POSTER CONTACTS: goncalo.mordido@hpi.de, matthijsv@nvidia.com

Instant Quantization of Neural Networks using Monte Carlo Methods

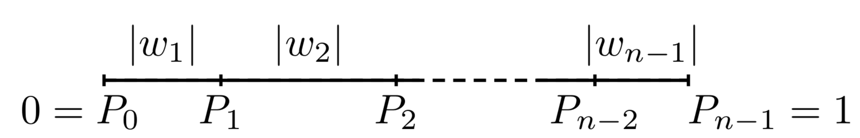
Gonçalo Mordido (HPI), Matthijs Van keirsbilck (NVIDIA), Alexander Keller (NVIDIA)

Linking Monte Carlo Methods and Neural Networks

Subsampling Networks

Monte Carlo methods are ubiquitous in neural networks [1]: dropout, weight noise regularization, weight normalization, pruning/sparsification, initialization, etc. all rely on random sampling. In fact, various pruning techniques may be considered special cases of importance sampling [2].

Normalizing the weights of a neuron, they form a probability distribution function of the connections to the next layer:



Biases and incoming weights for neuron j in layer l may then be normalized by

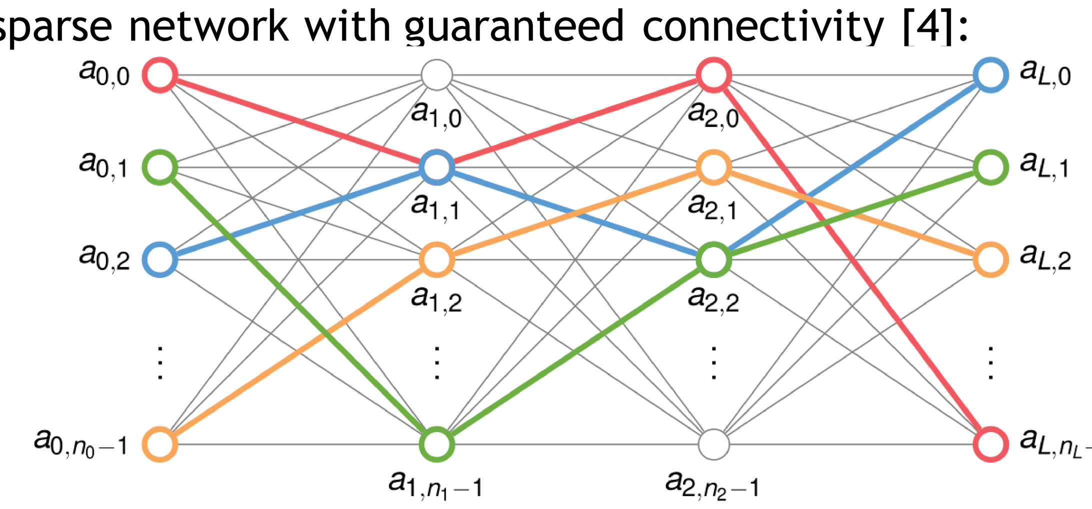
$$f = \|\mathbf{w}_{l-1,j}\|_1 = \sum_{i=0}^{N_{l-1}-1} |w_{l-1,i,j}|$$

Effectively, a neuron integrates its inputs and hence may be approximated using stochastic sampling. Drawing N samples according to the above probability distribution function and considering the sign of the weight, we approximate

$$\sum_{j=0}^{n-1} w_j a_j \approx \frac{1}{N \cdot f} \sum_{i=0}^{N-1} \underbrace{\text{sign}(w_{j_i})}_{\in\{-1,1\}} \times a_{j_i},$$

which in the limit converges to the true value. By varying the number of samples, sparsity and computational cost can be traded for accuracy. Various sampling techniques may be used to improve the approximation (jittered sampling, sorting, ...).

Tracing paths through a network yields a linear-complexity, sparse network with guaranteed connectivity [4]:



The same technique can be used to construct sparse networks from scratch. The connectivity may be optimized for memory accesses in hardware accelerators [1,3].

Monte Carlo Quantization

Quantizing neural network weights and activations to low bit-width integers enables higher efficiency, especially with respect to power consumption.

