COMBINING STOCHASTIC AVERAGE GRADIENT AND HESSIAN-FREE OPTIMIZATION FOR SEQUENCE TRAINING OF DEEP NEURAL NETWORKS

Pierre Dognin, Vaibhava Goel

IBM T.J. Watson Research Center
Yorktown Heights, NY 10598, USA
{pdognin, vgoel}@us.ibm.com

ABSTRACT

Minimum phone error (MPE) training of deep neural networks (DNN) is an effective technique for reducing word error rate of automatic speech recognition tasks. This training is often carried out using a Hessian-free (HF) quasi-Newton approach, although other methods such as stochastic gradient descent have also been applied successfully. In this paper we present a novel stochastic approach to HF sequence training inspired by recently proposed stochastic average gradient (SAG) method. SAG reuses gradient information from past updates, and consequently simulates the presence of more training data than is really observed for each model update. We extend SAG by dynamically weighting the contribution of previous gradients, and by combining it to a stochastic HF optimization. We term the resulting procedure DSAG-HF. Experimental results for training DNNs on 1500h of audio data show that compared to baseline HF training, DSAG-HF leads to better held-out MPE loss after each model parameter update, and converges to an overall better loss value. Furthermore, since each update in DSAG-HF takes place over smaller amount of data, this procedure converges in about half the time as baseline HF sequence training.

Index Terms— Deep Neural Network, Sequence Training, Stochastic Training, Hessian-free Optimization, Stochastic Average Gradient

1. INTRODUCTION

Deep neural networks (DNN) are gaining wide acceptance in automatic speech recognition (ASR) by allowing performance improvements previously unseen in state-of-the-art systems. However, new challenges arise from using DNNs in ASR. Finding the best procedure to train DNNs is an active area of research that is rendered more challenging by the availability of ever more training data.

A key component of the DNN training procedure is the so-called sequence training (ST) where the network parameters are optimized under a sequence classification criterion such as Minimum Phone Error (MPE) [1, 2]. This training is often carried out using a Hessian-free (HF) quasi-Newton approach, although other methods such as stochastic gradient descent (SGD) have also recently been applied successfully [3]. HF sequence training (HFST) uses a cross-entropy (CE) trained DNN as starting point, and is run until convergence, which is usually a computationally costly proposition. In this paper we present a novel stochastic approach to HFST inspired by recently proposed stochastic average gradient (SAG) [4] that alleviates the computational burden of HFST while allowing for better solutions, as measured by MPE losses on a held-out set.

This paper is organized as follows: First, second-order Hessian-free optimization is introduced in Section 2. Section 3 presents how to transform this second-order HF optimization into a stochastic Hessian-free (S-HF) optimization. Then, the concept of SAG is covered in Section 4, and it is extended and combined with HF sequence training in Section 5 where it is termed DSAG-HF for dynamic SAG Hessian-free optimization. Finally, experimental setup and results are presented in Section 6 and Section 7 respectively.

2. SECOND-ORDER HESSIAN-FREE OPTIMIZATION

Learning for DNN is a difficult task due to the issue of vanishing gradients, pathological objective function curves, and non-convex objective functions. Second-order methods can alleviate some of the burden from the first two challenges by leveraging curvature information of the loss function [5].

Let us consider a DNN with parameters $\boldsymbol{\theta}$ (weights and biases). The loss function $\mathcal{L}(\boldsymbol{\theta})$ can be approximated around $\boldsymbol{\theta}$ such that

$$\mathcal{L}(\boldsymbol{\theta}+\boldsymbol{\delta}) \approx \mathcal{L}(\boldsymbol{\theta}) + \nabla\mathcal{L}(\boldsymbol{\theta})^\top \boldsymbol{\delta} + \frac{1}{2}\boldsymbol{\delta}^\top \boldsymbol{B}(\boldsymbol{\theta})\boldsymbol{\delta}\,, \quad (1)$$

