Ternary Residual Networks
Extended Abstract

Abhisek Kundu, Kunal Banerjee, Naveen Mellempu, Dheevatsa Mudigere, Dipankar Das,
Bharat Kaul, Pradeep Dubey*
{abhisek.kundu,kunal.banerjee,naveen.k.mellempu,dheevatsa.mudigere,dipankar.das,bharat.kaul,pradeep.dubey}@intel.com

ABSTRACT
Sub-8-bit representation of DNNs incur some discernible loss of accuracy despite rigorous (re)training at low-precision. Such loss of accuracy essentially makes them equivalent to much shallower counterparts, diminishing the power of being deep networks. To address this problem of accuracy drop, we introduce the notion of residual edges, where we add more low-precision edges to sensitive branches of the sub-8-bit network to compensate for the lost accuracy (loss ~ 1% of FP32 baseline). We present a perturbation theory to identify such sensitive edges. Moreover, for varying accuracy requirements in a dynamic environment, the deployed model can be upgraded/downgraded on-the-fly by partially enabling/disabling residual edges. Finally, all the ternary edges are sparse in nature, and the ternary residual conversion can be done in a resource-constraint setting with no low-precision (re)training. Experiments are shown on ResNet-101, GoogLeNet v3, and AlexNet pre-trained on ImageNet.

1 INTRODUCTION
There is a growing need to make deep neural networks (DNNs) resource-friendly via a compact and/or reduced-precision representation for both mobile devices and data servers. One approach is to reduce the number of parameters in the network (e.g., SqueezeNet [12], MobileNet [10], SEP-Nets [16]) to have a very small size (~5 MB) typically targeting mobile devices. However, it is not surprising that their overall accuracy is limited on complex dataset ImageNet [4], e.g., SEP-Net Top-1 accuracy is ~10% off from that of ResNet-50. Deploying them on sensitive applications, e.g., autonomous cars and robots, might be impractical. The other approach is concerned with the reduction in size of parameter representation. Well-known methods of this kind are pruning [5, 23, 26], quantization [6, 11, 17, 19, 22, 27], binarization [3], ternarization [1, 8, 15, 18, 29], hashing [2], Huffman coding [7] and others [20, 28]. Such low-precision representation demands for specialized low-precision arithmetic [18, 24, 25] and hardware design. Google’s TPU [13] sets a trend for a low-precision inference pipeline (8-bit activations, 8-bit weights). Here we are mainly focused on the trade-off between sub-8-bit representation ([11, 18, 20, 28]) and accuracy of deeper networks, keeping an eye on the power-performance factors. In reality, sub-8-bit models for deep networks incur some noticeable drop in accuracy, despite rigorous low-precision (re)-training. This loss severely undermines the purpose of deploying a deep (sub-8-bit) network, as we may find an equivalent shallower 8-bit network. We seek to answer: Can a sub-8-bit model achieve similar accuracy as FP-32 model with better model size and power-performance numbers comparing to 8-8?

*Pradeep Dubey is at Intel Parallel Computing Labs, Santa Clara, USA. Other authors are at Intel Parallel Computing Labs, Bangalore, India.

• Contributions: (1) We introduce the notion of residual edges, where we add more low-precision edges to sensitive branches of a DNN, and this is guided by a perturbation theory, (2) Unlike existing sub-8-bit models, we show how to achieve extremely high accuracy (~1% off from FP32), (3) Existing models cannot be altered once they are deployed (‘fixed-accuracy-fixed-power’ mode). However, our model can be upgraded/downgraded on-the-fly by partially enabling/disabling some of the residual edges in a dynamic environment, depending on the varying accuracy/power requirements. For example, when autonomous cars or robots are in a less eventful environment where less number of objects are involved, the classification problem becomes considerably simpler (sufficient to distinguish among distinct objects, such as humans, dogs, vehicles, trees, etc. rather than discriminating among multiple breeds of dogs), and by disabling many edges we can downgrade the model in terms of compute, power, etc., yet maintain very high accuracy for those (less number of) classes. Drawing an analogy between human attention and precision, and also an analogy between stress due to attention and power consumption, it is natural for us to be selectively attentive to certain tasks that requires processing more information. Such upgrade/downgrade of low-precision DNNs mimics a more real-world scenario that other existing models are unable to imitate.

