of running text from the NYT section of English Gigaword. The validation data set contains 186K words. A held-out set of 230K words is used for testing (TEST). The vocabulary size is 194K. Our experiments rescore the N-best lists for various RNNLM models and the baselines that we use for comparison.

All the RNNLM models in our experiments have 200 hidden neurons and are trained using 4-step back propagation through time (BPTT). The baseline model is obtained using the most recent version of RNNLM toolkit, in which the class-based SGD [2] is implemented. We use 100 word-classes in all of our experiments. The other baseline model is a Kneser-Ney 5-gram model; a 5-gram model has been shown to provide good n-gram model performance on this data [27] and is used in [1, 2, 28]

4.2. Sentence Independence Verification

As we discussed in previous sections, SIS-SGD imposes the condition of sentence independence. Table 1 verifies that disrupting the sequences of sentences does not degrade the performance of RNNLM. As a matter of fact, going from DEP to INDEP-RAND, there is a small improvement in perplexity on PTB and a small degradation on WSJ. These experiments provide evidence confirm the motivation for our SIS-SGD.

4.3. Speed up with Multi Threads

On the single machine, we can apply our proposed parallel SIS-SGD to train RNNLM via multi-thread architecture. The last row in Table 2 shows that the RNNLM model trained using 16 parallel SIS-SGD threads achieves the best training speed without losing performance in terms of word error rate (WER). Each slave thread trains a model replica on a subset of 12.5% of the whole training data. That is, the 16 threads together train twice ($16 * 12.5\% = 2$) as much the original data, yet with a 3-fold speedup. The price is the increased memory requirement, as now 16 replicas of the models must be accommodated in mem-

Table 2: The perplexity and N-best rescored WER results of RNNLM trained by parallel SIS-SGD via multi-thread architecture on WSJ data. The 'size' column stands for the relative size of the slave training subset, compared to the whole training data set. 'PPL' shows the perplexity results on the test data.

model	size (%)	time (hours)	PPL TEST	WER DEV(%)	WER EVAL(%)
KN5-base	100	-	174.5	12.09	17.30
RNNLM-base	100	77.5	140.6	11.15	15.69
8 threads	100	78.4	140.5	11.00	16.10
8 threads	50	65.9	138.5	10.77	15.84
8 threads	40	46.1	140.4	11.01	15.71
8 threads	25	33.6	144.4	11.07	15.99
16 threads	12.5	26.1	151.0	11.14	15.66

Table 3: The perplexity and WER results of RNNLMs trained by the parallel SIS-SGD on WSJ data with different numbers of processors. '*' denotes that the model is trained with more iterations (a lower stopping criterion)

model WSJ	size (%)	time (hours)	PPL TEST	WER DEV(%)	WER EVAL(%)
KN5-base		-	174.5	12.09	17.30
RNNLM-base	100	77.5	140.6	11.15	15.69
8p	25	33.7	146.9	11.19	15.70
16p	12.5	23.8	150.3	11.03	15.73
24p	8.3	27.2	148.9	11.01	15.67
32p	7.8	15.4	159.5	11.21	16.03
40p	5	19.7	154.7	11.10	15.92
48p	4.2	18.1	156.4	11.17	16.09
100p	2	11.2	172.5	11.41	16.16
*100p	2	27.4	142.0	11.14	15.62

ory, along with an additional copy on the master. In our case, our machine is an HP Z600 workstation with 12GB of RAM and 4 cores.

4.4. Speed up with Multi-Processors

The single machine has limited resources in terms of both cores and memory. In order to fully take advantage of a distributed hardware architecture, we implemented a multi-process version of the parallel SIS-SGD. The models given in Table 3 are trained on Microsoft HPC clusters via MPI processes.