where the right-hand side is a quadratic approximation of $\mathcal{L}(\boldsymbol{\theta}+\boldsymbol{\delta})$, and $\nabla\mathcal{L}(\boldsymbol{\theta})$ is the gradient of the loss function at $\boldsymbol{\theta}$. If $\boldsymbol{B}(\boldsymbol{\theta})$ is equal to the Hessian $\boldsymbol{H}(\boldsymbol{\theta})$ (or an approximation of it), and $\boldsymbol{B}(\boldsymbol{\theta})$ is positive-semidefinite, then $\mathcal{L}(\boldsymbol{\theta})$ is locally convex and a minimum can be found. A direct optimization

of (1) would give the minimizer

$$\boldsymbol{\delta} = -\boldsymbol{B}^{-1}(\boldsymbol{\theta})\nabla\mathcal{L}(\boldsymbol{\theta}), \tag{2}$$

which is a solution to the system

$$\boldsymbol{B}(\boldsymbol{\theta})\boldsymbol{\delta} = -\nabla\mathcal{L}(\boldsymbol{\theta}). \tag{3}$$

Minimizing (1) w.r.t $\boldsymbol{\delta}$ is equivalent to solving the system in (3) for $\boldsymbol{\delta}$. Consequently, minimizing the loss $\mathcal{L}(\boldsymbol{\theta})$ can be done iteratively,

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k\boldsymbol{\delta}_k, \tag{4}$$

where $\alpha_k\boldsymbol{\delta}_k$ is a step taken in the Newton's direction $\boldsymbol{\delta}_k$, $\alpha_k \in [0, 1]$. The equivalent of (3) with iterative solutions, an equation that we will need later in this paper, becomes

$$\boldsymbol{B}(\boldsymbol{\theta}_k)\boldsymbol{\delta}_k = -\nabla\mathcal{L}(\boldsymbol{\theta}_k). \tag{5}$$

Since DNNs have large number of parameters, direct computation of the Hessian is impractical, but fortunately not necessary when using the *conjugate gradient* method, or CG. CG is an iterative algorithm used to find a solution direction $\boldsymbol{\delta}$ given a linear system like the one in (3). CG is usually not run to convergence but to some pre-determined point of relative solution improvement, and is often limited to a maximum number of iterations.

An important property of CG is that the Hessian is only needed in the context of *curvature-vector* product $\boldsymbol{B}(\boldsymbol{\theta})\boldsymbol{\delta}$ as seen in (3). It is therefore possible to define a Hessian-free optimization of (1) when using CG to find an approximate solution $\boldsymbol{\delta}$. Such Hessian-free optimization has been described and investigated thoroughly in [5]. In practice, the Hessian of the loss function $\mathcal{L}(\boldsymbol{\theta})$ is not positive-definite for DNNs, and a Gauss-Newton matrix $\boldsymbol{G}(\boldsymbol{\theta})$ is used instead. However $\boldsymbol{G}(\boldsymbol{\theta})$ is positive-*semidefinite*, and needs to be regularized by using $\boldsymbol{G}(\boldsymbol{\theta}) + \lambda\boldsymbol{I}$. λ is a damping factor which is adjusted heuristically.

3. STOCHASTIC HESSIAN-FREE OPTIMIZATION

The Hessian-free optimization procedure described in [5] consists of four steps run in sequence at each pass k over all the training data.

First, the gradient $\nabla\mathcal{L}(\boldsymbol{\theta}_k)$ is computed on all the training data. Second, CG is used to find iterative solutions $\boldsymbol{\delta}_k$ to (5). CG requires an initial search direction $\boldsymbol{\delta}_k^0$, which is commonly set to $\mathbf{0}$. However, faster convergence is attained when $\boldsymbol{\delta}_k^0$ is chosen to be $\sigma\boldsymbol{\delta}_{k-1}$, a scaled version ($\sigma = 0.95$) of the previous final search direction from CG. This acceleration technique is called δ-momentum. Furthermore, CG is usually truncated to a maximum of 250 iterations to balance the overall CG cost. Third, a "backtrace" over all CG iterates is done to find a potentially better candidate that minimizes the loss on a held-out set $\mathcal{L}^h()$. Finally, once a valid search direction $\boldsymbol{\delta}_k$ is found, a linear search is done to find the best α_k given the chosen direction, as seen in (4).

