Sparse Deep Neural Networks for Embedded Intelligence

Jia Bi, Steve R. Gunn
School of Electronics and Computer Science
University of Southampton, Southampton, United Kingdom
Email: jb4e14@soton.ac.uk, srg@ecs.soton.ac.uk

Abstract—Deep learning is becoming more widespread due to its power in solving complex classification problems. However, deep learning models often require large memory and energy consumption, which may prevent them from being deployed effectively on embedded platforms, limiting their application. This work addresses the problem of memory requirements by proposing a regularization approach to compress the memory footprint of the models. It is shown that the sparsity-inducing regularization problem can be solved effectively using an enhanced stochastic variance reduced gradient optimization approach. Experimental evaluation of our approach shows that it can reduce the memory requirements both in the convolutional and fully connected layers by up to 300× without affecting overall test accuracy.

I. Introduction

Artificial intelligence is finding wider application across a number of domains where computational resources can vary from large data centres to mobile devices. However, state-of-the-art techniques such as deep learning [1] require significant resources, including large memory requirements and energy consumption. Reducing the size of the deep learning model to a compact model that has a small memory footprint without compromising its performance is a desirable research aim to address the challenges for deploying these leading approaches on mobile and embedded devices.

Given training labels \( y_1, y_2, \ldots, y_n \) and corresponding input data \( x_1, x_2, \ldots, x_n \), the regularized objective function to estimate the weights, \( w \), in the model, \( f \), is defined by

\[
\min_w \frac{1}{n} \sum_{i=1}^{n} \mathcal{L} (y_i, f(x_i; w)) + \gamma R(w),
\]

where \( \mathcal{L} \) is a loss function, \( R(w) \) is the regularizer and \( \gamma \) is a hyper-parameter controlling the balance between these two objectives. This optimization problem can be strongly convex or non-convex [2] and is largely determined by the form of the model. In the case of deep neural networks the problem will be non-convex.

The choice of regulariser and regularization parameter is important in order to control the generalisation ability of the model. One additional benefit of a regularizer is that it can be used to enforce certain properties on the model parameters. For example, the choice of an \( \ell_1 \) regularizer as used in a number of learning approaches including Lasso, group Lasso [3], and logistic Lasso [4] enables model compression by encouraging weights to be zero.

To find the optimal solution of Eq. 1, stochastic variance reduced gradient (SVRG) is preferable to stochastic gradient descent (SGD) as SGD optimization has slow convergence asymptotically due to noise [5]. However, directly applying this to a deep neural network objective with a regulariser with non-continuous derivatives such as \( \ell_1 \) regularization presents challenges for SVRG. During optimization the discontinuity can effect the robustness of the SVRG optimization algorithm, which slows down the convergence. To address this problem, we propose our compression method \( \textit{Sparse SVRG} \) (SSVRG) by introducing an adaptive learning rate, as an approach to deal with the discontinuity in the derivative and a hyper-parameter to control the balance of the variance reduction. We show that this can improve SVRG optimization for non-convex functions such as those resulting from sparse \( \ell_1 \) regularization.

II. Related works

There are two main approaches to reducing the memory footprint of models by either reducing the number of weights in the model and/or reducing the number of bits to store each weight. Regularization-based approaches are restricted to the former whereas other approaches are capable of weight quantization.

A. Regularization-based compression methods

Weight decay as a penalty was one of first techniques for regularization that aims to improve generalization [6], by encouraging a reduction in the magnitude of the weight vector of the neural network [7]. However, the regularization term in weight decay using an \( \ell_2 \)-norm does not produce sparsity and hence there is no compression. In contrast, Lasso estimators can penalize weights to zero [8]. These approaches are based on \( \ell_1 \) regularization and its ability to reduce weights to zero in the training process. However, the results have not always been effective at high compression rates with some results only reducing the weights by \( 4\times \) and with inferior accuracy [9]. To directly construct sparse models, a simple regularization
operator using the $\ell_0$-norm has been proposed [9], but the method faces challenges when used to train deep neural networks by using stochastic gradient-based optimization. $\ell_1$ regularization has been shown to achieve good compression [10], but the choice of SGD optimization has slow asymptotical convergence due to the inherent variance [5]. Group lasso [11] is an efficient regularization-based method for training sparse learning structures. This concept was extended to include a group penalty for network connections with lasso which creates a combined $\ell_{2,1}$ regularizer to remove neurons [12], which was shown to reduce the number of weights; a compression rate of $3\times$ was achieved for a small increase in error. Another approach proposed group-wise brain damage to prune the convolutional kernel tensor in a group-wise fashion [13], and was shown to speed-up the computation of convolutional layer training. The “dropout” approach which refers to dropping out neurons from visible and hidden layers in the neural network during training can also be shown to be a form of regularization [14]. Sparse Variational Dropout (Sparse VD) extends the concept of “dropout” by providing a different dropout rate for each layer of the model [15] and is shown to have competitive compression performance.

