CGPA: Coarse-Grained Pruning of Activations for Energy-Efficient RNN Inference

Marc Riera, Jose-Maria Arnau, Antonio Gonzalez
Universitat Politècnica de Catalunya (UPC), Barcelona, Spain

Abstract—Activation pruning has been previously proposed in Convolutional Neural Networks (CNNs) and Multi-Layer Perceptrons (MLPs) to avoid computations by dynamically removing connections whose input value is (almost) zero. However, fine-grained activation pruning results in sparse computations, reducing the efficiency of the hardware implementation, and heavily relies on ReLU activation functions that are unpopular in Recurrent Neural Networks (RNNs). RNN cells perform element-wise multiplications across the activations of different single-layer fully-connected networks, a.k.a. gates, being sigmoid and tanh the activation functions. We show that a significant percentage of activations are saturated towards zero or one in different gates, which results in a large percentage of neuron outputs being multiplied by a close to zero value. We propose Coarse-Grained Pruning of Activations (CGPA) to avoid the computation of entire neurons, rather than individual connections, based on the activation values of the gates. We show that CGPA results in much less sparse computation and memory access patterns than previous proposals and, hence, it can be easily implemented on top of a TPU-like architecture with negligible area overhead, resulting in 12% speedup and 12% energy savings on average for a set of widely used RNNs.

Index Terms—Machine Learning, RNN, Accelerators, Low Energy

1 Introduction

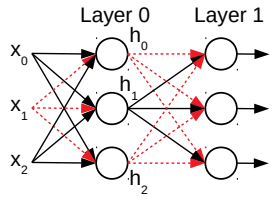
Dynamic operation pruning [1], a.k.a. activation pruning [2], is an optimization technique to reduce the cost of evaluating CNNs and MLPs. Based on the observation that ReLU activations tend to generate a large number of zero values, activation pruning dynamically avoids computations when the synaptic weight is multiplied by a zero-value input. However, such a fine-grained activation pruning results in very sparse computations and memory accesses, since individual connections are selectively computed or skipped [3]. Sparse computations increase the complexity of a hardware accelerator, as it has to handle data representations based on indexes of non-zero values [4], which results in sparse memory access patterns that are challenging for the hardware. On the other hand, ReLU activations are rarely used in Recurrent Neural Networks (RNNs) [5], leading to smaller potential for fine-grained activation pruning.

RNNs represent the state-of-the-art solution for sequence processing problems such as machine translation [6], speech recognition [7] or language modeling [8]. Unlike CNNs or MLPs, RNN units store information from past executions to improve the accuracy of future predictions. Deep RNNs consists of multiple RNN layers, a.k.a. cells, stacked on top of each other. The most successful RNN cell architectures are the Long-Short Term Memory (LSTM) [5] and the Gated Recurrent Unit (GRU) [9]. Figure 2 shows the computations performed in an LSTM cell, whereas Figure 3 shows the GRU equations. As it can bee seen, in both cases the cell consists of multiple single-layer fully-connected networks commonly referred as gates. Furthermore, the evaluation includes element-wise multiplications across the outputs, i.e. activations, of different gates. Equation 5 shows the element-wise operations for LSTM, whereas Equation 10 depicts the GRU's element-wise computations.

