On the Efficiency of Recurrent Neural Network Optimization Algorithms

On the Efficiency of Recurrent Neural Network Optimization Algorithms

Ben Krause, Liang Lu, Iain Murray, Steve Renals
University of Edinburgh Department of Informatics
s1470305@sms.ed.ac.uk, llu@staffmail.ed.ac.uk,
i.murray@ed.ac.uk, s.renals@ed.ac.uk

Abstract

This study compares the sequential and parallel efficiency of training Recurrent Neural Networks (RNNs) with Hessian-free optimization versus a gradient descent variant. Experiments are performed using the long short term memory (LSTM) architecture and the newly proposed multiplicative LSTM (mLSTM) architecture. Results demonstrate a number of insights into these architectures and optimization algorithms, including that Hessian-free optimization has the potential for large efficiency gains in a highly parallel setup.

1 Introduction

Recurrent neural networks (RNNs) have achieved remarkable success in a number of machine learning tasks including speech recognition (Robinson et al., 1996; Graves et al., 2013; Sak et al., 2014), language modelling (Mikolov et al., 2010), image captioning (Vinyals et al., 2015; Xu et al., 2015), and machine translation (Sutskever et al., 2014; Cho et al., 2014). However, training RNNs remains a challenging optimization problem. The recursive nature of recurrent weight matrices can lead simple gradient based algorithms to become unstable, or converge very slowly. Heuristics such as gradient clipping (Pascanu et al., 2013) and specially designed architectures such as LSTM (Hochreiter and Schmidhuber, 1997) have made RNNs work on difficult problems. However, these methods generally require the use of small minibatches for efficient training, rendering them unable to fully take advantage of highly parallelizable hardware such as GPUs. The Hessian-free optimization algorithm has been used to train RNNs with much larger batch sizes (Martens and Sutskever, 2011; Sutskever et al., 2011), allowing greater use of parallelization. However, it is non-trivial to compare the efficiency of Hessian-free optimization with commonly used first order algorithms, because the efficiency ratio of methods with different batch sizes is highly dependent on the hardware being used. Additionally, second order methods tend to make larger but more computationally expensive updates than first order methods. We compare Hessian-free optimization to gradient descent with gradient norm clipping in both a stochastic and a large batch setting, using two metrics of efficiency: the first proportional to runtime on fully parallelizable hardware, and the second proportional to runtime on fully sequential hardware. Since the performance of RNN optimization is highly architecture dependent, this study focuses on the widely used LSTM architecture, and the newly proposed multiplicative LSTM architecture (Krause, 2015), which builds on the multiplicative RNN (Sutskever et al., 2011).

The standard RNN architecture operates over a sequence of input and output pairs of T timesteps, and predicts the output $O(t)$ using the input $I(t)$ as well as the previous hidden state $H(t-1)$, i.e.

$$H(t) = f\left(W_{hi}I(t) + W_{hh}H(t-1)\right) \tag{1}$$

$$O(t) = g\left(W_{oh}H(t)\right) \tag{2}$$

where W_{hi},W_{hh} and W_{oh} are weight matrices, and $f(\cdot)$ is the hidden layer nonlinear activation function (e.g. $\tanh$), and $g(\cdot)$ denotes the output layer activation function (e.g. $\mathrm{softmax}$). Training an RNN with stochastic gradient descent is difficult due to the vanishing/exploding gradient problem (Bengio et al., 1994; Hochreiter et al., 2001), as the gradient tends to decay or explode exponentially as it is back-propagated through time. This problem makes it difficult for RNNs to learn long-term dependencies, which has previously been addressed through choice of architecture and optimization algorithms. LSTMs are popular because their gated units control the transition of memory states, making them less susceptible to vanishing gradient problems. Multiplicative RNNs (mRNNs, Sutskever et al., 2011), on the other hand, rely on the strength of 2nd order algorithms to handle pathological curvature, and allow complex transitions between states by factorizing the recurrent weight matrix in a standard RNN as

$$W_{hh} = W_{hm} \, \mathrm{diag}(W_{mi} I(t)) W_{mh}. \tag{3}$$

A multiplicative LSTM – a combination of mRNNs and LSTM – was shown to be especially successful at the task of character level modelling (Krause, 2015). This paper presents further insights on optimizing these RNN architectures.

