End to end network compression using differentiable quantized weights

Santanu Rathod*
Department of Electrical Engineering,
IIT-Bombay.
CVL-ETH Zurich.
santanusrathod4415@gmail.com

July 1, 2019

Abstract

In the recent years neural networks have seen remarkable success on many of the computer vision and other tasks, but they usually require an army of GPUs in order to perform robustly. There have been attempts to have a comparable robust performance without using huge computational resources or simply put compress the information contained in the original neural network architectures. In this study we propose a novel end to end architecture with differentiable quantized weights which gives robust performance when compared with the original models learned weights for inference tasks. This in our knowledge is the first attempt towards having an end to end architecture with all the mechanisms occuring in a single pipeline without going through explicit pruning, sparsification or any parent teacher networks.

Network Compression Differentiable quantization

1 Introduction

In recent years, deep neural networks (DNNs) have led to many breakthrough results in machine learning and computer vision, and are now widely deployed in industry. Modern DNN models often have millions or tens of millions of parameters, leading to highly redundant structures, both in the intermediate feature representations they generate and in the model itself. Although over-parametrization of DNN models can have a favorable effect on training, in practice it is often desirable to compress DNN models for inference, e.g., when deploying them on mobile or embedded devices with limited memory. The ability to learn compressible feature representations, on the other hand, has a large

*Links for homepage of authors.

potential for the development of (data-adaptive) compression algorithms for various data types such as images, audio, video, and text, for all of which various DNN architectures are now available.

The work in the network compression community so far for can be roughly divided in a basic approaches 1. Using pruning-quantization-pruning[2], 2. Training a smaller student from a parent [4], 3. Learning sparse representations [12], 4. Using MDL(Minimum Description Principle) to learn compact bayesian representations [11]. Some others explore the possibility of finding the best compact neural architectures, using either reinforcement learning(RL) or NAS(neural architecture search).

We take a bit of a detour from the above mentioned techniques and look at the problem of network compression from a rate-distortion perspective. In order to compress a set of continuous model parameters or features, we need to approximate each parameter by one representative from a set of quantization levels (or vectors, in the multi-dimensional case), each associated with a symbol, and then store the assignments (symbols) of the parameters or features, as well as the quantization levels.

Representing each parameter of a DNN model will come at the cost of a distortion D, i.e., a loss in performance (e.g., in classification accuracy for a classification DNN with quantized model parameters. The rate R, i.e., the entropy of the symbol stream, determines the cost of encoding the model or features in a bitstream.To learn a compressible DNN model or feature representation we need to minimize D + βR, where $\beta > 0$ controls the rate-distortion trade-off. Including the entropy into the learning cost function can be seen as adding a regularizer that promotes a compressible representation of the network or feature representation. However, two major challenges arise when minimizing (D + β R) for DNNs:
i) Coping with the non-differentiability (due to quantization operations) of the cost function D+β R and ii) obtaining an accurate and differentiable estimate of the entropy (i.e., R).

In this study, we propose a unified end-to-end learning framework for learning , jointly optimizing the model parameters, the quantization levels, and the entropy of the resulting symbol stream to compress the model itself. We address both challenges i) and ii) above with methods that are novel in the context DNN model and feature compression.

2 Related work

2.1 Non-bayesian based approaches:

There is an ample amount of research on compressing neural networks, so that we will only discuss the most prominent ones, and those which are related to our work. An early approach is *Optimal Brain Damage* [1] which employs the Hessian of the network weights in order to determine whether weights can be pruned without significantly impacting training performance. A related but simpler approach was proposed in [2], where small weights are truncated to zero, alternated with re-training. This simple approach yielded – somewhat surprisingly – networks which are one order of magnitude smaller, without impairing performance. The approach was re- fined into a systematic pipeline called Deep Compression, where magnitude-based weight pruning is followed by weight quantization (clustering weights) and Huffman coding [3].

HashNet proposed by [6] also follows a simple and surprisingly effective approach: They exploit the fact that training of neural networks is resistant to imposing random constraints on the weights. In particular, they use hashing to enforce groups of weights to share the same value, yielding memory reductions of up to 64x with gracefully degrading performance.

Several attempts at KD (knowledge distillation) have also been made, most notable of them include, by [4], [5]. In these techniques they essentially try to distill the learning into a smaller student network, jointly with the teacher network. Extensions to their work tries to implement them quantized manner.

2.2 Bayesian based approaches:

These methods are primarily motivated by using a bayesian variational framework and somehow attempting to use the *bits back argument* [7] into the framework. Works like *Bayesian Compression* [8] use a Bayesian variational framework and are motivated by the bits-back argument. They introduce using hierarchical priors to prune nodes instead of individual weights and use the posterior uncertainties to determine the optimal fixed point precision to encode the weights.