Curvature-products dominate the computational cost of CG after few passes over the data, and can become prohibitive when compared to the fixed cost for the gradient. Therefore, it is advised in [5] to use a 1% sample of the training data for curvature-products, balancing cost and accuracy of CG.

Given the description above, it is tempting to define a stochastic approach to this Hessian-free procedure. Intuitively, it does not appear necessary to process all the training data to update our model, like for stochastic gradient methods. Indeed, nothing in the HF training from [5] needs to be inherently run on all the training data. This seems especially true for the first few model updates. Therefore, one can easily define a "vanilla" stochastic HF procedure (S-HF) that would take a random subsample of the training set, perform one (or a few) model updates (4), then take a new random sample and iterate the process till convergence. We explore this S-HF process in addition to the DSAG-HF procedure as described in Section 4 and Section 5.

Parallel to our work, a stochastic Hessian-free optimization was investigated in [6] starting from the work in [5]. In [6], very short CG runs are performed (only 3-5 iterations), which has the consequence of having to re-tune the implementation from [5] since decisions about δ-momentum, Levenberg-Marquardt damping, size of gradient and curvature mini-batches need be revisited. Also, [6] integrates dropout which we do not address in our paper. Our approach is different in that it was influenced by ideas developed in [4] to improve SGD that we adapted to second-order Hessian-free optimization.

4. STOCHASTIC AVERAGE GRADIENT

In [4], a new technique called stochastic average gradient is proposed as a way to improve on conventional stochastic gradient (SG) techniques [7]. Writing the loss function as a sum of loss values over training samples,

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{T}\sum_{i=1}^{T} f_i(\boldsymbol{\theta}), \tag{6}$$

where T is the number of samples in the training data, and $f_i(\boldsymbol{\theta})$ the value of the loss for sample i. With stochastic gradient, parameters $\boldsymbol{\theta}_k$ are updated for each random sample i_k as

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \beta_k\nabla f_{i_k}(\boldsymbol{\theta}_k), \tag{7}$$

where β_k is a step size. In contrast, the stochastic average gradient method updates model parameters as

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \frac{\beta_k}{T}\sum_{i=1}^{T}\boldsymbol{\mu}_k^i, \tag{8}$$

where, for each training example i_k randomly selected,

$$\boldsymbol{\mu}_k^i = \begin{cases} \nabla f_i(\boldsymbol{\theta}_k) & \text{if } i = i_k, \\ \boldsymbol{\mu}_{k-1}^i & \text{otherwise.} \end{cases} \quad (9)$$

Effectively, SAG re-uses gradient information computed at previous iterations to help the convergence of its solution.

5. DYNAMIC STOCHASTIC AVERAGE GRADIENT WITH HESSIAN-FREE OPTIMIZATION

Reusing gradient information from previous iterations is a particularly powerful property of SAG that can be directly used in our second-order HF sequence training. Let the training data be split into B batches; then (3) can be rewritten equivalently as

$$\boldsymbol{B}(\boldsymbol{\theta})\boldsymbol{\delta} = -\sum_{b=1}^{B} \nabla \mathcal{L}^b(\boldsymbol{\theta}), \quad (10)$$

where $\nabla \mathcal{L}^b(\boldsymbol{\theta})$ is the gradient of the loss computed from samples in batch b. We propose to use the following SAG-like system for our iterative approach:

$$\boldsymbol{B}(\boldsymbol{\theta}_k)\boldsymbol{\delta}_k = -\sum_{b=1}^{B} \gamma_k^b \boldsymbol{\Lambda}_k^b, \quad (11)$$