In Table 3, the SIS-SGD models actually subsample the whole training data twice, except for the one with 32 processes, where the original train data is processed 2.5 times. When we used 32 processors, we achieved a 5-fold speed up with a relative 2% degradation on the evaluation data in terms of WER. When we used 100 processors, we could speed up the training by almost 7 times with a relative 3% increase of the WER in evaluation data. We note that the model can be further improved by an additional 16 hours of training iterations. As it is shown at the bottom of the table, SIS-SGD uses only one third of the standard SGD training time but results in a model with approximately the same performance.

4.5. Asynchronous SIS-SGD

Parallel SIS-SGD can also be implemented in an asynchronous manner. In [21, 6], the master immediately updates the model when it receives the direction from any slave. Afterwards, the master directly sends the updated model back to the corresponding slave. Basically, the master suffers extremely low inter-process latency as it does not have synchronization among slaves. However, asynchronous communication introduces model parameter delay among slaves during training. In [21], it is shown that a small delay to the master in fact improved the performance of the model as well as accelerated the training. Large delay from the slave to the master in fact degraded the quality of the model. That is, one would like to send updates to the master as frequently as possible.

Figure 2 shows the whole training data entropy of our proposed synchronous SIS-SGD, asynchronous SIS-SGD and baseline model against wall clock time. In both the synchronous and asynchronous SIS-SGD, we use 8 processors, each of which has 25% of the entire training data set. Asynchronous SIS-SGD is implemented in the round-robin way in order to obtain a deterministic result across different training runs. For this reason, it could have been faster if no round-robin constraint were enforced.

As we can see in Figure 2, the synchronous SIS-SGD converges faster than the asynchronous one. The probable reason is that with asynchronous SIS-SGD, model delay is too large. The delay in this case is one model learned from the 25% of the whole training data. Although asynchronous SIS-SGD reduces the inter-processor latency, the large model delay actually slows the convergence.

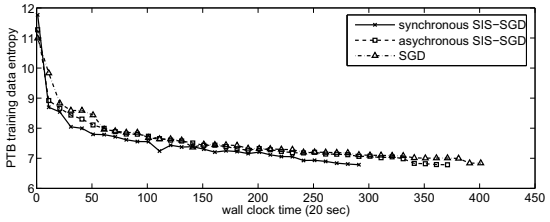

Figure 2: Training data entropy (Penn Treebank) of asynchronous parallel SIS-SGD, synchronous parallel SIS-SGD and standard SGD against the wall clock time. The parallel SIS-SGD use 8 processors, each of which process 25% of training data.

5. Conclusion

This paper has presented a sentence-independent subsampling stochastic gradient descent method (SIS-SGD), in which RNNLM is trained in parallel under the condition of sentence independence. In SIS-SGD, each slave trains a replica of the model on its own training data subset, which is generated by subsampling instead of disjoint partitioning. In order to minimize the data communication overhead, each slave sends its update to the master only after it finishes one pass of SGD learning on its own training subset. The master aggregates all the directions in a heuristic weighted way in order to perform the final model update. The experimental results showed that using a multi-thread technique, SIS-SGD can achieve a 3-fold speed up without losing performance. Using the multi-process technique and taking advantage of 100 processors, it can obtain a 7-fold speed up with a 3 percent relative degradation in terms of WER by taking advantage of 100 processors.

6. Acknowledgements

The authors are grateful for the insights of and discussions with Dong Yu, Frank Seide, Tomas Mikolov and Puyang Xu, in addition to the technical help from Yong Ni.

7. References

[1] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur, "Recurrent neural network based language model," in *Proceedings of Interspeech*, 2010, pp. 1045–1048.

[2] T. Mikolov, S. Kombrink, L. Burget, J. Cernocky, and S. Khudanpur, "Extensions of recurrent neural network language model," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2011, pp. 5528 –5531.

[3] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 1995, pp. 181–184.

[4] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech and Language*, vol. 13, no. 4, pp. 359–394, 1999.

[5] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, Mar. 2003.

[6] H.-K. J. Kuo, E. Arisoy, A. Emami, and P. Vozila, "Large scale hierarchical neural network language models," in *Proceedings of Interspeech*, 2012.