Ullrich et al. [9] they show that competitive compression rates can be achieved by using a version of 'soft weight-sharing' [10], they achieve both quantization and pruning in one simple (re-)training procedure, exposing the relation between compression and the minimum description length (MDL) principle.

Previous works however still essentially sample the weights from a learned distribution and then use it, to which *Minimal Random Code Learning(MIRACLE)* [11] study argues can be still improved by relaxing weight determinism and using a full variational distribution over weights allows for more efficient coding schemes and consequently higher compression rates.

2.3 Other:

Other techniques include using sparsification, like in *Variational Dropout Sparsifies Deep Neural Networks* by [12], their work essentially uses a trick which reduces the huge gradient variation which occurs during the training of Variational dropout for dropout rate $\alpha -> \infty$, and thereby providing an elegant technique for sparsification. Techniques as suggested by [13] use novel methods of NAS(Neural Archihtecture Search) called Differentiable Architecture Search(DARC).
[14] use reinforcement learning techniques to learn a sparse representation of a DNN architecture wherein the agent learns as to which layers it can afford to drop without significant drop in accuracy.

3 Proposed Method

In this section we include the problem formulation, challenges, and our implementation techniques. What we in essence do is, 1. replace the basic layers' weights by the differentiable quantized weights, with the resulting cross entropy loss, $\mathbf{L}(\mathbf{y}, \hat{\mathbf{y}})$ between labels and our final feature map being interpreted as our distortion, 2. assume that each of our weight comes from a $N(\mu, \sigma^2)$ and thus get a rate(R) term, H($\mathbf{w}, \hat{\mathbf{w}}$). Where $\mathbf{w}$: original weights, $\hat{\mathbf{w}}$: quantized weights. Thus we intend to minimize:
L_{net}= $\mathbf{L}(\mathbf{y}, \hat{\mathbf{y}})$ + βH($\mathbf{w}, \hat{\mathbf{w}}$).
Where $\mathbf{w}$: original weights, $\hat{\mathbf{w}}$: quantized weights, β>0:controls the rate-distortion trade-off.

The reader should recall that higher the rate(R) lesser the distortion[15] and vice versa. So we intend to find a middle ground between the rate-distortion trade-off.

3.1 Problem Formulation

Preliminaries and Notations. We consider the standard model for Deep neural networks, where we have an architecture F : $R^{d_1} \rightarrow R^{d_{K+1}}$ composed of K layers F = F_K.....F_1 , where layer F_i maps $R^{d_i} \rightarrow R^{d_{i+1}}$, and has parameters $w_i \in R^{m_i}$. We refer to $W = [w_1, ..., w_K]$ as the parameters of the network and we denote the intermediate layer outputs of the network as $x(0) := x$ and $x(i)$:= $F_i(x^{(i-1)})$, such that $F(x) = x^{(K)}$ and x(i) is the feature vector produced by layer F_i. The parameters of the network are learned w.r.t. training data $X = x_1, ..., x_N \subset R^{d_1}$ and labels Y = $y_1, ..., y_N \subset R^{d_{K+1}}$, by minimizing a real-valued loss L(X,Y;F). Typically, the loss can be decomposed as a sum over the training data plus a regularization term,

$$\text{L(X,Y;F)}= \frac{1}{N}\sum_{i=1}^{N} L(F(x_i), y_i) + \lambda R(W) \quad (1)$$

where $\mathbf{L}(\mathbf{F}(\mathbf{x}), \mathbf{y})$ is the sample loss, $\lambda > 0$ sets the regularization strength, and R(W) is a regularizer (e.g., $\text{R(W)} = \sum_i {w_i}^2$ for l_2 regularization). In this case, the parameters of the network can be learned using stochastic gradient descent over mini-batches. Assuming that the data X , Y on which the network is trained is drawn from some distribution $P_{X,Y}$, the loss L(X,Y;F) can be thought of as an estimator of the expected loss $E[L(F(X), Y) + \lambda R(W)]$.

