FAST MODEL COMPRESSION

Miguel Á. Carreira-Perpiñán and **Arman Zharmagambetov**,
EECS, UC Merced

1 Motivation and summary

Compressing neural nets is an active research problem and many algorithms have been proposed that achieve significant compression based on pruning, quantization, low-rank decomposition, etc. However, most of these algorithms require access to the original training set. This imposes considerable resources in runtime and storage—for instance, modern image classification datasets such as ImageNet contain millions of high-resolution images. In some applications, for example in-device compression, it is desirable to do much faster (almost instant) compression.

In this work we focus on the framework of the “Learning-Compression” (LC) algorithm [1–3] because it can be applied to potentially any kind of compression type and combinations thereof. The basic idea is to replace the original loss function with an approximate, simpler loss that does not require access to the training set. The resulting optimization problem can be solved analytically or using the LC algorithm. We show that we can still achieve significant compression but much faster.

Work supported by NSF award IIS–1423515, by a UC Merced Faculty Research Grant, by computing in the MERCED cluster (NSF grant ACI–1429783), and by NVIDIA GPU donation.

2 Fast model compression

Assume we have a large, *reference* model with P parameters that has been trained on a *loss* L (e.g. cross-entropy on a given training set) to solve a task (e.g. classification). That is, $\overline{\mathbf{w}} = \arg\min_{\mathbf{w}} L(\mathbf{w})$. We define *compression* as finding a low-dimensional parameterization $\mathbf{\Delta}(\mathbf{\Theta})$ of $\mathbf{w}$ in terms of $Q < P$ parameters $\mathbf{\Theta}$. *We seek a* $\mathbf{\Theta}$ *such that its corresponding model has (locally) optimal loss.* We define *model compression as a constrained optimization* problem:

$$\min_{\mathbf{w},\mathbf{\Theta}} \tilde{L}(\mathbf{w}) \text{ s.t. } \mathbf{w} = \mathbf{\Delta}(\mathbf{\Theta}),$$

where $\tilde{L}$ is approximation of the original loss L which we define later. Note that the original LC algorithm uses the true loss L, therefore it requires access to dataset.

The *decomposition mapping* $\mathbf{\Delta}$: $\mathbf{\Theta} \in \mathbb{R}^Q \rightarrow \mathbf{w} \in \mathbb{R}^P$ maps a low-dimensional parameterization to uncompressed model weights. The *compression mapping*:

$$\mathbf{\Pi}(\mathbf{w}) = \arg\min_{\mathbf{\Theta}} \|\mathbf{w} - \mathbf{\Delta}(\mathbf{\Theta})\|^2,$$

behaves as its “inverse” and appears in the C step of the LC algorithm. Our framework includes well-known types of compression (and combinations thereof), such as: pruning, quantization, low-rank compression, etc.

3 Approximation to the loss

We approximate the original loss L using Taylor’s theorem.Assume a given loss on a training set, e.g. the cross-entropy loss $L(\mathbf{w}) = -\sum_n y_n \log f(\mathbf{x}_n; \mathbf{w})$. Then the loss function can be approximated as:

$$\tilde{L}(\mathbf{w}) = L_0 + \mathbf{g}^T(\mathbf{w} - \overline{\mathbf{w}}) + \frac{1}{2}(\mathbf{w} - \overline{\mathbf{w}})^T\mathbf{H}(\mathbf{w} - \overline{\mathbf{w}}),$$

where $L_0 = L(\overline{\mathbf{w}})$ is the loss value of the reference model, $\mathbf{g}$ its gradient, $\mathbf{H}$ its Hessian and $\overline{\mathbf{w}}$ are the weights of the reference model ($\overline{\mathbf{w}}$ need not be an exact minimizer, so its gradient need not be zero). This approximation is very good near $\overline{\mathbf{w}}$ but degrades progressively as we go away from it. Therefore we have to expect that the resulting solution will not be as accurate as the original LC algorithm. But this is the price we need to pay in order to get a fast compression.