where γ_k^b is a weighting factor at iteration k of the gradient computed for training batch b, and

$$\boldsymbol{\Lambda}_k^b = \begin{cases} \nabla \mathcal{L}^b(\boldsymbol{\theta}_k) & \text{if } b = b_k, \\ \boldsymbol{\Lambda}_{k-1}^b & \text{otherwise,} \end{cases} \quad (12)$$

starting with $\boldsymbol{\Lambda}_0^b = \mathbf{0}$, $\forall b$. In (11), each step $k = 0, 1, ...$ corresponds to an update over training data contained in batch b_k. However, for batches other than b_k, we use the gradient information from previous steps. To find an approximate solution to (11), we still use truncated CG as in regular HF optimization. At iteration k, the CG procedure is carried out on 1% of data from batch b_k.

A key aspect of this approach is to properly define the weighting factors γ_k^b. Indeed, these weights can make DSAG-HF behaves purely like a S-HF process if no gradient from previous batches is used, to an approximation of a full training if all gradients from previous batches are used uniformly. However, using uniform weights does not yield good results, and we found experimentally that those weights need be re-evaluated at each step k. We use an approach that keeps track of the held-out losses corresponding to each batch b as follows:

$$\mathcal{L}_k^b = \begin{cases} \mathcal{L}^h(\boldsymbol{\theta}_k) & \text{if } b = b_k, \\ \mathcal{L}_{k-1}^b & \text{otherwise.} \end{cases} \quad (13)$$

$\mathcal{L}^h(\boldsymbol{\theta}_k)$ is the loss on a held-out set. The weights γ_k^b are chosen to be

$$\gamma_k^b = \exp\left(\eta \left[\mathcal{L}_k^{b_k} - \mathcal{L}_k^b\right]\right) \quad (14)$$

where η is a tunable parameter that controls the exponentiation of our weights across batches. The weights are dynamically estimated to provide a loss function gradient before any CG iterations take place. In that sense, this is a dynamic schedule that controls the gradients' averaging weights. We term the resulting procedure dynamic stochastic average gradient with hessian-free (DSAG-HF) optimization.

For the first few steps k, (11) is only a *rough* approximation to (5); it is indeed closer to the S-HF procedure mentioned in Section 3. For the first step $k = 0$, a gradient computed only from data in b_k is used. For further steps, gradients computed from parameters $\boldsymbol{\theta}_k, \boldsymbol{\theta}_{k-1}, ...$, that are expected to be quite different from one another, are starting to be incorporated into the final gradient term. This is particularly true for a model's first few updates, as the held-out loss changes significantly across these first steps. However, as the parameters converge to a solution, the difference between the model parameters $\boldsymbol{\theta}_k, \boldsymbol{\theta}_{k-1}$, etc. becomes smaller, and (11) asymptotically converges to (5).

Therefore, DSAG-HF is able to simulate the use of more data than is actually observed for gradient computation at each batch b since we bring in gradients computed for previously observed batches. After all B batches have been observed, the final gradient contains contribution of virtually all the training data while only $1/B$ has been truly observed. This property enables our updates to be faster than for regular HF training, but also, not unlike for a SG process, it enables updates to be potentially quite different from regular HF training. In the rest of this paper, we compare DSAG-HF used for sequence training of DNNs to regular HF and stochastic HF sequence training.

6. EXPERIMENTAL SETUP

Our experiments are conducted using an IBM internal US English ASR task. The training set consists of 1500 hours of recorded audio. Training transcripts are obtained by decoding the audio using an existing large vocabulary continuous speech recognition system. Recordings that are deemed to be all silence or noise, and sentences decoded with very low confidence are excluded from the training data. The final training data amounts to 1.53M utterances for 3.7k speakers.

The first step in our DNN training procedure is to "pre-train" our DNN by growing it layer-wise under CE criterion. Once the final topology is reached, a CE model is trained fully using SGD with 5 passes over all the training data. This model (CE-5) is used as initial guess for the DNN parameters for subsequent HFST. Since HFST requires lattices, the CE-5 model is used to generate lattices on all the training data.