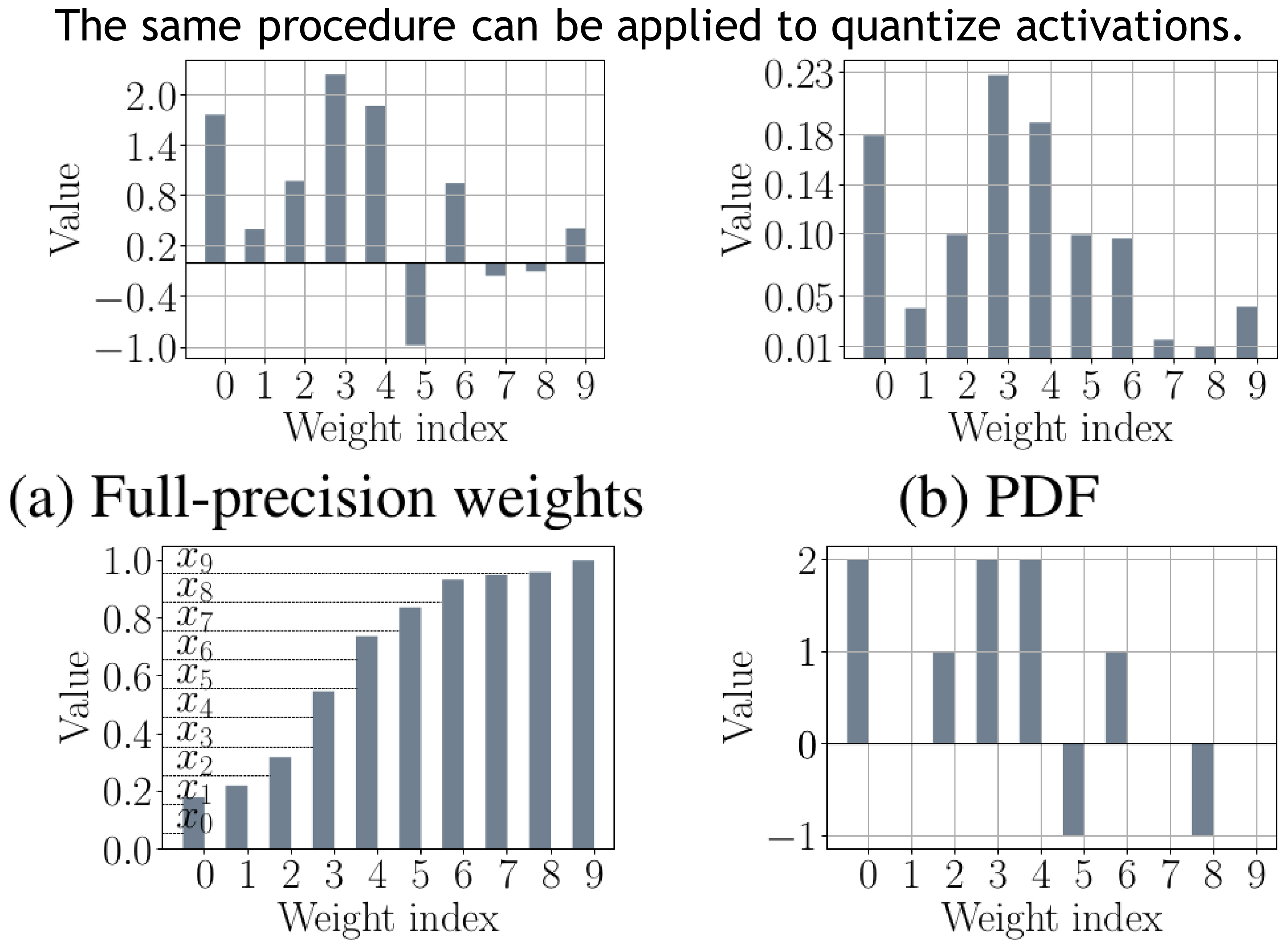
Importance sampling can be used to generate a lower-complexity version of a neural network by quantizing weights and/or activations, with no modification to the training procedure.

Importance sampling automatically results in a sparse, quantized network, where the complexity is proportional to the number of samples taken. We use jittered equidistant sampling to improve the quality of the approximation.

Monte Carlo quantization applies the following steps layer-by-layer:

1. Create a probability density function (PDF) for all $N_{l,w}$ weights of layer l such that $\sum_{i=0}^{N_{l,w}-1} |w_{l,i}| = 1$
2. Perform importance sampling on the weights by sampling from the cumulative density function (CDF) of the magnitudes of the weights, and counting the number of hits per weight
3. The hit count corresponds to a low bit-width, integer representation of the weight

The same procedure can be applied to quantize activations.