2 PERTURBATION IN PRE-TRAINED DNN
A DNN is a composition of parametric functions $f_i$ where (locally) optimal values of parameters $W^{(i)}$ are achieved via network training. We interpret quantization and/or sparsification as adding a noise to $W^{(i)}$ to produce sub-optimal $\hat{W}^{(i)}$. We want to quantify the effect of $\hat{W}^{(i)}$ on the final outcome, e.g., classification scores (Top-1 accuracy). For this, let us assume that our DNN has $\ell$ layers and let $y \in \mathbb{R}^d$ ($d$ is the number of classes) be the output vector of layer $\ell$ such that its $i$-th component $y_i$ contains the classification score for $i$-th class, for a given input $x$. Let $\hat{y} \in \mathbb{R}^d$ denote the perturbed vector $y$ due to added noise. Then, the largest component of $y$ should remain the largest in $\hat{y}$ for no loss of accuracy. Practically, we want to first derive an upper bound on $\|y - \hat{y}\|_F / \|y\|_F$ in terms of layer-wise perturbation of the pre-trained network, and then we want to control such perturbations to keep the final accumulated noise small. Let the set of functions be $f_{dat} = \{ \text{Convolution with Batch Normalization, Matrix Multiplication, ReLU, Pooling } \}$. Functions in $f_{dat}$ can be linear or non-linear, parametric or non-parametric, convex or non-convex, smooth or non-smooth. DNN: $f = f_\ell f_{\ell-1} \ldots f_0 : D_0 \rightarrow D_{\ell}$, where each $f_i \in f_{dat}$, $f_i : D_{i-1} \rightarrow D_i$ with parameters $W^{(i)}$, and
each \( \mathcal{D}_i \) is an arbitrary metric space where \( \| \cdot \| \) denotes a distance metric on set \( \mathcal{D}_i \) (for simplicity, we focus on normed space only). For all \( X^{(i)} \in \mathcal{D}_i \), we define \( X^{(i)} \in \mathcal{D}_i \) and \( \bar{X}^{(i)} \in \mathcal{D}_i \) as follows. For \( i = 1, \ldots, t \), \( X^{(i)} = f_i(X^{(i-1)}; \bar{W}^{(i)}), \|X^{(i)}\| = f_i(X^{(i-1)}; \bar{W}^{(i)}) \), where \( \bar{X}^{(i)} \) and \( \bar{W}^{(i)} \) are perturbed versions of \( X^{(i)} \) and \( W^{(i)} \), respectively.

We want to measure how the outcome of \( f \) gets perturbed in the neighborhood of the local optima. Simplifying for small noise:

\[
\delta_i = \|X^{(i)} - X^{(i)}\| \leq \epsilon_i \qquad \text{and} \qquad \epsilon_i = \|\bar{W}^{(i)} - W^{(i)}\| / \|W^{(i)}\|.
\]

We derive the following for constants \( \gamma_i \).

\[
\|f(X^{(i-1)}; \bar{W}^{(i)}) - f(X^{(i-1)}; W^{(i)})\| = \sum_{i=1}^{t} \|f_i(X^{(i-1)}; \bar{W}^{(i)}) - f_i(X^{(i-1)}; W^{(i)})\| = \sum_{i=1}^{t} \|\bar{W}^{(i)} - W^{(i)}\| / \|W^{(i)}\|.\]

Keeping both \( \gamma_i \) and \( \epsilon_i \) small implies overall small perturbation. Also, for earlier layers \( \epsilon_i \) should be kept much smaller than those in later layers to have an overall small perturbation. Our empirical evaluation on quantized DNNs corroborates this theory.

### 2.1 Low-Precision DNN

Constraining activations and weights to 8 bits appears to induce only small perturbation, resulting in typically < 1% loss in accuracy. This can be explained by the theoretical bounds presented above, where \( \gamma_i \) and \( \epsilon_i \) are small. More challenging cases are sub-8-bit DNNs, e.g., 8-bit activations and ternary weights.