B. Other compression methods

Other methods have been proposed to reduce the number of weights in the models. One method mainly explored pruning which is a direct approach to remove small values of connection and emphasize the important connections with large weight values in all layers of the network [16]. However, a disadvantage is that after pruning the networks needs to be retrained. A similar pruning method, dynamic network surgery (DNS) combines pruning and splicing to remove unimportant weights [17]. Another idea from matrix factorization can be applied to compress parameters in models by finding a low-rank approximation of the weight matrix [18]. However, in practice whilst it improves computation performance, it does not significantly reduce memory requirements. Weight sharing aims to approximate many weights by a single weight and an index table to provide compression. HashedNets is a weight sharing method that bins network connections into hash buckets by a randomized hash function [19]. Another weight sharing approach uses k-means clustering to identify the shared weights for each layer of a trained network [16].

Weight quantization for reducing the bit-width to store each weight is another approach to reduce memory requirements of models. It can successfully condense CaffeNet and SqueezeNet to 8-bits with only slight accuracy loss [20]. One refinement adapts the bit-depth for each layer quantizing the sparse weight matrix to be an index which encodes in 8-bit for convolutional layers and 5-bit for fully connected layers with only 1% loss in accuracy [16]. In a more extreme case it is possible to use 1-bit to represent the weights with the added advantage that computation can then be done with binary operations, but the accuracy loss can be over 12% [21].

III. Preliminaries

The focus of this work is around an efficient sparse regularized approach for weight reduction. This could be extended by exploiting the opportunity for weight quantization but this is not explored here. Before introducing our method, we introduce the background to SVRG optimization and $\ell_1$ regularization.

A. SVRG optimization

SVRG is a gradient-based optimization approach that updates the weights using a combination of the full batch gradient with a more frequent stochastic update using a subset of the training set as:

$$w_{i+1} = w_i - \eta_i \left( \frac{1}{n} \sum_{i=1}^{n} (\nabla f_i(w_{i+1}^s) - \nabla f_i(\bar{w})) + \bar{g}_{i+1}^s \right),$$

where $0 \leq t \leq m - 1$ and $0 \leq s \leq S - 1$ control the number of iterations and the frequency of the stochastic updates. $t_i$ is randomly selected from $\{1, ..., n\}$, $\bar{w}$ is the weight after every period iteration and $g = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\bar{w})$ is the average value of full gradients. Gradients in Eq 2 are generated by two terms, $\frac{1}{n} \sum_{i=1}^{n} (\nabla f_i(w_{i+1}^s) - \nabla f_i(\bar{w}))$ and $\bar{g}_{i+1}^s$. They ensure that a reduction of variance helps initial weights close to a global minima boost the convergence rate of SGD in strongly convex problems. However, for non-strongly convex problems, global minimization of a non-convex function is NP-hard [22]. It has been argued that SVRG can also be applied in non-convex neural network objectives to accelerate the local convergence rate of SGD [5]. An improved version of SVRG was proposed that was shown to have non-asymptotic rates of convergence for non-convex optimization and provably faster than SGD [22]. The details are described in Algorithm 1.

Algorithm 1: SVRG ($w^0$, $\eta$, $\{p_i\}_{i=0}^{m}$, $m$, $S$)

**Input**: Update epoch length $m$, learning rate $\eta$, number of epochs $S$, discrete probability distribution $\{p_i\}_{i=0}^{m}$ and initialize $\bar{w}^0 = w^0 = w_0^0$.