2 First-order and second-order optimization of RNNs

A learning algorithm for RNNs must be able to compensate for vanishing and exploding gradients to learn long range dependencies and keep updates stable. A simple approach is first-order stochastic gradient descent with gradient norm clipping (Pascanu et al., 2013), which reduces the step size in the direction of the gradient when the magnitude of the gradient exceeds a certain threshold. While this method only explicitly addresses the exploding gradient problem, an LSTM architecture can be used in combination with this optimization method to address vanishing gradients.

Second-order optimization approaches take advantage of the full curvature matrix, which can be used to appropriately rescale the gradient according to the second derivatives. As the second derivatives are expected to decay and explode at a similar rate to first derivatives (Martens, 2010), they can provide a sensible rescaling to account for the training difficulties of RNNs. In Newton's method, the error function $\mathcal{F}(\cdot)$ is approximated by the second order Taylor series expansion around the current estimate of the model parameters θ, and the update of the model parameters $\Delta\theta$ is obtained as

$$\Delta\theta = -H(\mathcal{F}(\theta))^{-1} \nabla \mathcal{F}(\theta). \tag{4}$$

However, storing and inverting the curvature matrix $H(\mathcal{F}(\theta))$ is inefficient and computationally infeasible for large neural networks. Hessian-free optimization uses the conjugate gradient method to exploit the potentially sparse and low rank structure of the curvature matrix to solve the linear system

$$H(\mathcal{F}(\theta))\Delta\theta = -\nabla \mathcal{F}(\theta), \tag{5}$$

instead of directly computing Eq. (4). Because the conjugate gradient method is only guaranteed to converge for a positive semi-definite matrix, the Hessian matrix $H(\mathcal{F}(\theta))$ can be approximated using the Gauss–Newton matrix (G) which only includes positive curvature. In this case, matrix-vector products (Gv) required by the conjugate gradient method can be computed efficiently using the method derived by Schraudolph (2002). However, the matrix G is poorly conditioned, which makes conjugate gradients converge far outside the region where the local Taylor series approximation is accurate. Damping methods such as Tikhonov damping and structural damping can be used to improve the condition of the matrix G (Martens and Sutskever, 2011).

Batch sizes: Batching can be used to train RNNs by splitting the training data into sequences of fixed length. For both first and second order algorithms, B_g sequences are used in parallel to approximate the gradient. In Hessian-free optimization, B_c training examples must also be used for computing Gv products during conjugate gradient iterations. Since there are many conjugate gradient iterations per gradient computation, B_g is typically several times larger than B_c because it is relatively cheap to get a more accurate approximation to the gradient. While first order methods must rely on many small updates to converge, Hessian-free optimization can use larger batch sizes. An accurate local approximation of the curvature allows large updates, justifying the cost of processing a large batch.

Batch size greatly affects the efficiency and effectiveness of RNN optimization. Small batch sizes give noisy estimates of the derivative information cheaply, which can be very efficient if there is limited variance in the gradient across training examples. Larger batches give better derivative estimates, but also require more computation. However, the cost of this extra computation can be relatively cheap, as batch updates are easily parallelized.

Optimization methods for neural networks are traditionally evaluated by measuring the convergence rate over the number of training epochs. However, this is not a sensible approach for comparing first and second order methods as well as approaches with different batch sizes, as the number of epochs will not be a consistent measure of the computation time required. Furthermore, the ratio of computational efficiency of algorithms of different batch sizes will depend highly on the hardware being used. For these reasons, we derive two new units of training computation proportional to computation time using fully sequential and fully parallelized operations for an RNN of fixed size.