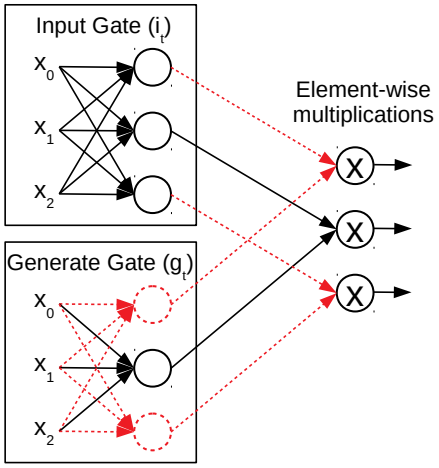
Note that RNN activation functions exhibit a very narrow range: sigmoid ranges from 0 to 1, whereas hyperbolic tangent (tanh) is constrained to the interval $[-1, 1]$. We observe that every time a neuron in the input gate (i_t) of an LSTM cell is saturated towards zero, evaluation of its peer neuron in the generate gate (g_t) can be safely avoided. In a similar manner, whenever the tanh activation in Equation 6 of an LSTM cell is zero the corresponding neuron in the output gate (o_t) can be skipped. Our numbers show that we can avoid more than 18% of the computations and memory accesses by exploiting these zero-value activations in LSTMs. On the other hand, GRU cells exhibit similar behavior: when a neuron is saturated towards one in the update gate (z_t) of a GRU cell, computation of its peer neuron in the generate gate (g_t) can be skipped as the product $(1 - z_t) \times g_t$ is granted to be zero. Our results show that more than 7% of neuron evaluations can be skipped based on this observation.

In this paper, we propose to exploit the aforementioned saturations of activation functions to avoid computation of entire neurons in LSTM and GRU networks. We term this method as Coarse-Grained Pruning of Activations (CGPA).

Figure 1 shows the key difference between fine-grained activation pruning and CGPA. Under fine-grained activation pruning (Figure 1a), individual connections of a fully-connected or convolutional layer are dynamically skipped if their input value is zero, leading to sparse memory accesses. In this example, we assume x_1, h_0 and h_2 are almost zero. In this case, neurons in layer 0 require accessing their respective first and third weights, whereas neurons in layer 1 only require accessing their respective second weight (pruned synapses are shown in red). Such fine-grained and sparse accesses are challenging for hardware. On the other hand, Figure 1b illustrates CGPA. In this example, neurons



(a) Fine-grained activation pruning



(b) Coarse-Grained Pruning of Activations (CGPA)

Fig. 1. Fine-grained (a) vs coarse-grained (b) activation pruning. Connections/neurons shown with red dashed lines are dynamically pruned.

zero and two in the input gate of an LSTM cell are saturated towards zero. This means that neurons zero and two (peer neurons) in generate gate (g_t) can be safely skipped, avoiding all their computations and memory accesses.

Note that CGPA is orthogonal to previous proposals of fine-grained activation pruning and static weight pruning. CGPA is applied at the neuron granularity by exploiting the element-wise operations of RNNs of GRUs and LSTMs and the saturations of the activation functions of the gates. CGPA dynamically prunes entire neurons which generates a compacted pruned model that avoids sparsity, unlike fine-grained activation pruning and static weight pruning which are applied at the connection granularity. Previous works such as Scalpel [10] have shown that in order to exploit the benefits of sparse models in CPU/GPU the degree of fine-grained pruning has to be extremely high, which cannot always be achieved without accuracy loss. In addition, an specific accelerator design to exploit sparse models is required, which in turn increases the complexity of the hardware due the overheads of managing the sparse data. Both fine-grained activation pruning and static weight pruning can provide high percentages of pruning, but they also require the accelerator to be able to manage the sparse data. On the other hand, our CGPA technique can achieve speedups and energy savings with lower pruning degrees, since CGPA requires minor changes to the hardware and can be used in any dense accelerator like TPU.

In this paper, we implement CGPA on top of a Tensor Processing Unit (TPU)-like accelerator. Our technique has a very small area overhead, as it mainly requires a few additional comparators to detect saturation of activation functions and small changes to the control unit to selectively skip entire neurons. In case a neuron is evaluated, all its

$$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + b_i) \quad (1)$$
$$f_t = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \quad (2)$$
$$g_t = \phi(W_{gx}x_t + W_{gh}h_{t-1} + b_g) \quad (3)$$
$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \quad (4)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad (5)$$
$$h_t = o_t \odot \phi(c_t) \quad (6)$$

Fig. 2. LSTM cell Computations. , and are element-wise multiplication, hyperbolic tangent and sigmoid function respectively.

$$z_t = \sigma(W_{zx}x_t + W_{zh}h_{t-1} + b_z) \quad (7)$$
$$r_t = \sigma(W_{rx}x_t + W_{rh}h_{t-1} + b_r) \quad (8)$$
$$g_t = \phi(W_{gx}x_t + W_{gh}(r_t \odot h_{t-1}) + b_g) \quad (9)$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot g_t \quad (10)$$

Fig. 3. Computations performed in a GRU cell. , and are element-wise multiplication, hyperbolic tangent and sigmoid function respectively.

weights and inputs are fetched from memory and injected into the systolic array, whereas they are completely avoided in case the neuron can be safely skipped, reducing sparsity by a large extent compared to fine-grained activation pruning.