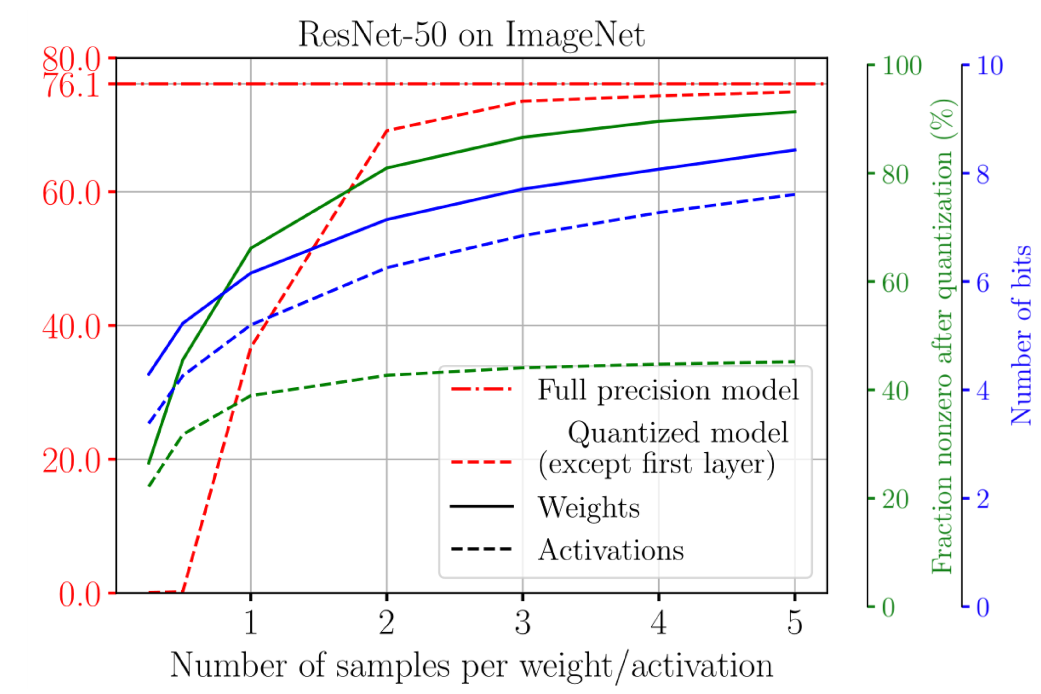


(a) Full-precision weights (b) PDF

(c) sampling on CDF (d) integer weights

Results

The figure below shows the accuracy on ImageNet for models with quantized weights and activations, where the number of samples is varied. The red lines show accuracy for full precision and quantized models. The blue and green lines show the number of bits and the density respectively, for weights (solid line) and activations (dashed line).



With just 5 binary samples per weight we reach within 0.2% of full precision (32-bit) accuracy, and 1.2% if activations are quantized as well. We require about 8 bits for storing the largest weights and activations.

MCQ also performs well in various Natural Language Processing tasks, illustrating its general applicability.

MCQ on language modeling (LM), speech recognition (SR), and machine translation (MT).

Task	Dataset	Model	Metric	Full Precision	Δ MCQ (quantized w)
LM	WikiText-103	Transformer	Perplexity ↓	18.7	+0.21 (8.21w-32a)
LM	WikiText-2	LSTM 2x650	Perplexity ↓	71.05	+0.51 (7.17w-32a)
SR	VCTK	DeepSpeech2	CER ↓	7.00	+0.09 (7.26w-32a)
MT	WMT14 en-fr	Transformer	BLEU ↑	40.83	-0.23 (7.71w-32a)

References

[1] Keller, Van keirsbilck, Yang. (2019): GTC S9389: “Structural Sparsity: Speeding up training and inference of neural networks by linear algorithms

[2] Molchanov et al. (2017): Pruning Convolutional Neural Networks for Resource Efficient Inference

[3] Dey et al. (2017): Interleaver Design for Deep Neural Networks

[4] Gamboa, Keller (2018): GTC S8780: “Monte Carlo Methods and Neural Networks”

[5] Zhao et al. (2019): Improving Neural Network Quantization without Retraining using Outlier Channel Splitting

The 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing @ NeurIPS 2019