Iteratively Training Look-Up Tables for Network Quantization

Fabien Cardinaux* Sony Europe Ltd.†

Stefan Uhlich* Sony Europe Ltd.†

Kazuki Yoshiyama* Sony Corporation‡

Javier Alonso García Sony Europe Ltd.†

Stephen Tiedemann Sony Europe Ltd.†

Thomas Kemp Sony Europe Ltd.†

Akira Nakamura Sony Corporation‡

Abstract

Operating deep neural networks on devices with limited resources requires the reduction of their memory footprints and computational requirements. In this paper we introduce a training method, called *look-up table quantization, LUT-Q*, which learns a dictionary and assigns each weight to one of the dictionary's values. We show that this method is very flexible and that many other techniques can be seen as special cases of LUT-Q. For example, we can constrain the dictionary trained with LUT-Q to generate networks with pruned weight matrices or restrict the dictionary to powers-of-two to avoid the need for multiplications. In order to obtain *fully multiplier-less* networks, we also introduce a multiplier-less version of batch normalization. Extensive experiments on image recognition and object detection tasks show that LUT-Q consistently achieves better performance than other methods with the same quantization bitwidth.

1 Introduction and Proposed Training Method

In this paper, we propose a training method for reducing the size and the number of operations of a *deep neural network* (DNN) that we call *look-up table quantization* (LUT-Q). As depicted in Fig. 1, LUT-Q trains a network that represents the weights $\mathbf{W} \in \mathbb{R}^{O \times I}$ of one layer by a dictionary $\mathbf{d} \in \mathbb{R}^K$ and assignments $\mathbf{A} \in [1, \ldots, K]^{O \times I}$ such that $Q_{oi} = d_{A_{oi}}$, i.e., elements of $\mathbf{Q}$ are restricted to the K dictionary values in $\mathbf{d}$. To learn the assignment matrix $\mathbf{A}$ and dictionary $\mathbf{d}$, we iteratively update them after each minibatch. Our LUT-Q algorithm, run for each mini-batch, is summarized in Table 1.

LUT-Q has the advantage to be very flexible. By simple modifications of the dictionary $\mathbf{d}$ or the assignment matrix $\mathbf{A}$, it can implement many weight compression schemes from the literature. For example, we can constrain the assignment matrix and the dictionary in order to generate a network with pruned weight matrices. Alternatively, we can constrain the dictionary to contain only the values $\{-1, 1\}$ and obtain a *Binary Connect Network* [4], or to $\{-1, 0, 1\}$ resulting in a *Ternary Weight Network* [13]. Furthermore, with LUT-Q we can also achieve *Multiplier-less networks* by either choosing a dictionary $\mathbf{d}$ whose elements d_k are of the form $d_k \in \{\pm 2^{b_k}\}$ for all $k = 1, \ldots, K$ with $b_k \in \mathbb{Z}$, or by rounding the output of the k-means algorithm to powers-of-two. In this way we can learn networks whose weights are powers-of-two and can, hence, be implemented without multipliers.

The memory used for the parameters is dominated by the weights in affine/convolution layers. Using LUT-Q, instead of storing $\mathbf{W}$, the dictionary $\mathbf{d}$ and the assignment matrix $\mathbf{A}$ are stored. Hence, for an affine/convolution layer with N parameters, we reduce the memory usage in bits from NB_{float}

*Equal contribution: fabien.cardinaux@sony.com, stefan.uhlich@sony.com, kazuki.yoshiyama@sony.com

†Sony European Technology Center, Stuttgart, Germany

‡R&D Center, Tokyo, Japan

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

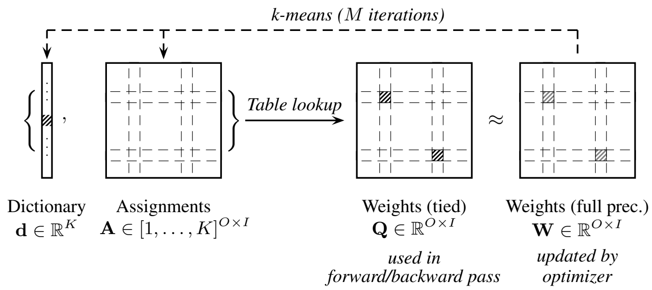

Figure 1: Proposed look-up table quantization scheme.