1. for $s=0$ to $S-1$ do
2.  $\bar{w}_{0}^{s+1} = w_{0}^{s} = \bar{w}^{s}$;
3.  $g^{s+1} = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\bar{w}^{s})$;
4. for $t = 0$ to $m - 1$ do
5.  randomly select $i$ from $\{1, ..., n\}$;
6.  $v_i^{s+1} = \nabla f_i(w_{i}^{s+1}) - \nabla f_i(\bar{w}) + \bar{g}^{s+1}$;
7.  $w_{i}^{s+1} = w_{i}^{s+1} - \eta_i v_i^{s+1}$;
8.  $w_{i}^{s+1} = \sum_{i=0}^{m} p_i w_{i}^{s+1}$;

**Output**: $w_0$ chosen uniformly randomly from $\{w_{i}^{s+1}\}_{i=0}^{m}$
B. Cumulative $\ell 1$ regularization

$\ell 1$ regularization induces sparsity in the resulting model parametrisation, which can be implemented efficiently to encourage many connections in the network to be removed. In practice the regularisation parameter $\gamma$ must be chosen carefully to control the generalisation ability of the model. Furthermore, care must be taken in the optimization approach to handle the non-differentiable element of the $\ell 1$ regularizer. Cumulative $\ell 1$ regularization has been proposed to address this by cumulating the $\ell 1$ penalties to resolve these problems [10]. The method clips regularization updates at zero and provides a more stable convergence for the weights. Moreover, the cumulative penalty can reduce the weight to zero more quickly. Cumulative $\ell 1$ regularization can be integrated with SVRG, by using the cumulative update to calculate the gradient associated with the regularization term,  

\[
\text{if } w^{s+1}_{t+\frac{1}{2}} > 0 \text{ then } \quad w^{s+1}_{t+\frac{1}{2}} = \max \left(0, w^{s+1}_{t+\frac{1}{2}} - (u + q^{s+1}_t)\right), \\
\quad \text{else if } w^{s+1}_{t+\frac{1}{2}} < 0 \text{ then } \quad w^{s+1}_{t+\frac{1}{2}} = \min \left(0, w^{s+1}_{t+\frac{1}{2}} + (u - q^{s+1}_t)\right),
\]

where $u$ is a constant value to control the compression rate of model given by the average value of the total $\ell 1$ penalty multiplied by $\gamma$, $w^{s+1}_{t+\frac{1}{2}}$ represents the weight after Eq. 2 update, and $q^{s+1}_t$ is the difference between weights over one epoch and is given by  

\[
q^{s+1}_t = \sum_{i=0}^{m-1} (w^{s+1}_{t+\frac{1}{2}} - w^{s+1}_{t-\frac{1}{2}}).
\]

IV. Non-convex SVRG for $\ell 1$ regularization

There are two problems when directly combining SVRG with cumulative $\ell 1$ regularization for non-convex problems: (1) adding the $\ell 1$ regularizer to SVRG limits the theoretical and practical performance of the optimization [23], (2) the trade-off in variance reduction versus the sparsity of the solution: when many weights become zero the robustness and convergence rate can be reduced. To address these two problems our compression method SSVRG introduces a hyper-parameter $\lambda$ in Eq. 2 alongside an adaptive learning rate $\eta$. The weight update then becomes  

\[
\text{if } w^{s+1}_{t+\frac{1}{2}} > 0 \text{ then } \quad w^{s+1}_{t+\frac{1}{2}} = w^{s+1}_{t+\frac{1}{2}} - \eta \left(\frac{1 - \lambda}{n} \sum_{i=1}^{n} (\nabla f_i(w^{s+1}_{t+\frac{1}{2}}) - \nabla f_i(\tilde{w})) + \lambda g^{s+1}\right) \\
\quad \text{else if } w^{s+1}_{t+\frac{1}{2}} < 0 \text{ then } \quad w^{s+1}_{t+\frac{1}{2}} = \max \left(0, w^{s+1}_{t+\frac{1}{2}} - (u + q^{s+1}_t + b)\right), \\
\quad \text{else if } w^{s+1}_{t+\frac{1}{2}} < 0 \text{ then } \quad w^{s+1}_{t+\frac{1}{2}} = \min \left(0, w^{s+1}_{t+\frac{1}{2}} + (u - q^{s+1}_t - b)\right),
\]

where $0 \leq \lambda \leq 1$ is a hyper-parameter that balances the gradients and $b$ is the smallest bias term amongst all layers. The full procedure is illustrated in Algorithm 2. The following sections describe the main contributions associated with the introduction of the adaptive learning rate and the hyper-parameter controlling the variance reduction, as well as supplementary details on the cumulative learning schedule and bias pruning.