[7] H. Schwenk, "Efficient training of large neural networks for language modeling," in *Proceedings of IEEE International Joint Conference on Neural Networks*, pp. 3059–3064.

[8] E. Arisoy, T. N. Sainath, B. Kingsbury, and B. Ramabhadran, "Deep neural network language models," in *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, June 2012, pp. 20–28.

[9] H. Schwenk, "Continuous space language models," *Comput. Speech Lang.*, vol. 21, no. 3, pp. 492–518, Jul. 2007.

[10] Y. Bengio and J. S. Senecal, "Adaptive importance sampling to accelerate training of a neural probabilistic language model," *Trans. Neur. Netw.*, vol. 19, no. 4, pp. 713–722, Apr. 2008.

[11] P. Xu, A. Gunawardana, and S. Khudanpur, "Efficient subsampling for training complex language models," in *in Proceedings of EMNLP*, 2011, pp. 1128–1136.

[12] A. Mnih and Y. W. Teh, "A fast and simple algorithm for training neural probabilistic language models," in *Proceedings of the 29th International Conference on Machine Learning*, 2012, pp. 1751–1758.

[13] H. J. Kushner and G. G. Yin, *Stochastic Approximation Algorithm and Applications*. New York: Springer-Verlag, 1997.

[14] C. T. Chu, S. K. Kim, Y. A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, and K. Olukotun, "Map-Reduce for Machine Learning on Multicore," in *Advances in Neural Information Processing Systems*, 2006, pp. 281–288.

[15] G. Mann, R. Mcdonald, M. Mohri, N. Silberman, and D. D. Walker, "Efficient large-scale distributed training of conditional maximum entropy models," in *Advances in Neural Information Processing Systems*, 2009, pp. 1231–1239.

[16] D. Yu and L. Deng, "Deep convex net: A scalable architecture for speech pattern classification," in *Proceedings of Interspeech*, 2011, pp. 2285–2288.

[17] L. Deng, B. Hutchinson, and D. Yu, "Parallel training for deep stacking networks." in *Proceedings of Interspeech*, 2012.

[18] L. Deng, D. Yu, and J. C. Platt, "Scalable stacking and learning for building deep architectures," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2012, pp. 2133–2136.

[19] M. Zinkevich, M. Weimer, A. Smola, and L. Li, "Parallelized stochastic gradient descent," in *Advances in Neural Information Processing Systems 23*, 2010, pp. 2595–2603.

[20] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, 2012.

[21] M. Zinkevich, A. Smola, and J. Langford, "Slow Learners are Fast," in *Advances in Neural Information Processing Systems 22*, 2009, pp. 2331–2339.

[22] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," in *Proceedings of the 51th IEEE Conference on Decision and Control*, 2012, pp. 5451–5452.

[23] B. Recht, C. Re, S. J. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2011, pp. 693–701.

[24] Y. Lecun, L. Bottou, G. B. Orr, and K. R. Müller, "Efficient BackProp," in *Neural Networks—Tricks of the Trade*, ser. Lecture Notes in Computer Science, G. Orr and K. Müller, Eds. Springer Verlag, 1998, vol. 1524, pp. 5–50.

[25] R. Salomon and J. L. V. Hemmen, "Accelerating backpropagation through dynamic self-adaptation," *Neural Networks*, vol. 9, pp. 589–601, 1996.

[26] W. Wang and M. P. Harper, "The SuperARV language model: Investigating the effectiveness of tightly integrating multiple knowledge sources," in *Proceedings of Conference of Empirical Methods in Natural Language Processing*, 2002, pp. 238–247.

[27] D. Filimonov and M. Harper, "A joint language model with fine-grain syntactic tags," in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, 2009, pp. 1114–1123.

[28] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. ernock, "Empirical evaluation and combination of advanced language modeling techniques," in *Proceedings of Interspeech*, 2011, pp. 605–608.