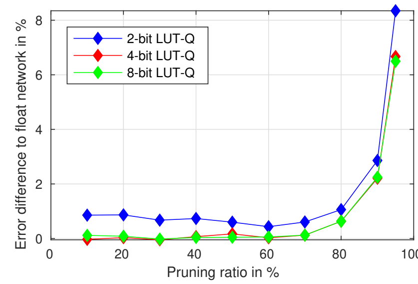

Figure 2: CIFAR-10: Val. error for LUT-Q with pruning.

to just $KB_{\text{float}} + N \lceil \log_2 K \rceil$, where B_{float} is the number of bits used to store one weight. Furthermore, using LUT-Q we also achieve a reduction in the number of computations: for example, affine layers trained using LUT-Q need to compute just K multiplications at inference time, instead of I multiplications for a standard affine layer with I input nodes.

2 Experiments

For the description of our results we use the following naming convention: *Quasi multiplier-less* networks avoid multiplications in all affine/convolution layers, but they are not completely multiplier-less since they contain multiplications in standard *batch normalization* (BN) layers. For example, the networks described in [24] are quasi multiplier-less. *Fully multiplier-less* networks avoid all multiplications at all as they use our multiplier-less BN (see appendix A). Finally, we call all other networks *unconstrained*.

We conducted extensive experiments with LUT-Q and multiplier-less networks on the CIFAR-10 image classification task [11], on the ImageNet ILSVRC-2012 task [21] and on the Pascal VOC object detection task [5]. All experiments are carried out with the *Sony Neural Network Library*[4].

For CIFAR-10, we first use the full precision 32-bit *ResNet-20* as reference (7.4% error rate). Quasi multiplier-less networks using LUT-Q achieve 7.6% and 8.0% error rate for 4-bit and 2-bit quantization respectively. Fully multiplier-less networks with LUT-Q achieve 8.1% and 9.0% error rates, respectively. LUT-Q can also be used to prune and quantize networks simultaneously. Fig. 2 shows the error rate increase between the baseline full precision *ResNet-20* and the pruned and quantized network. Using LUT-Q we can prune the network up to 70% and quantize it to 2-bit without significant loss in accuracy.

For Imagenet, we used *ResNet-18*, *ResNet-34* and *ResNet-50* [8] as reference networks. We report their validation error in Table 2. In Table 2, we compare LUT-Q against the published results using the INQ approach [24], which also trains networks with power-of-two weights. We also compare with the baseline reported [15] which correspond the best results from the literature for each weight and quantization configuration. Note that we cannot directly compare the results of this *appentrice* method [15] itself because they do not quantize the first and last layer of the *ResNets*. We observe that LUT-Q always achieves better performance than other methods with the same weight and activation bitwidth except for *ResNet-18* with 2-bit weight and 8-bit activation quantization. Remarkably, *ResNet-50* with 2-bit weights and 8-bit activations achieves 26.9% error rate which is only 1.0% worse than the baseline. The memory footprint for parameters and activations of this network is only 7.4MB compared to 97.5MB for the full precision network. Furthermore, the number of multiplications is reduced by two orders of magnitude and most of them can be replaced by bit-shifts.

Finally, we evaluated LUT-Q on the Pascal VOC [5] object detection task. We use our implementation of YOLOv2 [19] as baseline. This network has a memory footprint of 200MB and achieves a *mean average precision (mAP)* of 72% on Pascal VOC. We were able to reduce the total memory footprint by a factor of 20 while maintaining the mAP above 70% by carrying out several modifications: replacing the feature extraction network with traditional residual networks [8], replacing the convolution layers by factorized convolutions[5], and finally applying LUT-Q in order to quantize the weights of the network to 8-bit. Using LUT-Q with 4-bit quantization we are able to further reduce the total memory footprint down to just 1.72MB and still achieve a mAP of about 64%.

[4]Neural Network Libraries by Sony: https://nnabla.org/

[5]Each convolution is replaced by a sequence of pointwise, depthwise and pointwise convolutions (similarly to MobileNetV2 [22]

Table 1: LUT-Q training algorithm

// Step 1: Compute tied weights
for $l = 1$ **to** L **do**
  $\mathbf{Q}^{(l)} = \mathbf{d}^{(l)}[\mathbf{A}^{(l)}]$
end for

// Step 2: Compute current cost and gradients
$C = \text{Loss}\left(\mathbf{T}, \text{Forward}\left(\mathbf{X}, \mathbf{Q}^{(1)}, \ldots, \mathbf{Q}^{(L)}\right)\right)$

$$\left\{\mathbf{G}^{(1)}, \ldots, \mathbf{G}^{(L)}\right\} = \left\{\frac{\partial C}{\partial \mathbf{Q}^{(1)}}, \ldots, \frac{\partial C}{\partial \mathbf{Q}^{(L)}}\right\} = \text{Backward}\left(\mathbf{X}, \mathbf{T}, \mathbf{Q}^{(1)}, \ldots, \mathbf{Q}^{(L)}\right)$$