This paper focuses on energy-efficient and high-performance RNN inference. We claim the following contributions:

- We show that a significant fraction of activations are saturated towards zero or one in popular RNNs.
- We propose Coarse-Grained Pruning of Activations (CGPA) to avoid the evaluation of entire neurons whenever the outputs of peer neurons are saturated. CGPA significantly reduces the amount of computations and memory accesses while avoiding sparsity by a large extent.
- We implement our technique on top of a TPU-like accelerator. Our numbers show CGPA provides a speedup of 12% and energy savings of 12% on average while it requires a small area overhead of less than 0.003%.

2 Analysis of RNN Activations

Long-Short Term Memory (LSTM) networks and Gated Recurrent Unit (GRU) are the two most successful RNN architectures. Both employ sigmoid and hyperbolic tangent activation functions. This section analyzes the activation values of different RNNs, and introduces a technique that exploits the activation values that are close to zero or one in order to save computations and memory accesses. Table 1 shows the RNNs employed for the analysis. DeepSpeech2 [7] consists of multiple GRU layers employed to perform end-to-end speech recognition. We use two models of DeepSpeech2, a model trained with Librispeech [11] dataset (clean audio from audiobooks) and a Tedlium trained model for spontaneous and noisy speech. NMT [6] is an LSTM network for neural machine translation based on Google Translator trained using the WMT16 dataset with texts of newspapers

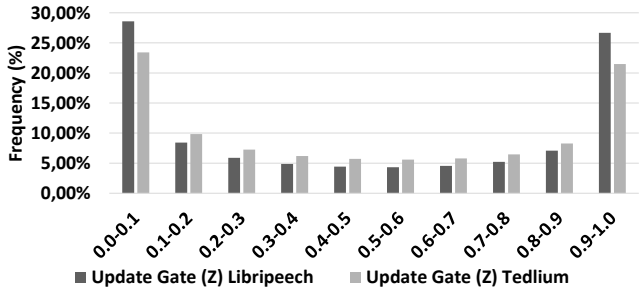
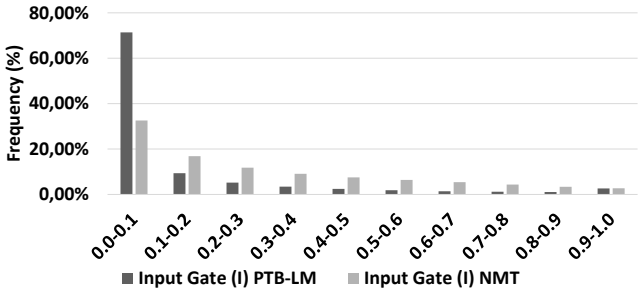
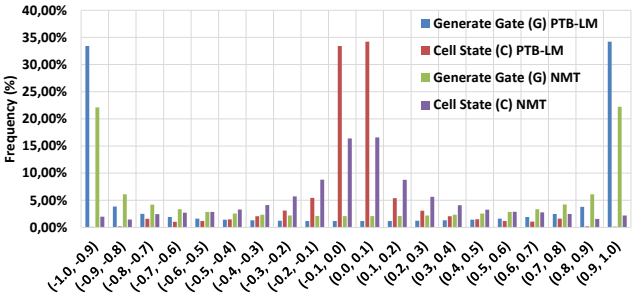


Fig. 4. Histogram of activations of the Update Gate (Z) for DeepSpeech2 using Librispeech and Tedlium.