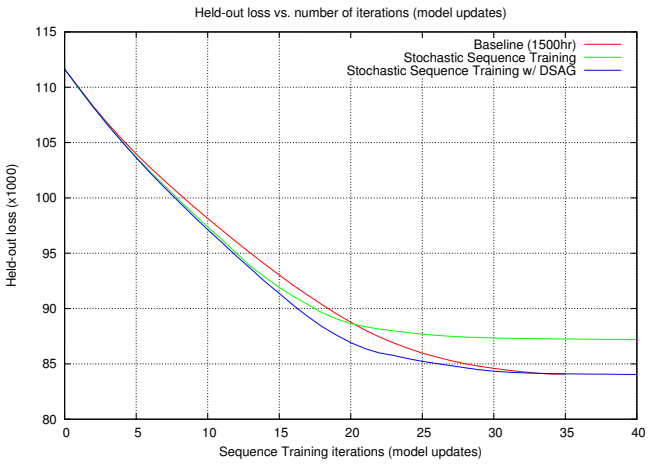

Fig. 1. Held-out MPE loss as a function of the number of updates for three approaches to sequence training: regular HF sequence training (Baseline) using 1500 hours of data per iteration, stochastic HF using batches of one 10th of training data, and DSAG-HF using the same data batches. All held-out MPE losses are computed on a 15 hours held-out set.

Our baseline model is trained using regular HFST, where each training iteration uses 1500h of training data. It is run until convergence as measured by a MPE loss on a held-out set of 15h of data set aside from our training data. The implementation of HFST for our baseline system is based on [1], with an HF procedure pretty similar to [5].

Our input features are post-LDA features from a 48 dimensional frames composed of 12 static cepstra and their 1st, 2nd, and 3rd time derivatives. These 48-dim features are projected onto a LDA space of 32 dimensions. Then, temporal context is created by splicing 9 successive frames of 32 dimensional post-LDA features to generate 288 dimensional features. These 288-dim features are the inputs to our DNNs. The topology of our DNNs is 6 layers of 1024 hidden units with sigmoid non-linearities. The output targets are 512 context-dependent states from building a context dependency tree from our training data.

All models are compared with measure of MPE loss over our held-out set mentioned previously. We also measure word error rates (WER) on a dev set composed of 684 utterances. Our sequence training infrastructure is identical for each of the three techniques. All computations (gradient, curvature product, etc.) are distributed over many machines and the distributed computing topology is kept identical for all our trainings.

7. EXPERIMENTAL RESULTS

The DSAG-HF, and S-HF procedures were performed starting from CE-5. The training data was split in 10 batches randomly sampled without replacement from the full training data, with each batch balanced for duration. After the first 10 steps, all batches are processed and the training is ensured to have seen all the training data. Further training steps rotate among the batches, and every 10 steps, the model will have seen all 1500h of training data again. All of the experiments presented here use DSAG-HF parameter $\eta = 0.7$.

Figure 1 shows the held-out MPE loss as a function of model updates for three approaches. Comparing S-HF to the baseline, we note that for the initial update iterations, the S-HF procedure results in better held-out loss than the baseline. However, S-HF saturates early and converges to a significantly worse held-out loss value. The S-HF and baseline loss curves cross at around iteration 20, never to recover. In contrast, the DSAG-HF procedure appears to consistently outperform the baseline, and converges to a better final held-out loss, as also seen in Table 1. In addition, we note that for both S-HF and the DSAG-HF procedures, the cost of each update is substantially less than baseline. Each stochastic update is indeed processing roughly a 10th of the training data.