// Step 3: Update full precision weights (here: SGD)
for $l = 1$ **to** L **do**
  $\mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \eta \mathbf{G}^{(l)}$
end for

// Step 4: Update weight tying by M k-means iterations
for $l = 1$ **to** L **do**
  for $m = 1$ **to** M **do**
    $A_{ij}^{(l)} = \underset{k=1,\ldots,K^{(l)}}{\arg\min} \left| W_{ij}^{(l)} - d_k^{(l)} \right|$
    for $k = 1$ **to** $K^{(l)}$ **do**
      $d_k^{(l)} = \frac{1}{\sum_{ij,\, A_{ij}^{(l)}=k} 1} \sum_{ij,\, A_{ij}^{(l)}=k} W_{ij}^{(l)}$
    end for
  end for
end for

Table 2: ImageNet: LUT-Q compared to other quantization methods.
✓✓: fully multiplier-less. ✓: quasi multiplier-less. ×: unconstrained.

Quantization Weights	Quantization Activations	Source	Multiplier-less	Validation error ResNet-18	Validation error ResNet-34	Validation error ResNet-50
32-bit	32-bit	our implementation	×	31.0%	28.1%	25.9%
5-bitpow-2	32-bit	INQ [24]	✓	31.0%	-	25.2%
4-bitpow-2	32-bit	INQ [24]	✓	31.1%	-	-
4-bit	8-bit	*apprentice* [15]	×	33.6%	29.7%	28.5%
4-bitpow-2	8-bit	**LUT-Q pow-2**	✓	31.6%	28.1%	25.5%
4-bitpow-2	8-bit	**LUT-Q pow-2**	✓✓	35.1%	30.7%	26.9%
2-bitpow-2	32-bit	INQ [24]	✓	34.0%	-	-
2-bit	32-bit	*apprentice* [15]	×	33.4%	28.3%	26.1%
2-bitpow-2	32-bit	**LUT-Q pow-2**	✓	31.8%	-	-
2-bit	8-bit	*apprentice* [15]	×	33.9%	30.8%	29.2%
2-bitpow-2	8-bit	**LUT-Q pow-2**	✓	35.8%	30.5%	26.9%
2-bitpow-2	8-bit	**LUT-Q pow-2**	✓✓	43.2%	35.2%	29.8%

3 Comparison to state-of-the-art

Different *compression* methods were proposed in the past in order to reduce the memory footprint and the computational requirements of DNNs: pruning [6, 12], quantization [2, 7, 23], teacher-student network training [9, 15, 18, 20] are some examples. In general, we can classify the methods for quantization of the parameters of a neural network into three types:

- *Soft weight sharing*: These methods train the full precision weights such that they form clusters and therefore can be more efficiently quantized [1, 2, 14, 17, 23].
- *Fixed quantization*: These methods choose a dictionary of values beforehand to which the weights are quantized. Afterwards, they learn the assignments of each weight to the dictionary entries. Examples are *Binary Neural Networks* [4], *Ternary Weight Networks* [13] and also [15, 16].
- *Trained quantization*: These methods learn a dictionary of values to which weights are quantized during training. However, the assignment of each weight to a dictionary entry is fixed [7].

Our LUT-Q approach takes the best of the latter two methods: For each layer, we jointly update both dictionary and weight assignments during training. This approach to compression is similar to *Deep Compression* [7] in the way that we learn a dictionary and assign each weight in a layer to one of the dictionary's values using the k-means algorithm, but we update iteratively both assignments and dictionary at each mini-batch iteration.

4 Conclusions and Future Perspectives

We have presented look-up table quantization, a novel approach for the reduction of size and computations of deep neural networks. After each minibatch update, the quantization values and assignments are updated by a clustering step. We show that the LUT-Q approach can be efficiently used for pruning weight matrices and training multiplier-less networks as well. We also introduce a new form of batch normalization that avoids the need for multiplications during inference.

As argued in this paper, if weights are quantized to very low bitwidth, the activations may dominate the memory footprint of the network during inference. Therefore, we perform our experiments with activations quantized uniformly to 8-bit. We believe that a non-uniform activation quantization, where the quantization values are learned parameters, will help quantize activations to lower precision. This is one of the promising directions for continuing this work.

Recently, several papers have shown the benefits of training quantized networks using a *distillation* strategy [9, 15]. Distillation is compatible with our training approach and we are planning to investigate LUT-Q training together with distillation.

References

[1] Achterhold, J., Koehler, J. M., Schmeink, A., and Genewein, T. Variational network quantization. *International Conference on Learning Representations (ICLR)*, 2018.