(a) Histogram of activations of the Input Gate (I)



(b) Histogram of activations of the Generate Gate (G) and Cell State (C)

Fig. 5. Input Gate (I) (a) and Generate Gate (G) and Cell State (C) (b) histogram of activations for the NMT and PTB-LM RNNs.

(DE-EN). Finally, PTB-LM [8] is an LSTM network for language modeling using the Penn Treebank dataset. Accuracy is reported as word error rate (WER), bilingual evaluation understudy (BLEU) and perplexity respectively.

Figure 2 and Figure 3 describe the computations performed in the LSTM and GRU cells respectively. Both are composed of multiple gates that are implemented as single-layer Fully-Connected (FC) networks taking two different inputs: x_t, a.k.a. feed-forward input from previous layer, and $h_{t\ \ 1}$, a.k.a. recurrent input from the same layer. Evaluation of FC layers for the different gates takes most of the computations in an RNN. Careful analysis of the LSTM equations reveals that if a neuron of the input gate (i_t) is saturated to zero, we could save the computations and memory access of its peer neuron in the generate gate (g_t) and vice versa. Similarly, if the hyperbolic tangent of the cell state (c_t) equals to zero, we could save the computations and memory access of its peer neuron in the output gate (o_t) needed for computing the cell output (h_t) . Note that if the output gate (o_t) is saturated to zero we can only save the computations related to the activations since the cell state has to be computed for the next timestep, and if the forget gate (f_t) is saturated to zero, we cannot save computations since the cell state from the previous timestep has already been computed. A similar analysis can be done to the GRU equations where we could save computations and memory accesses of neurons from the generate gate (g_t) when the peer neuron in the update gate (z_t) is saturated to one. Note that in case that a neuron of the generate gate (g_t) is zero, the update gate (z_t) still has to be computed since it is multiplied by the output cell of the previous timestep.

Figure 4 shows the histogram of activation values of the GRU's update gate (Z) for DeepSpeech2 trained with Librispeech and Tedlium datasets in steps of 0.1. The range of the histogram is from 0 to 1 since the sigmoid is used as activation function. We found that between $40-60\%$ of the values are near the saturation points of $0-0.1$ and $0.9-1.0$ (i.e. $20-30\%$ at each end of the histogram), while the rest of the values are equally distributed. Therefore, a significant percentage of the activations of the update gate (z_t) are close to the saturation points (zero or one), pointing out potential to save computations.

Figure 5a shows the histogram of the activation values (sigmoid activation) of the LSTM's input gate (I) for PTB-LM and NMT. We can see a high concentration of values in the range of $0-0.1$, that is 70% for the PTB-LM and 33% for the NMT network. On the other hand, Figure 5b shows the histogram of activations of the generate gate (G) and the cell state (C) for PTB-LM and NMT. The range of the histograms is from -1 to 1 since the hyperbolic tangent is used as activation. We can see that the activation values of the cell state (C) are concentrated around zero: more than 60% of activations in PTB-LM and more than 30% in NMT are in the range $[-0.1, 0.1]$. Furthermore, the activation values of the generate gate (G) are concentrated at both ends of the histogram, with values close to -1 or 1. The large percentage of near-zero activations in both input gate (I) and cell state (C) points out significant potential for saving computations in LSTM cells.

In this paper, we propose to avoid computations of entire neurons in LSTM and GRU cells whenever the activation of some gates is close to zero or one. To this end, we define low and high thresholds so that activation values that are lower than a small threshold t_l are set to zero and all the activation values that are higher than t_h in absolute value are set to one. We apply these thresholds in gates that allow saving computations, i.e. update gate (z_t) in GRU cells and input gate (i_t) and cell state (c_t) in LSTM cells. We call this technique Coarse-Grained Pruning of Activations (CGPA) since it prunes whole neurons from the gates.