Sequential computation cost measures: In a computer that can only perform sequential operations, the computation time will be proportional to the number of floating point operations performed. We reflect this extreme with the number of batch dependent forward passes, P_D, which is proportional to the amount of computation in a forward pass through a single training sequence of fixed length for an RNN of fixed size. For a first order gradient algorithm, one forward pass and one gradient computation must be performed for I_t training iterations. Since a gradient computation requires approximately twice the computation of a forward pass,

$$P_D = 3I_tB_g, \quad \text{for first order methods.}$$

For Hessian-free optimization, the gradient must also be computed at each training iteration, and additionally, a Gv product must be performed for I_c conjugate gradient iterations. A Gv product requires approximately 4 times the computation of a forward pass, so

$$P_D = 3I_tB_g + 4I_cB_c, \quad \text{for Hessian-free methods.}$$

Parallel computation cost measures: On the opposite end of the spectrum would be a computer that has enough processors to fully parallelize batches so that computation time is independent of batch size. To reflect this scenario, we measure the number of batch independent forward passes, P_I, in which the batch size terms become irrelevant:

$$P_I = 3I_t, \quad \text{for first order methods}$$
$$P_I = 3I_t + 4I_c, \quad \text{for Hessian-free methods.}$$

3 Experiments

Experiments were carried out to compare the efficiency and effectiveness of Hessian-free optimization compared with a variant of gradient descent with norm clipping for small and large batch sizes for LSTM and mLSTM RNN architectures, trained on sequences of length 200. We compared stochastic Hessian-free optimization (Kiros, 2013) (SHF) with $B_g = 320$ and $B_c = 32$, large batch Hessian-free optimization (BHF) with $B_g = 14000$ and $B_c = 3500$, stochastic gradient descent (SGD) with $B_g = 32$, and large batch gradient descent (BGD) with $B_g = 3500$. The training and validation error were measured over training as a function of our computational cost measures P_I and P_D. We benchmarked these architectures and optimization methods with the task of character level sequence modelling. Character level models are challenging to optimize, and push the limits of fitting algorithms, making them a good option for benchmarking advanced optimization methods. These experiments were run on a subset of the Penn Treebank corpus (`catalog.ldc.upenn.edu/LDC99T42`), containing the first 2.8 million characters for training, and approximately 200,000 for validation (same data set as used in (Krause, 2015)). These experiments used the Penn Treebank corpus in its raw form, with no preprocessing, and numbers and rare words included. The LSTM and mLSTM had approximately 220,000 and 215,000 trainable parameters respectively.

Results: Convergence plots for training error as a function of training computation are presented in Figure 1. Validation error was also obtained, however it is not included in the graphs. SHF and BGD for LSTM were too inefficient to fully converge in the time allotted. mLSTM with stochastic Hessian-free optimization achieved the best validation error of 1.82 bits/char. Other runs for both mLSTM and LSTM that converged all had validation errors that ranged from 1.89-1.93 bits/char.

4 Discussion

There are several important insights from these experiments. The first is that Hessian-free optimization was far more efficient than gradient descent at the batch level, regardless of the architecture being