Algorithm 2: SSVRG($w^0$, $\{p_i\}_{i=0}^m$, $m$, $S$)

Input: Update epoch length $m$, learning rate $\eta$, number of epochs $S$, discrete probability distribution $\{p_i\}_{i=0}^m$ and $u = 0;$

1. Initialize $\tilde{w}^0 = w^0 = \tilde{w}^0$, $q_i = 0$ for all $i$;
2. for $s=0$ to $S-1$ do
3. \hspace{1em} $w^0_{s+1} = w^0$;
4. \hspace{1em} $g^{s+1} = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\tilde{w})$;
5. \hspace{1em} for $t=0$ to $m-1$ do
6. \hspace{2em} randomly select $i$ from $\{1, ..., n\}$;
7. \hspace{2em} $v^{s+1}_{t+\frac{1}{2}} = (1 - \lambda) (\nabla f_i(w^{s+1}_{t+\frac{1}{2}}) - \nabla f_i(\tilde{w}) + \lambda g^{s+1})$
8. \hspace{2em} $w^{s+1}_{t+\frac{1}{2}} = w^{s+1}_{t+\frac{1}{2}} - \eta v^{s+1}_{t+\frac{1}{2}}$;
9. \hspace{2em} $z = w^{s+1}_{t+\frac{1}{2}}$;
10. \hspace{2em} $u \leftarrow u + \omega \lambda \gamma n$;
11. \hspace{2em} $\tilde{b}$ is minimal bias in all layers;
12. \hspace{2em} if $w^{s+1}_{t+\frac{1}{2}} > 0$ then
13. \hspace{3em} $w^{s+1}_{t+\frac{1}{2}} = \max \left(0, w^{s+1}_{t+\frac{1}{2}} - (u + q^{s+1}_t - b)\right)$;
14. \hspace{2em} else if $w^{s+1}_{t+\frac{1}{2}} < 0$ then
15. \hspace{3em} $w^{s+1}_{t+\frac{1}{2}} = \min \left(0, w^{s+1}_{t+\frac{1}{2}} + (u - q^{s+1}_t + b)\right)$;
16. \hspace{2em} $q^{s+1}_t \leftarrow q^{s+1}_t + (w^{s+1}_{t+\frac{1}{2}} - z)$;
17. \hspace{2em} $\tilde{u}^{s+1} = \sum_{i=0}^{m} p_i w^{s+1}_{t+\frac{1}{2}}$;

Output: $w_a$ chosen uniformly randomly from $\{[w^{s+1}_{t+\frac{1}{2}}]_{i=0}^{m}\}$.

A. Hybrid adaptive learning rate and variance control

Selecting the learning rate is important because it can dramatically affect the performance and convergence rate of optimization. In SVRG a single fixed learning rate is specified. If $\ell 1$ regularization is combined with non-convex SVRG the single fixed learning steps cannot control the reduced variance in SVRG to achieve fast convergence. Another criticism against variance reduced schemes is that the ability to escape local minimum and saddle points in non-convex optimization problems is reduced. Research has shown that adaptive learning rates can be applied with reduced variance to provide faster convergence rates on non-convex optimization [24], [25]. Furthermore, a single fixed learning rate may not be able to balance the gradient terms associated with the loss and regularizer.

In the standard SVRG algorithm (Algorithm 1) the balance of the gradient update between the full batch and
stochastic estimate is fixed. In our approach we choose to introduce a hyper-parameter to allow the algorithm to span a range of behaviours from stochastic, through reduced variance, to batch gradient descent. Consequently with careful adoption of this hyper-parameter the algorithm is able to gain benefits from the stochastic element to speed up computation and escape local minima/saddle points, and reduced variance to improve convergence rate.