Setting appropriate thresholds is key to achieve a good trade-off between savings in computations and accuracy. We evaluated different high and low thresholds for our set of RNNs ranging from $0.9-1.0$ and $0.0-0.1$ with steps of 0.01. In general, a high threshold (t_h) of 0.95 and a low threshold (t_l) of 0.05 provide significant savings with negligible accuracy loss across our set of RNNs. However, we have empirically selected a particular set of thresholds for each RNN in order to maximize the savings while maintaining the accuracy.

TABLE 1
RNNs employed for the evaluation. Only LSTM and GRU layers are included since these layers take up the bulk of computations in RNNs.

DEEPSPEECH2 (145MB)				NMT (800MB)				PTB-LM (250MB)			
TEDLIUM WER: 29.2% - LIBRISPEECH WER: 10.2%				BLEU: 29.8%				PERPLEXITY: 78.1%			
LAYER	INPUT DIM	OUTPUT DIM	CELL DIM	LAYER	IN DIM	OUT DIM	CELL DIM	LAYER	IN DIM	OUT DIM	CELL DIM
BIGRU1	1472	800	800	BILSTM1	2048	2048	1024	UNILSTM1	3000	1500	1500
BIGRU2	1600	800	800	UNILSTM2	3072	1024	1024	UNILSTM2	3000	1500	1500
BIGRU3	1600	800	800	UNILSTM3	2048	1024	1024				
BIGRU4	1600	800	800	UNILSTM4	2048	1024	1024				
BIGRU5	1600	800	800	UNILSTM5	3072	1024	1024				
				UNILSTM6	3072	1024	1024				
				UNILSTM7	3072	1024	1024				
				UNILSTM8	3072	1024	1024				

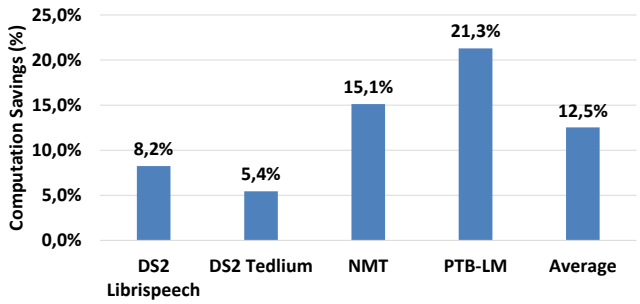


Fig. 6. Computation Savings for our set of RNNs.

Note that we are introducing a small distortion in the RNNs to clamp some activations to zero or one. In order to improve the effectiveness of CGPA, we can include the thresholds during the training of the RNN, so the models learns to compensate these small deviations in the activations. As a proof of concept, we verified that including CGPA during training results in larger savings: in NMT the savings in computations are increased by a factor of approximately 3x with the same negligible accuracy loss. In other words, training with CGPA allows to use more generous thresholds in order to increase the savings, since more activations are clamped to zero or one, while retaining accuracy by a large extent. Accuracy loss has been computed in absolute terms for each of our RNNs. The NMT network accuracy loss is 0.8 in absolute terms which means that the BLEU has been reduced from 29.8% to 29.0%. The PTB-LM network accuracy loss is 2.43 in perplexity while the DeepSpeech2 accuracy loss with Librispeech and Tedlium is 0.92 and 1.65 of the WER respectively.

We have used our selection of thresholds to compute the number of activations that are saturated layer-wise. We have observed that the number of saturations in the update gate (z_t) of the RNNs with GRU cells are homogeneous across most of the layers. DeepSpeech2 with Librispeech and Tedlium have on average 24% and 15% saturations per layer respectively. On the other hand, the LSTM networks have more variance across layers with a minimum of 40% and a maximum of 60% saturations per layer taking into account both the input gate (i_t) and cell state (c_t) activations. Figure 6 shows the computation savings that are achieved for our set of RNNs. We can see that the computation savings are directly related to the histograms of the activations. On average, 12.5% of the computations and memory accesses can be avoided with a small accuracy loss of less than 1.5%.

