Supplementary Material: Boosting Deep Neural Network Efficiency with Dual-Module Inference

1. Motivation for Dual-Module Inference

As shown in Figure 1, the nonlinear activation functions – $sigmoid$ and $tanh$ – have insensitive regions where the output activations are resilient to errors introduced in pre-activation accumulation results.

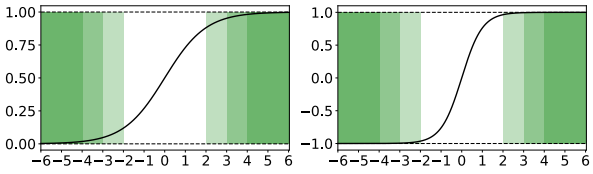

Figure 1. Insensitive (green shaded) and sensitive (white) regions of $sigmoid$ (left) and $tanh$ (right) nonlinear functions.

The selection of which neurons should be in the (in)sensitive region is dynamic and input-dependent, which can be seen in Figure 2. Unlike the static weight sparsity that we can prune the unused connections offline in advance, the dynamic region speculation requires a very lightweight criterion for real-time processing. Taking all these into account, we propose a dual-model inference method that efficiently determines (in)sensitive region and significantly saves the memory access and computational cost.

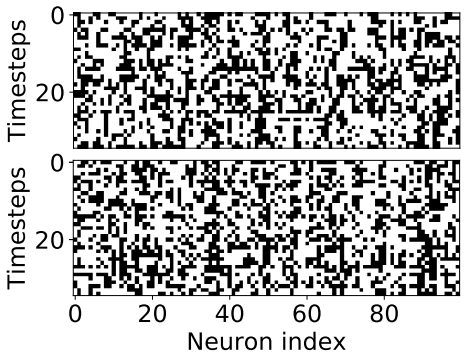

Figure 2. Dynamic region distribution across time-steps and inputs. The white and black colors denote neurons in the insensitive and sensitive regions, respectively. The upper and lower patterns are from different inputs.

2. Experiments

Settings. Our evaluation for single-layer RNNs is adapted from PyTorch's word language modeling example, where the dataset has 10K tokens. We do not use dropout when training the LL module(s); the starting learning rate is 5, and we decay it by four if no loss descent has been seen on the validation dataset. The RNNs used in language modeling have 35 timestamps; the maximum generated sequence length in GNMT is 80. For language modeling, we choose 1500 hidden units following the word language modeling example, and we compare our method which dynamically reduces 50% of weight accesses to the static case where only 750 hidden units are used. Besides single-layer LSTM and GRU, we also evaluate four-layer stacked LSTMs as in GNMT. For GNMT experiments, we use the same set of parameters when training the base model [1]. We train the four *little* modules in the four-layer stacked LSTM in a total of 4 epochs.

Our experiments on ResNet-18 is adapted from the image classification example of PyTorch.

Additional results. In addition to the results of LSTMs using 1500 hidden units in the main text, We observe a similar quality-performance trade-off for LSTM with 750 hidden units as shown in Table 1. Comparing the case of base LSTM with 750 hidden units with dual-module LSTM with 1500 hidden units and 50% insensitive ratio, although the memory access reduction is at the same level, our proposed dual-module approach achieves much better model quality because we kept the expressive power of a larger LSTM layer.

We further report the results using single-layer GRU on word-level language modeling tasks as in Table 2. Using dual-module inference on GRUs expresses the similar quality-performance trade-off as of LSTMs. Our dual-module method is generally applicable to both LSTMs and GRUs. We also measured the execution time of GNMT layer with the hidden size of 1024 and the input size of 2048 in Table 3.

Evaluation on the *little* module. Using one layer in ResNet-18, we first show the histogram of feature map values of both the original module and the *little* module learned and approximated from the original module. As shown in Figure 3, the distribution of the little module exhibits the same as the original module. We then show the visualization of feature maps of both the original, i.e., the *big* module,

[1] From https://github.com/NVIDIA/DeepLearningExamples

Table 1. LSTM perplexity and execution time (ms).

Insensitive ratio	hidden size: 1500				hidden size: 750			
	PPL	Diff.	Time	Speedup	PPL	Diff.	Time	Speedup
Base	80.64	n/a	1.477	1.00x	84.32	n/a	0.546	1.00x
10%	80.72	-0.08	1.315	1.12x	84.42	-0.10	0.448	1.22x
30%	80.56	0.08	1.095	1.35x	84.43	-0.11	0.415	1.32x
50%	81.36	-0.72	0.885	1.67x	84.29	0.03	0.342	1.60x
70%	87.48	-6.83	0.641	2.30x	84.89	-0.57	0.287	1.90x
90%	109.37	-28.73	0.380	3.89x	88.44	-4.12	0.216	2.53x

Table 2. GRU perplexity and execution time (ms).

Insensitive ratio	hidden size: 1500				hidden size: 750			
	PPL	Diff.	Time	Speedup	PPL	Diff.	Time	Speedup
Base	85.48	n/a	1.182	1.00x	89.64	n/a	0.466	1.00x
10%	85.62	-0.14	1.024	1.15x	89.81	-0.17	0.383	1.22x
30%	86.01	-0.53	0.869	1.36x	89.63	0.01	0.334	1.40x
50%	88.73	-3.25	0.726	1.63x	89.69	-0.05	0.302	1.54x
70%	98.09	-12.61	0.545	2.17x	92.51	-2.87	0.284	1.64x
90%	122.75	-37.27	0.350	3.38x	102.37	-12.73	0.198	2.35x

Table 3. GNMT BLEU score and execution time (ms). (1024, 2048) indicates the hidden size is 1024 and the input size is 2048; similarly for (1024, 1024).