Following the work from [25] which developed an approach to determine an adaptive learning rate η, we extend it to show how the hyper-parameter λ can be estimated and adapted to control the variance. We show that the hybrid adaptive learning rate in each epoch defines a range of λ values. The argument shows that an adaptive learning rate could be derived as a maximum of two terms. We adopt the same approach for Algorithm 2 and let i from 1 to T, p_m = 1 and p_i = 0 for 0 ≤ i < m, giving the learning rate as

$$\eta_i = \max\{\eta_0/\sqrt{t}, \mu_1/L\eta_0^2\}$$

where m = \lfloor n(1-\mu_1)^4/3\mu_1\rfloor, (0 < \mu_0, \mu_1 < 1) and T = mS is total number of iterations. The first term is dependent on t and hence will decrease with increasing iterations, whereas the second term is independent of t but dependent upon the data size. λ can be used to adjust the weighting of the stochastic element with the full batch gradient to control the variance of the optimization approach and this can be done at each iteration. The first term of the learning rate expression is larger than second term when t is small. In the first case, the variance will be reduced to accelerate the convergence rate in early iterations. On the other hand, when the second term becomes larger later in optimization, higher variance will help gradients escape from local minimum. Consequently, our hypothesis is that when the first term in Eq. 6 is maximum a reduced variance approach (0.5 < λ ≤ 1) will be better, and in the second case a more stochastic gradient approach (0 ≤ λ < 0.5) will provide better optimization.

B. Learning rate for cumulative ℓ1 regularization.

We set the learning rate ω_0 for cumulative ℓ1 regularization following [10] as,

$$\omega_t = \omega_0 \alpha^t$$

where the α is a decay constant. The scheduling is practical because exponential decay ensures that the learning rate does not drop too fast at the beginning and too slowly at the end.

C. Bias-based pruning

To further reduce the number of weights, we add a bias-based pruning b after the ℓ1 regularization in each iteration. The pruning rule is based on the following heuristic [26]: connections (weights) in each layer will be removed if their value is smaller than the network’s minimal bias. If the absolute value of weight connections are smaller than the absolute value of the smallest bias of the entire network in each batch, these connections have least contribution to the node, which can be removed. In practice, bias-based pruning has no effect on train and test loss.

V. Experiments

In order to estimate and compare the effect of our compression method on different topologies, we select deep neural networks (DNNs) and convolutional neural networks (CNNs). The DNN chosen is LeNet-300-100 which has two fully connected layers as hidden layers with 300 and 100 neurons respectively. The CNN chosen is LeNet-5 which has two convolutional layers and two fully connected layers. We evaluate the performance of our new compression method using MNIST, and CIFAR as benchmarks. Our compression method was implemented using Caffe^1.

A. Experimental settings

a) Learning rate η and ω: Our experiments determined the two learning rates η and ω, according to Eq. 6 and 7 respectively. The coefficients η_0, μ_0, μ_1 and L are empirically chosen so that SSVRG gives the best performance on the training loss. The value of η_0/√t is will be reduced by the increasing iteration number, t, but the value of μ_1/L\eta_0^2 is fixed for a given dataset. A value of μ_0 = 1/3 was determined by experiment and L = 4 was taken from [5], [27]. For the learning rate ω, α = 0.8 was chosen by experiment to give good performance.

b) Initialization: Weight initialization is important when training neural networks. Following the approach by [28], we use normalized initialization to scale the value of initial weights in each layer by uniform r = [-\sqrt{6/n_{in} + n_{out}}, \sqrt{6/n_{in} + n_{out}}], where n_in is the number of neurons in the input layer and n_out is the number of neurons in the output layer.

c) Loss function: In our experiments the quality is evaluated using cross entropy by the softmax log-loss function as the result of training/test error. We also use mean squared error (MSE) to estimate the effectiveness of the training/test loss.