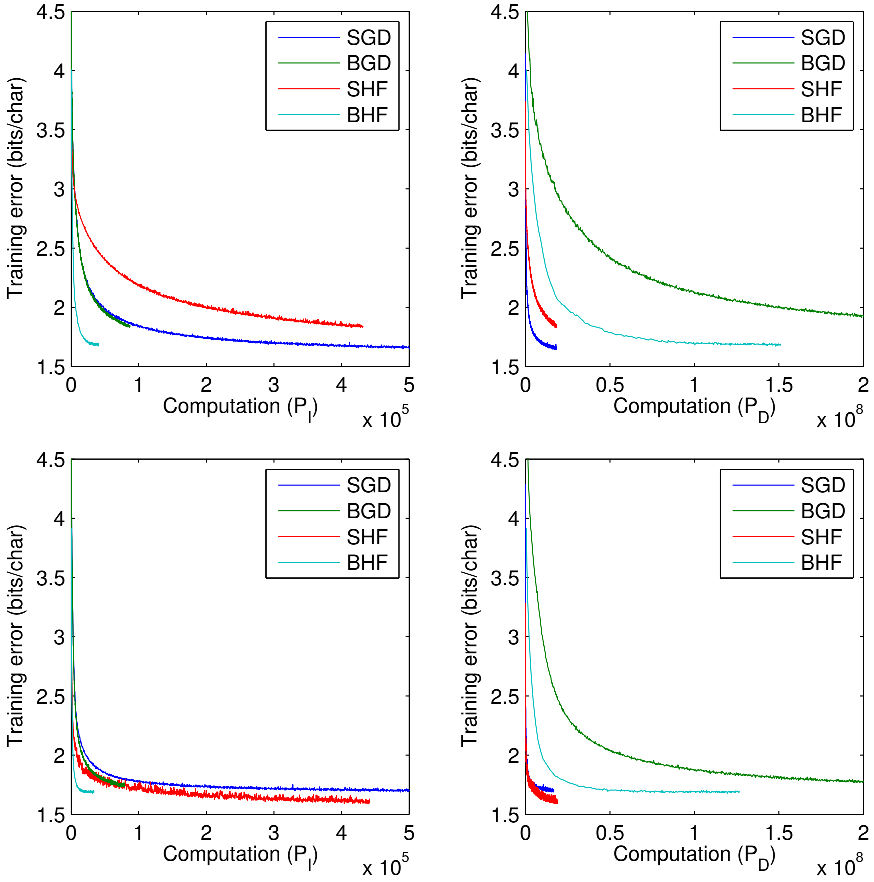

Figure 1: Training errors (bits/char) for LSTM over batch independent (upper left) and batch dependent (upper right) computation, and mLSTM over batch independent (lower left) and batch dependent (lower right) computation.

trained. This suggests that the curvature in the error surface was significant, and that enforcing conjugacy of the search directions with respect to the curvature matrix was greatly increasing the convergence rate. For LSTM, there was very little difference in the batch independent convergence rate of BGD and SGD, suggesting that the gradient of LSTM can be well approximated with a few examples. This may partially explain why LSTM has traditionally been easy to optimize with first order methods. In mLSTM on the other hand, the batch independent convergence rate was similar for BGD and SGD at the beginning of training, but BGD began to converge faster fairly early on. This suggests that the gradient of mLSTM has a higher variance across training examples than LSTM.

There was an enormous difference in the performance of SHF optimization for LSTM and mLSTM. In LSTM, SHF failed to fully converge, suggesting a greater variance in the curvature across training examples for LSTM. In roughly the same number of training iteration, SHF applied to mLSTM achieved by far the best training and generalization results in these experiments. It is likely that the large stochastic updates observed in this run created a sort of weight noise that improved regularization, and possibly helped avoid local minima.

An important comparison can be drawn from SGD, which is used in nearly all LSTM experiments, and BHF, which was more than an order of magnitude more efficient on this dataset with full parallelization. While full parallelization is not a realistic goal, large compute clusters and multi GPU machines can often come close to this. Overall, these experiments highlight several important differences in training LSTMs and mLSTMs with first and second order algorithms. The results suggest that Hessian-free methods have the potential to be much more efficient on highly parallelizable hardware.

Acknowledgements

This work was supported by a donation from MERL.

References

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the Conference on Empirical Methods on Natural Language Processing*.

Graves, A., rahman Mohamed, A., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Proceddings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649. IEEE.

Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. (2001). Gradient flow in recurrent nets: The difficulty of learning long-term dependencies. In Kremer and Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9:1735–1780.

Kiros, R. (2013). Training neural networks with stochastic Hessian-free optimization. *arXiv preprint arXiv:1301.3641*.

Krause, B. (2015). Optimizing and contrasting recurrent neural network architectures. *arXiv preprint arXiv:1510.04953*.

Martens, J. (2010). Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 735–742.

Martens, J. and Sutskever, I. (2011). Learning recurrent neural networks with Hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1033–1040.

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Proceedings of the Annual Conference of the International Speech Communication Association*, pages 1045–1048.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*.

Robinson, T., Hochberg, M., and Renals, S. (1996). The use of recurrent networks in continuous speech recognition. In Lee, C. H., Paliwal, K. K., and Soong, F. K., editors, *Automatic Speech and Speaker Recognition – Advanced Topics*, pages 233–258. Kluwer Academic Publishers.

Sak, H., Senior, A., and Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Proceedings of the Annual Conference of International Speech Communication Association*.

Schraudolph, N. N. (2002). Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738.

Sutskever, I., Martens, J., and Hinton, G. E. (2011). Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the Computer Vison and Pattern Recognition*.

Xu, K., Ba, J., Kiros, R., Courville, A., Salakhutdinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the International Conference on Machine Learning*.