Here, $\mathbf{H}$ can be the full Hessian, diagonal, block diagonal, sparse or even zero if we use the first order approximation; here we focus on the diagonal approximation. Then the above loss approximation takes the following form (by neglecting constant term L_0):

$$\tilde{L}(\mathbf{w}) = \sum_{i=1}^{P} \left[g_i(w_i - \overline{w}_i) + \frac{1}{2}h_i(w_i - \overline{w}_i)^2 \right],$$

where g_i and h_i are the elements of the gradient vector and diagonal elements of the Hessian, respectively.

4 Fast “Learning-Compression” (LC) algorithm

This follows from using a penalty method (for simplicity, we describe the quadratic penalty) and alternating optimization, with the goal of separating the machine learning part (loss L) from the compression part (Δ). This results in an algorithm that alternates two generic steps while slowly driving the penalty parameter $\mu \rightarrow \infty$:

- **L (learning) step:**

  $$\min_{\mathbf{w}} \sum_{i=1}^{P} \left[g_i(w_i - \overline{w}_i) + \frac{1}{2}h_i(w_i - \overline{w}_i)^2 \right] + \frac{\mu}{2}\|\mathbf{w} - \mathbf{\Delta}(\mathbf{\Theta})\|^2$$

  It is a separable quadratic optimization whose solution is: $w_i = (h_i\overline{w}_i + \mu\Delta_i(\theta) - g_i)/(h_i + \mu)$. Note that in the original LC algorithm this step requires access to the training set and the neural net, which is computationally very costly (and requires stochastic gradient descent optimization in a GPU). Now the L step is data-independent and vastly faster.
- **C (compression) step:** $\min_{\mathbf{\Theta}} \|\mathbf{w} - \mathbf{\Delta}(\mathbf{\Theta})\|^2 \Leftrightarrow \mathbf{\Theta} = \mathbf{\Pi}(\mathbf{w})$. This means finding the best (lossy) compression of $\mathbf{w}$ in the ℓ_2 sense. This step is identical to the original LC algorithm. It is independent of the loss, training set and task. It can be solved by calling a compression mapping (e.g. thresholding, SVD, k-means, etc.) corresponding to the desired compression type.

5 Analytical solution of the optimization problem

In some particular cases the exact solution of the constrained optimization problem can be obtained analytically (without using iterative algorithms). For example, in the case of pruning [3], the optimization problem takes the form: $\min_{\mathbf{w}} \sum_{i=1}^{P} \left[g_i(w_i - \overline{w}_i) + \frac{1}{2}h_i(w_i - \overline{w}_i)^2 \right]$ s.t. $\|\mathbf{w}\|_0 \leq \kappa$. Its exact solution is given by picking the weights having the largest values of $\alpha_i = g_i\overline{w}_i - \frac{1}{2}h_i\overline{w}_i^2 - g_i^2/(2h_i)$ and setting them to $w_i = \overline{w}_i - g_i/h_i$. If $\mathbf{g} = \mathbf{0}$ this solution corresponds to the Optimal Brain Damage algorithm of [4].

Another example is a problem of weight binarization: $\min_{\mathbf{w}} \sum_{i=1}^{P} \left[g_i(w_i - \overline{w}_i) + \frac{1}{2}h_i(w_i - \overline{w}_i)^2 \right]$ s.t. $w_1, \dots, w_P \in \{-1, +1\}$. The problem separates over the weights and can be solved for each w_i by enumeration (try -1 and $+1$ and pick the one which gives the lowest value of the loss).

6 Experiments

The figure shows compression results (as a tradeoff curve of error vs compression level) for the VGG-13 neural nets on CIFAR-10. Currently, Tensorflow (and some other deep learning frameworks) are not able to provide just the diagonal of the Hessian. Therefore, we estimate it using the Gauss-Newton approximation. As we can see the fast LC algorithm is able to achieve low test error as long as we don’t compress much. The original LC shows better results on all experiments but its runtime is about 6 hours, whereas the fast compression runs only about 2 minutes for quantization, 1 second for pruning.