B. Evaluating the range of λ

We experimentally estimate the appropriate range of the hyper-parameter λ via calculating test error and training loss of two LeNet models on MNIST, CIFAR-10 and CIFAR-100 datasets shown in Fig. 1 and Fig 2. Based on the two cases of the learning rate η, we found that when the value of learning rate η depends on number of iterations, it can achieve good performance and fast convergence when 0.5 < λ ≤ 1 as shown in Fig. 1. Particularly, the optimal value of λ in the first case is close to 0.8 because if λ is larger than 0.8 (e.g. λ = 0.95) the performance deteriorates. Otherwise, when the value of

^1Caffe is a deep learning framework. Source code can be download: http://caffe.berkeleyvision.org
Fig. 1. Effects of $\lambda$ on the performance and convergence rate. LeNet-5 is applied to three datasets using SVRG ($\lambda = 0.5$ and $\eta = \eta_0/\sqrt{t}$) as a baseline. When $t$ is small, $\lambda = 0.8$ is shown to achieve the best performance and fastest rates of convergence.
Fig. 2. Effects of $\lambda$ on the performance and convergence rate. LeNet-5 is applied to three datasets using SVRG ($\lambda = 0.5$ and $\eta = \mu_1/Ln^{2/3}(1-\mu_0)^2$) as a baseline. $\lambda = 0.1$ is shown to achieve the best performance and fastest rates of convergence.
Fig. 3. Results are shown for the error vs compression rates on three datasets. LeNet-300-100 is used for MNIST and LeNet-5 for the two CIFAR datasets. The comparison is shown for three compression methods, SSVRG(with/without bias pruning) and MSVRG-C-L1 that combines MSVRG with cumulative ℓ₁ regularization. SSVRG is shown to outperform MSVRG-C-L1 to provide superior models over the pareto front. Bias pruning is shown to offer some benefit but the benefit is limited and not universal.

Fig. 4. Comparison of the convergence rate of SSVRG and MSVRG by training/test loss and gradient magnitude. The significantly faster convergence of SSVRG over MSVRG can be seen in the lower training/test loss.
learning rate depends on the number of training samples, good performance and fast convergence can be obtained if $\lambda$ is close to zero as shown in Fig. 2.

C. Evaluation of the trade-off between memory requirements and performance

The compression rates of the SSVRG approach are controlled by $\gamma$, providing a range of performance-compression operating points. Fig. 3 shows how the test error and weight sparsity vary as the regularization parameter $\gamma$ is adjusted. In Fig. 3 the results are compared with MSVRG optimization [25] with cumulative $\ell_1$ regularization. On all three datasets SSVRG has the pareto optimal front on the error-compressions axes.

To highlight the results we show the weight statistics for each layer of a good operating point of the model for MNIST in Table I and II. The results show that SSVRG can significantly remove many weights in the fully connected layers. For LeNet-300-100 models the number of weights in the first fully connected layers (ip1) contains about 88% of the total number of weights and this can be compressed by 97%. Furthermore, SSVRG is effective to reduce the number of weights in both of the convolutional layers (conv1 and conv2) and the two fully connected layers (ip1 and ip2) in the LeNet-5 model. SSVRG reduces the number of weights to 4% with a 1.58% test error rate on the LeNet-300-100 model and 0.6% of the weights with a 0.72% test error rate on the LeNet-5 model.

A further experiment compares the convergence rate of SSVRG with MSVRG without regularization via estimating the training/test loss and the number of iterations when the value of gradients being converged to zero. In this experiment, we choose the optimal value of $\lambda$ for our method which has significantly faster convergence than MSVRG on all three different datasets shown in Fig. 4.

D. Comparison with leading results

In Table I, we choose three other competitive compression methods with leading results for comparison. Table I shows that the compression rate of our method on LeNet-5 can be up to $300\times$ with lower test error than the original model. Consequently, after convergence the network parameters of LeNet-300-100 and LeNet-5 are reduced by $103\times$ and $295\times$, respectively, which results in less than 3% of the network connections being maintained while the prediction accuracies are slightly better than the original models.

VI. DISCUSSION

In this paper, we proposed the method SSVRG that can efficiently reduce the number of weights in neural networks without sacrificing accuracy by employing a hybrid learning rate and hyper-parameter to control the variance reduction. The appropriate value of $\lambda$ can balance the different part of the gradients in SVRG and work with a flexible hybrid learning rate to accelerate the convergence and improve the performance of the model on sparse learning representations. In our experiments, our method can significantly reduce the number of weights by up to $300\times$ without accuracy loss. After compression by our method, a compact deep neural network can be efficiently deployed on an embedded device with performance of the original model.

REFERENCES