[2] Chen, W., Wilson, J., Tyree, S., Weinberger, K., and Chen, Y. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning (ICML)*, pp. 2285–2294, 2015.

[3] Courbariaux, M. and Bengio, Y. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016.

[4] Courbariaux, M., Bengio, Y., and David, J.-P. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pp. 3123–3131, 2015.

[5] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J. M., and Zisserman, A. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88:303–338, 2009.

[6] Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015.

[7] Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

[8] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[9] Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[10] Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[11] Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. 2009.

[12] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pp. 396–404, 1990.

[13] Li, F., Zhang, B., and Liu, B. Ternary weight networks. In *NIPS Workshop on Efficient Methods for Deep Neural Networks (EMDNN)*, 2016.

[14] Louizos, C., Ullrich, K., and Welling, M. Bayesian compression for deep learning. *Conference on Neural Information Processing Systems (NIPS)*, 2017.

[15] Mishra, A. and Marr, D. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *International Conference on Learning Representations (ICLR)*, 2018.

[16] Mishra, A., Nurvitadhi, E., Cook, J. J., and Marr, D. Wrpn: Wide reduced-precision networks. *arXiv preprint arXiv:1709.01134*, 2017.

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

[18] Polino, A., Pascanu, R., and Alistarh, D. Model compression via distillation and quantization. *International Conference on Learning Representations (ICLR)*, 2018.

[19] Redmon, J. and Farhadi, A. Yolo9000: better, faster, stronger. *arXiv preprint*, 2017.

[20] Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. Fitnets: Hints for thin deep nets. In *International Conference on Learning Representations (ICLR)*, 2015.

[21] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[22] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[23] Ullrich, K., Meeds, E., and Welling, M. Soft weight-sharing for neural network compression. In *International Conference on Learning Representations (ICLR)*, 2017.

[24] Zhou, A., Yao, A., Guo, Y., Xu, L., and Chen, Y. Incremental network quantization: Towards lossless CNNs with low-precision weights. In *International Conference on Learning Representations (ICLR)*, 2017.

A Multiplier-less Batch Normalization

From [10] we know that the traditional *batch normalization* (BN) at inference time for the *o*th output is

$$y_o = \gamma_o \frac{x_o - \mathrm{E}\left[x_o\right]}{\sqrt{\mathrm{VAR}\left[x_o\right] + \epsilon}} + \beta_o, \tag{1}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the input and output vectors to the BN layer, $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are parameters learned during training, $\mathrm{E}\left[\mathbf{x}\right]$ and $\mathrm{VAR}\left[\mathbf{x}\right]$ are the running mean and variance of the input samples, and ϵ is a small constant to avoid numerical problems. During inference, $\boldsymbol{\gamma}$, $\boldsymbol{\beta}$, $\mathrm{E}\left[\mathbf{x}\right]$ and $\mathrm{VAR}\left[\mathbf{x}\right]$ are constant and, therefore, the BN function (1) can be written as

$$y_o = a_o \cdot x_o + b_o, \tag{2}$$

where we use the scale $a_o = \gamma_o / \sqrt{\mathrm{VAR}[x_o] + \epsilon}$ and offset $b_o = \beta_o - \gamma_o \, \mathrm{E}\left[x_o\right] / \sqrt{\mathrm{VAR}\left[x_o\right] + \epsilon}$. In order to obtain a multiplier-less BN, we require $\mathbf{a}$ to be a vector of powers-of-two during inference. This can be achieved by quantizing $\boldsymbol{\gamma}$ to $\hat{\boldsymbol{\gamma}}$. The quantized $\hat{\boldsymbol{\gamma}}$ is learned with the same idea as for WT: During the forward pass, we use traditional BN with the quantized $\hat{\boldsymbol{\gamma}} = \hat{\mathbf{a}} / \sqrt{\mathrm{VAR}[\mathbf{x}] + \epsilon}$ where $\hat{\mathbf{a}}$ is obtained from $\mathbf{a}$ by using the power-of-two quantization. Then, in the backward pass, we update the full precision $\boldsymbol{\gamma}$. Please note that the computations during training time are not multiplier-less but $\hat{\boldsymbol{\gamma}}$ is only learned such that we obtain a multiplier-less BN during inference time. This is different to [3] which proposed a shift-based batch normalization using a different scheme that avoids all multiplications in the batch normalization operation by rounding multiplicands to powers-of-two in each forward pass. Their focus is on speeding up training by avoiding multiplications during training time, while our novel multiplier-less batch normalization approach avoids multiplications during inference.