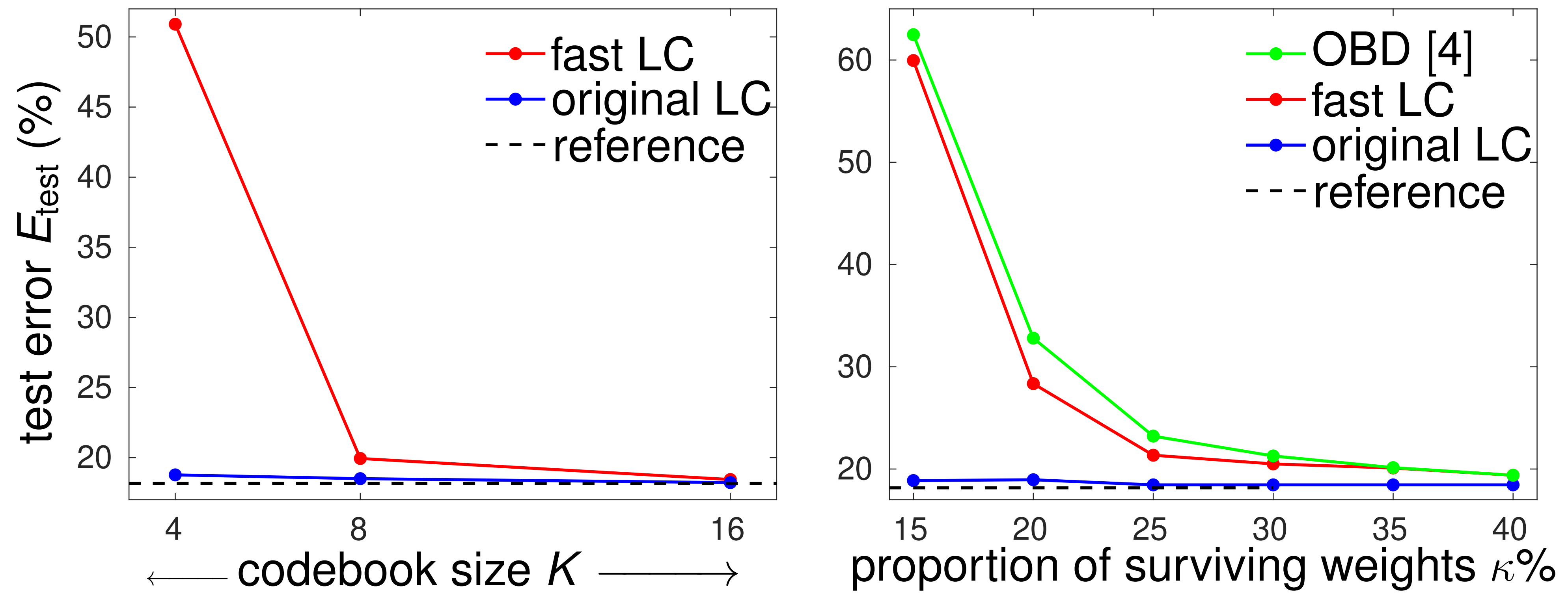

Figure 1: Error-compression curves for VGG-13 on CIFAR-10: quantization (left), pruning (right).

7 References

[1] M. Á. Carreira-Perpiñán: “Model compression as constrained optimization, with application to neural nets. Part I: General framework”. *arXiv:1707.01209 [cs.LG]*, July 5 2017.

[2] M. Á. Carreira-Perpiñán and Y. Idelbayev: “Model compression as constrained optimization, with application to neural nets. Part II: Quantization”. *arXiv:1707.04319 [cs.LG]*, July 13 2017.

[3] M. Á. Carreira-Perpiñán and Y. Idelbayev: ““Learning-Compression” algorithms for neural net pruning”. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[4] Y. LeCun, J. S. Denker, and S. A. Solla: “Optimal brain damage”. In *Advances in Neural Information Processing Systems 2*, p. 598–605, 1990.