Procedure (# iterations)	Loss (x1000)	CPU time (days)	WER (%)
Baseline			
10	98.15	1.11	14.0
20	88.77	3.72	12.4
30	84.60	7.49	11.9
34	84.08	9.06	11.8
Stochastic HF			
10	97.38	0.38	13.9
20	88.64	1.13	12.5
30	87.34	1.73	12.2
40	87.20	2.08	12.2
50	87.17	2.49	12.1
DSAG-HF			
10	97.13	0.42	14.0
20	86.91	1.46	12.3
30	84.34	2.88	12.0
40	84.04	3.95	11.9
50	83.88	4.74	11.8

Table 1. Held-out MPE loss, CPU time, and WER for regular HF (Baseline), Stochastic HF, and DSAG-HF. Values are reported for every 10 iterations up to convergence.

Table 1 shows the held-out loss at a few specific model update iterations for the three procedures depicted in Figure 1. It also provides the computation time (CPU time) up to those respective iterations, as well as a WER on a dev set.From this table, it is clear that DSAG-HF reaches a better held-out loss at about *half the processing time* than our baseline system. S-HF, despite being quite fast, converges to a much worse solution. The difference of processing time between S-HF and DSAG-HF is accounted for by the difference of CG itera-

tions required at each step. Indeed, DSAG-HF requires longer CG runs than S-HF. This points to some of the comments in [6] about the need to revisit our Levenberg-Marquardt style heuristic for damping as we notice a slight oscillation of the λ damping factor over our iterations.

In terms of WERs on the dev set, we observe the common trend in all our experimental results with sequence trained DNNs. The held-out losses are positively correlated to the WERs. Baseline and DSAG-HF models both converge around 11.8% WER while S-HF is reaching 12.1% at convergence. In fact, one can notice that for the sole purpose of WER, the models need not completely converge to provide the best WER. Small improvements in held-out loss do not completely translate into better WER.

One note worthy property of DSAG-HF is that no tuning is required except for our η parameter, since all scheduling of the weights are done based on the difference of held-out losses between the current and previous steps.

8. CONCLUSIONS

In this paper, we have introduced a novel stochastic Hessian-free sequence training for DNNs. This procedure, termed DSAG-HF, leverages gradient averaging as proposed recently in the stochastic average gradient approach, and carries out a Hessian-free CG-based optimization using these averaged gradients. Experimentally, we observe that DSAG-HF not only allows for faster convergence in computation time than regular HF sequence training (about half the time required), but also allows for faster held-out loss improvements, especially in the early updates.

Future work includes further exploration of the CG procedure including pre-conditioning and search characteristics in stochastic contexts.

9. ACKNOWLEDGMENTS

The authors would like to thank Brian Kingsbury for constructive discussions about sequence training. Peder Olsen, Etienne Marcheret, and Upendra Chaudhari for providing their help with this research work.

10. REFERENCES

[1] Brian Kingsbury, "lattice-based optimization of sequence classification criteria for neural-network acoustic modeling," in *ICASSP*, April 2009, pp. 3761–3764.

[2] Daniel Povey and Philip C. Woodland, "Minimum phone error and i-smoothing for improved discriminative training," in *ICASSP*, May 2002, pp. 105–108.

[3] Hang Su, Gang Li, Dong Yu, and Frank Seide, "error back propagation for sequence training of context-dependent deep networks for conversational speech transcription," in *ICASSP*, May 2013, pp. 6664–6668.

[4] Nicolas Le Roux, Mark Schmidt, and Francis Bach, "A stochastic gradient method with an exponential convergence rate for finite training sets," in *Advances in Neural Information Processing Systems 25*, pp. 2672–2680. NIPS, 2012.

[5] James Martens, "Deep learning via hessian-free optimization," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, Johannes Fürnkranz and Thorsten Joachims, Eds., Haifa, Israel, June 2010, pp. 735–742, Omnipress.

[6] Ryan Kiros, "Training neural networks with stochastic hessian-free optimization," in *International Conference on Learning Representations*, May 2013.

[7] Léon Bottou and Yann LeCun, "Large scale online learning," in *Advances in Neural Information Processing Systems 16*, Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, Eds. MIT Press, Cambridge, MA, 2004.