Compressible Representations We say that a weight parameter w_i if it can be serialized to a binary stream using few bits. For DNN compression, we want the entire network parameters W to be compressible.
To store w_i with a finite number of bits, we need to map it to a discrete space. Specifically, we map w_i to a sequence of m symbols using a (symbol) encoder E: $R^d \rightarrow [L]^m$, where each symbol is an index ranging from 1 to L, i.e., [L] := 1,...,L. We assume that w_i is a sample from $N(\mu, \sigma^2)$.
So now our rate-distortion loss function becomes:

$$\min_{W,D,E} E_{X,Y}(L(\hat{\mathbf{F}}(x)), y + \lambda R(W)) + \beta H(\mathbf{w}, \hat{\mathbf{w}})$$
$$= \min_{W,D,E} E_{X,Y}(L(\hat{\mathbf{F}}(x)), y + \lambda R(W)) + \beta \sum_{i=1}^{N} (\int_{-\infty}^{\infty} -p(w_i) d(w_i) * log(p(\hat{\mathbf{w}}_\mathbf{i})))$$
$$= \min_{W,D,E} E_{X,Y}(L(\hat{\mathbf{F}}(x)), y + \lambda R(W)) + \beta \sum_{i=1}^{N} (-1 * log(p(\hat{\mathbf{w}}_\mathbf{i}))) \text{.....(since we have } \hat{\mathbf{w}}_\mathbf{i} \in [L], \text{ so } p(w_i) \perp p(\hat{\mathbf{w}}_\mathbf{i}))$$

(2)

Challenges The problem of finding a good encoder E, and parameters W that minimize the objective still remains. First, we need to impose a form for the encoder and second we need an approach that can optimize (2) w.r.t. the parameters W. Independently of the choice of E, (2) is challenging since E is a mapping to a finite set and, therefore, not differentiable. This implies that neither H(p) is differentiable no is differentiable w.r.t. the parameters w. , These challenges motivate the design decisions of our soft-to-hard annealing approach, described in the next section.

3.2 Method for quantized differentiable weights

Our model involves three kinds of weights, 1. original learned weights, 2. soft weights, which are differentiable wrt to original weights and should ideally be

very close to any of the value from the quantized set C= $(\hat{\mathbf{w_1}}, \hat{\mathbf{w_2}}, ...\hat{\mathbf{w_L}})$, where L= number of quantized levels, 3. hard weights, which actually belong to the quantized set C= $(\hat{\mathbf{w_1}}, \hat{\mathbf{w_2}}, ...\hat{\mathbf{w_L}})$.

Hard assignments For hard assignments we simply use nearest neighbour criteria. For a weight w, $w_{hard} = c_i$, where $c_i = \min_i[|w - c_1|^2, |w - c_2|^2, ...|w - c_L|^2]$

Soft assignments For soft assignments we have,
$\phi(\hat{\mathbf{w}}) = softmax(-\sigma[|w - c_1|^2, |w - c_2|^2, ...|w - c_L|^2])$,
where $softmax(y_1, y_2, y_3...y_L)_j$:= $\frac{e^{y_j}}{e^{y_1} + e^{y_2} + ...e^{y_L}}$ is the standard softmax operator, such that $\phi(\hat{\mathbf{w}})$ has positive entries and $|\phi(\hat{\mathbf{w}})|_1$=1. We denote that the jth entry of the $\phi(\hat{\mathbf{w}})$ with $\phi_j(\hat{\mathbf{w}})$ and note that,

$lim_{\sigma\to\infty}\ \phi_j(\hat{\mathbf{w}})$ {= 1, if ... $j = argmin_{j'\in[L]}\ |\hat{\mathbf{w}} - c_j^{'}|$
. {= 0 ... otherwise.
such that w_{soft} in $\lim_{\sigma\to\infty}$ converges to one-hot encoding of the nearest quantized unit in C.

We finally have, w_{soft}:=$\sum_{j=1}^{L} c_j\phi_i(\hat{\mathbf{w}}) = C\phi(\hat{\mathbf{w}})$, which in ideal case would be very close to w_{hard} and its differentiable unlike the hard counterpart.

Entropy H(w,$\hat{\mathbf{w}}$)= $\sum_{i=1}^{N}(\int_{-\infty}^{\infty} -p(w_i)d(w_i)*log(p(\hat{\mathbf{w}_i}))) = \sum_{i=1}^{N}(-1*log(p(\hat{\mathbf{w}_i})))$

$here, p(x) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{\frac{-(x-\mu)^2}{2\sigma^2}}$ and $\hat{\mathbf{w}_i}$ is the hard quantized weight.

Soft to hard annealing Our soft assignment scheme gives us differentiable approximations $\hat{\mathbf{F}}$ and $\hat{\mathbf{H}}(\mathbf{w})$ of the discretized network $\mathbf{F}$ and the sample entropy H(p), respectively. However, our objective is to learn network parameters W that minimize (2) when using the encoder with hard assignments, such that we obtain a compressible symbol stream E(z) which we can compress using, e.g., arithmetic coding [**?**]. To this end, we anneal σ from some initial value σ goes from 0 to ∞ during training, such that the soft approximation gradually becomes a better approximation of the final hard quantization we will use. Choosing the annealing schedule is crucial as annealing too slowly may allow the network to invert the soft assignments (resulting in large weights), and annealing too fast leads to vanishing gradients too early, thereby preventing learning. In practice, one can either parametrize σ as a function of the iteration, or tie it to an auxiliary target such as the difference between the network losses incurred by soft quantization and hard quantization..