3 IMPLEMENTATION

This section describes the implementation of CGPA on top of a hardware accelerator and the evaluation methodology. We have evaluated the baseline RNNs including our CGPA technique on a well known Google TPU-like architecture using ScaleSim [12], a cycle accurate simulator for neural network accelerators. ScaleSim models a systolic array architecture which is specialized on CNNs but supports any kind of neural network, including RNNs, like TPU does.

Figure 7 shows a high-level schematic of the architecture. The main components are the blocks of SRAM used for the inputs, outputs and weights, and the systolic array of processing elements (PEs). Each PE includes a MAC (multiplier-accumulator) and some registers. We set the configuration parameters to match the TPU: a systolic array of $256x256$ PEs, 24 MB of total on-chip SRAM and a frequency of 700 MHz. However, we use an output stationary dataflow, as we empirically found that it is more efficient than weight stationary (originally used in TPU) for our set of RNNs. In output stationary, each PE computes the output of a different neuron. Therefore, the weights are stored and forwarded from top to bottom and the inputs are injected in the systolic array from left to right. As in TPU, the inputs and weights are quantized to integers of 8/16-bits without any accuracy loss so that the operations of the convolutional and fully-connected layers are performed with integers. However, the intermediate layers and the activation functions are performed in 32-bit floating point to maintain the accuracy. Regarding main memory, we model an LPDDR4 of 8 GB with a bandwidth of 16 GB/s (dual channel). CGPA requires a floating point comparator to determine which neurons must be executed depending on the activation values. The comparator is used for each neuron to determine whether its activation is smaller/larger than the given threshold. In addition, a small SRAM buffer of 512B is included to store the resulting bitmask from the comparisons. The bitmask contains one bit per neuron indicating whether the peer neuron in another gate has to be computed or skipped. All these overheads are taken into account for the area, timing and energy evaluation of the accelerator.

Regarding area and energy consumption, the array of MACs is implemented in Verilog and synthesized to obtain the power using the Synopsys Design Compiler with the technology library of 28/32nm. On the other hand, we characterize the on-chip SRAM by obtaining the delay, energy per access and area using CACTI-P. Finally, the energy consumption of main memory is estimated by using the MICRON power model for LPDDR4. The results obtained with the aforementioned tools are combined with the ac-

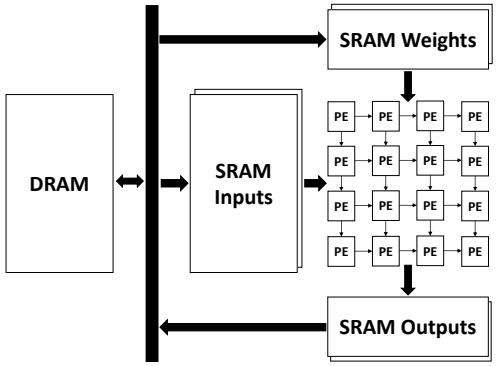


Fig. 7. TPU-like architecture modeled with ScaleSim.

Algorithm 1 Accelerator GRU Execution

```
High Threshold (t_h) = Constant
for Timestep (t) do
  Compute Update Gate (z_t);
  Compute Reset Gate (r_t);
  fAccelerator Executes Multiple Neurons in Parallelg
  for Neuron (n) do
    if z_tn  t_h then
      fGenerate Gate (g_tn) Computations Avoidedg
      Cell Output (h_tn) = h_tn 1;
    else
      Compute Generate Gate (g_tn);
      Compute Cell Output (h_tn);
    end if
  end for
end for
```

tivity factors and memory traces provided by ScaleSim to obtain the dynamic and static power of the accelerator.

The baseline accelerator has been extended to implement CGPA for GRU and LSTM cells. The execution of GRU cells under CGPA is described in the pseudo-code Algorithm 1. The accelerator starts by computing all the neurons of the update gate (z_t) and reset gate (r_t). Each neuron activation of the update gate is compared with the high threshold to generate a bitmask that determines which neurons from the generate gate are executed. The bitmask is stored in a small SRAM buffer. Finally, the generate gate and cell output are computed by checking the bitmask from the buffer. If the activation of the update gate is greater or equal than the high threshold for a given neuron, its corresponding bit in the bitmask will be zero indicating that the computations of the peer neuron from the generate gate can be avoided. In this case, the output of the cell will be the same as the output of the previous timestep. Otherwise, the generate gate and the cell output is computed as usual. The control unit is modified to schedule multiple neurons of the generate gate in the array of PEs after checking the bitmask.