* **Ternary Conversion of Pre-trained DNN** : We convert FP32 weights \( W \) to ternary values \( \{-\alpha, 0, +\alpha\}, \alpha \geq 0 \), without re-training, via a simple threshold (\( T > 0 \)) based approach similar to [15, 18]. Let \( W \) denote a ternary weight, such that, \( i \)-th element \( W_i = \text{sign}(W_i) \), if \( |W_i| > T \), and 0 otherwise. The goal is to minimize element-wise error \( E(\alpha, T) = |W - \alpha W|_F^2 \). For better accuracy, [18] introduced FGQ by dividing the weight tensors into disjoint sub-tensors of size \( N \), and then ternarize such sub-tensors independently. Each sub-tensor requires only one multiplication during inference. Smaller \( N \) leads to better accuracy but with more multiplications. \( k \) disjoint ternary sub-tensors can represent \( 2k + 1 \) distinct values, i.e., model capacity increases linearly in number of such sub-tensors. Sub-8-bit representation of sensitive parameters may have a ‘blurring’ effect on later activations for very deep networks; consequently, extremely high accuracy results might be elusive in 8-2 model.

* **Ternary Residual Edges** : Neither all the sub-tensors are approximated well in ternary, nor are they equally sensitive. For a poorly approximated sub-tensor, we ternarize the residual sub-tensor (difference between the FP32 one and the ternary one) and accumulate the output this ternary residual sub-tensor during inference. We might need multiple such residuals for a sub-tensor if all the earlier residuals are, as a whole, not good enough. Let the \( j \)-th layer weight tensor \( W^{(j)} \) be partitioned into \( k \) disjoint sub-tensors \( W^{(j)}_k \) (being the perturbed version), \( i = 1, \ldots, k \). Then, sensitivity \( \epsilon_i \) of \( i \)-th sub-tensor in \( j \)-th layer is defined as follows. \( \epsilon_i = \|W^{(j)}_i - W^{(j)}_i\| / \|W^{(j)}_i\| \). Note that, \( \sum_{i=1}^{k} \epsilon_i^2 \leq \epsilon_j^2 \) (defined earlier in Section 2). This suggests, for a given perturbation of weights of a layer, various sub-tensors may require different number of residuals. Let, \( i \)-th sub-tensor requires \( r_i \) number of residuals. Then, there is total \( \sum_{i=1}^{k} (r_i + 1) \) multiplications, model capacity is \( \sum_{i=1}^{k} 3^{(r_i+1)} - k + 1 \), and model size (in bits) is \((8 + \frac{1}{2}) \sum_{i=1}^{k} (r_i + 1) \), for \( k \) number of sub-tensors and tensor size \( n \).

* **Power-Performance Estimate** : Let \( X \) be the power-performance gain for ternary 8-2 operations over 8-8. Then, power-performance for residual method with \( C \times C \) compute using FGQ sub-tensor size \( N \) can be shown as \( X / (CX/N + 1) \).

### 3 EXPERIMENTS

We use pre-trained FP-32 GoogleNet3 [21], ResNet-101 [9], and AlexNet [14] models for ImageNet classification task. For 8-bit quantization, we have used the low-precision dynamic fixed point technique mentioned in [18]. Results of ternary residual networks are shown in Table 1. For power-performance gain, we estimate \( X \approx 5.5 \) for \( N = 64 \), which leads to an estimated power-performance gain of \( \sim 2 \times \) over 8-8 for accuracy \% off from FP32 results. For comparison, we mention a few results of other sub-8-bit networks on ImageNet using AlexNet. (1) Binarized weights and activations of [20] incurred a loss of \( \sim 12 \%), (2) the loss of binary weights and 2-bit activations of [28] was \( \sim 6 \% \) from FP-32, and (3) [111] with binary weight and 2-bit activations reduced the loss to \( \sim 5.5 \% \) from FP-32, (4) using FGQ with \( N = 4 \) (75\% elimination of multiplications), [18] achieved \( \sim 7.8 \% \) loss from FP-32 using ternary weights and 4-bit activations without any low-precision re-training. Note, however, that (1) and (2) used FP-32 weights and activations for the first and last layers. A detailed analysis on power-performance for various methods is beyond the scope of this paper.

---

**Table 1: Ternary Residual Networks using FGQ. # sub-tensors and compute for FGQ ternary is \( 1 \times \). Loss is w.r.t FP32.**

<table>
<thead>
<tr>
<th>Network</th>
<th>Loss ( \sim 1 %)</th>
<th>Loss ( \sim 2 %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-101</td>
<td>2.3%</td>
<td>2.5%</td>
</tr>
<tr>
<td>GoogLeNet v3</td>
<td>2.5%</td>
<td>2.5%</td>
</tr>
<tr>
<td>AlexNet</td>
<td>2.9%</td>
<td>2.9%</td>
</tr>
</tbody>
</table>
REFERENCES