4 Discussion and further scope

In our study while calculating the entropy loss($H(w, \hat{\mathbf{w}})$) we have assumed that the weights come from a $\mathbf{N}(\mu, \sigma^{\mathbf{2}})$, where μ= mean of pre-trained weights, σ^2= variance of pre-trained weights.
This I believe is a quite a big assumption and that our performance can be further robust by trying out the following two ideas:
1. To plot the means and variances of each layer and then handpick n mean(μ_i) and variance(σ_i^2) (n is experimental). After this consider that our weights come from $\pi_{\mathbf{1}}\mathbf{N}(\mu_{\mathbf{1}}, \sigma_{\mathbf{1}}^{\mathbf{2}}) + \pi_{\mathbf{2}}\mathbf{N}(\mu_{\mathbf{2}}, \sigma_{\mathbf{2}}^{\mathbf{2}}) + \pi_{\mathbf{3}}\mathbf{N}(\mu_{\mathbf{3}}, \sigma_{\mathbf{3}}^{\mathbf{2}}) + ...\pi_{\mathbf{n}}\mathbf{N}(\mu_{\mathbf{n}}, \sigma_{\mathbf{n}}^{\mathbf{2}})$, where $\pi_{\mathbf{1}}, \pi_{\mathbf{2}}...\pi_{\mathbf{n}}$ are learnable.

2. Instead of handpicking mean(μ_i) and variance(σ_i^2) like in (1) above we learn them along with the weights, so that we get μ_i and σ_i^2 suitable according to our model. However even in this case, n: number of gaussian distribution considered, is experimental and $\pi_{\mathbf{1}}, \pi_{\mathbf{2}}...\pi_{\mathbf{n}}$ in $\pi_{\mathbf{1}}\mathbf{N}(\mu_{\mathbf{1}}, \sigma_{\mathbf{1}}^{\mathbf{2}}) + \pi_{\mathbf{2}}\mathbf{N}(\mu_{\mathbf{2}}, \sigma_{\mathbf{2}}^{\mathbf{2}}) + \pi_{\mathbf{3}}\mathbf{N}(\mu_{\mathbf{3}}, \sigma_{\mathbf{3}}^{\mathbf{2}}) + ...\pi_{\mathbf{n}}\mathbf{N}(\mu_{\mathbf{n}}, \sigma_{\mathbf{n}}^{\mathbf{2}})$ are learnable.

I believe that going along these directions would give us better performance in terms of accuracy because they surely capture a more closer nature of our weights than a single $N(\mu, \sigma^2)$.

5 Conclusion

During our experiments we observe that back-propagating on the distortion loss itself gives quite a good accuracy inferring that our models do have some redundant information stored in them. It is this redundancy that we intended to remove by having representing our models as compactly as possible without significant decrease in accuracy. From the coding side of things, a lot of things are involved into the quantizing weights and main care should be taken wrt to the fact that gradients are actually being calculated wrt the origial weights and not the quantized ones which we've replaced in our model.

References

[1] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In Proceedings of NIPS, pp.598–605, 1990.

[2] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1135–1143, 2015

[3] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[4] Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015.

[5] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 535–541, New York, NY, USA, 2006. ACM.

[6] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *Proceedings of ICML*, pp. 2285–2294, 2015.

[7] G. E. Hinton and D. Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pp. 5–13. ACM, 1993.

[8] C. Louizos, K. Ullrich, and M. Welling. Bayesian compression for deep learning. In *Proceedings of NIPS*, pp. 3288–3298, 2017.

[9] Karen Ullrich, Edward Meeds, Max Welling. SOFT WEIGHT-SHARING FOR NEURAL NETWORK COMPRESSION ICLR 2017.

[10] Steven J Nowlan and Geoffrey E Hinton. Simplifying neural networks by soft weight-sharing. *Neural computation*, 4(4):473–493, 1992

[11] Marton Havasi, Robert Peharz, José Miguel Hernández-Lobato MINIMAL RANDOM CODE LEARNING : GETTING BITS BACK FROM COMPRESSED MODEL PARAMETERS *ICLR 2019*

[12] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. ICML 2017

[13] Shashank Singh, Ashish Khetan, Zohar Karnin DARC: Differentiable ARchitecture Compression

[14] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han AMC: AutoML for Model Compression and Acceleration on Mobile Devices

[15] Shannon, C. E. Coding theorems for a discrete source witha fidelity criterion. IRE Nat. Conv. Rec, 4(142-163):1,1959.