Insensitive ratio	Quality		(1024, 1024)		(1024, 2048)	
	BLEU	Diff.	Time	Speedup	Time	Speedup
Base	24.32	n/a	0.838	1.00x	1.092	1.00x
10%	24.33	0.01	0.679	1.23x	0.962	1.14x
30%	24.18	-0.14	0.541	1.55x	0.803	1.36x
50%	23.73	-0.59	0.480	1.75x	0.642	1.70x
70%	21.92	-2.40	0.360	2.33x	0.479	2.28x
90%	11.77	-12.55	0.243	3.45x	0.307	3.56x

and of the *little* module in Figure 4. From the visualized feature map comparison, we can see that the *little* module approximates the orginal module well and represents the almost same features.

3. Comparison with Weight Pruning Method

We compare our proposed dual-module inference approach with the automated gradual pruning method (Zhu & Gupta, 2017), which is a popular pruning method with open implementation[2]. Firstly, compared with weight pruning, our method achieves better quality with practical speedup – 1.54x to 1.75x reduction on wall-clock time – on commodity CPUs while element-wise weight pruning requires specialized hardware to gain real speedup of computation given irregular sparsity. Moreover, our dual-module inference method can be further applied on top of pruned models to reduce execution time by reducing memory access.

[2]From https://github.com/NervanaSystems/distiller

Table 4. Comparison of our proposed dual-module inference (DMI), using 50% insensitive ratio, with weight pruning using one LSTM layer with 1500 units in word language modeling task on WikiText-2 dataset.

Method	PPL w/o DMI	PPL w/ DMI
Dense	85.52	86.21
80% weight sparsity	86.42	88.46
90% weight sparsity	88.75	90.96

References

Zhu, M. and Gupta, S. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

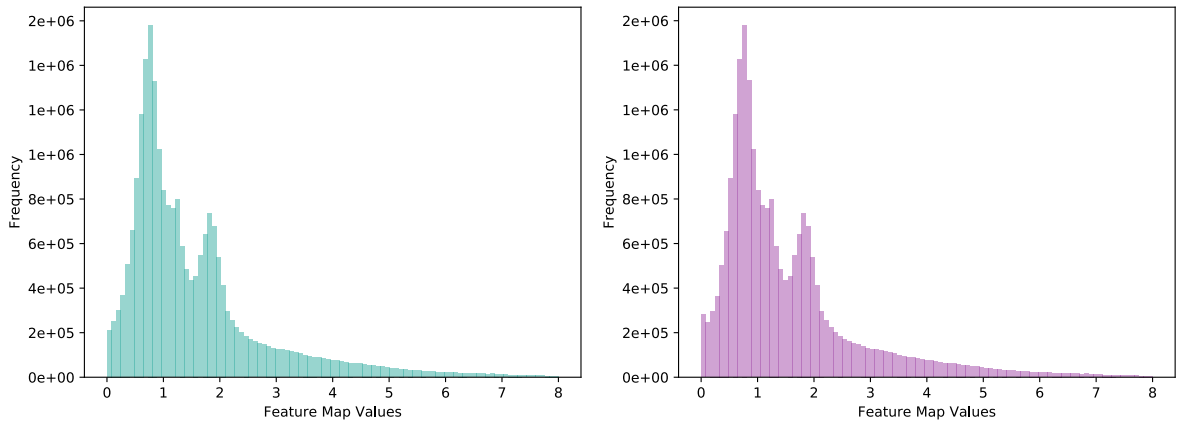

Figure 3. The distribution of the *little* module, on the right, exhibits the same as the original module on the left.

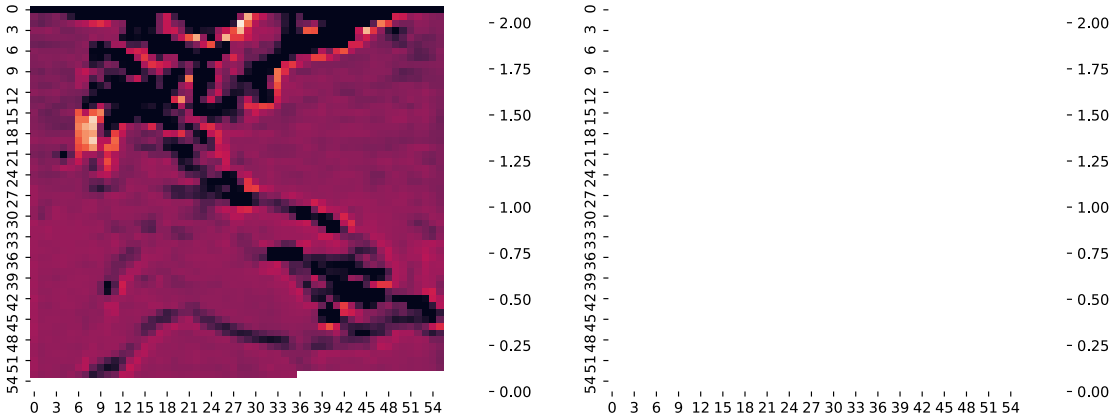

Figure 4. Visualization of the feature maps of the original layer, i.e., the *big* module on the left, and of the *little* module on the right. The *little* module can approximate the original layer well.