The execution flow for a layer of LSTMs is depicted in the pseudocode Algorithm 2. The accelerator computes all the neurons of the input (i_t) and forget (f_t) gates. After each activation of the input gate, a comparison with the low threshold is performed to generate a bitmask that determines which neurons from the generate gate are executed. The generate gate and the cell state are computed by checking the bitmask. If an activation of the input gate is lower or equal than the low threshold for a given neuron, the computations of its peer neuron from the generate gate

Algorithm 2 Accelerator LSTM Execution

```
Low Threshold (t_l) = Constant
for Timestep (t) do
  Compute Input Gate (i_t);
  Compute Forget Gate (f_t);
  fAccelerator Executes Multiple Neurons in Parallelg
  for Neuron (n) do
    if i_tn  t_l then
      fGenerate Gate (g_tn) Computations Avoidedg
      Cell State (c_tn) = f_tn c_tn 1;
    else
      Compute Generate Gate (g_tn);
      Compute Cell State (c_tn);
    end if
  end for
  for Neuron (n) do
    c_tn = abs(tanh(c_tn));
    if c_tn  t_l then
      fOutput Gate (o_tn) Computations Avoidedg
      Cell Output (h_tn) = 0;
    else
      Compute Output Gate (o_tn);
      Compute Cell Output (h_tn);
    end if
  end for
end for
```

are avoided and the cell state is computed as the product of the forget gate by the previous cell state. The activations of the cell state are compared against the low threshold to generate a new bitmask that determines which neurons from the output gate (o_t) are executed. If an activation of the cell state in absolute value is lower or equal than the low threshold for a given neuron, the computations of its peer neuron from the output gate are avoided and the cell output is set to zero. Otherwise, the output gate and the cell output is computed as usually. As in the GRU's execution, the control unit is modified to schedule multiple neurons in the array of PEs after checking the bitmasks.

Note that CGPA does not increase the complexity of the control unit of the accelerator significantly, since we are skipping entire neurons. We only need to schedule the neurons that require computations in the array of PEs while the rest are skipped by checking the bitmask generated from the comparators. Unlike a sparse accelerator that requires multiple indices to access the data and a complex control unit, we just need a constant offset with the size to skip the weights of an entire neuron, so the control unit computes the required addresses as indicated by the bitmask.

4 Experimental Results

This section evaluates the performance and energy consumption of CGPA when implemented on top of the TPU-like accelerator presented in Section 3.

Figure 8 shows the speedups achieved by CGPA. Our technique provides consistent speedups for the four RNNs that range from 1.07x (DS2 Tedlium) to 1.21x (PTB-LM), achieving an average performance improvement of 1.12x. The reduction in execution time is due to avoiding the computation of neurons from gates of the recurrent cells that would be multiplied by an activation value that is close to saturation. Furthermore, the overhead of performing the comparison of the activation values with the thresholds is fairly small, since it is performed per output and not per

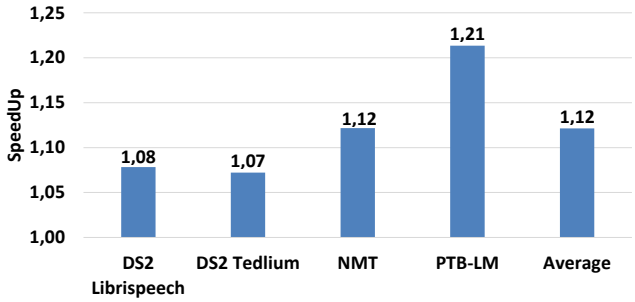


Fig. 8. Speedups achieved for each RNN. Baseline configuration is the TPU-like accelerator without CGPA.

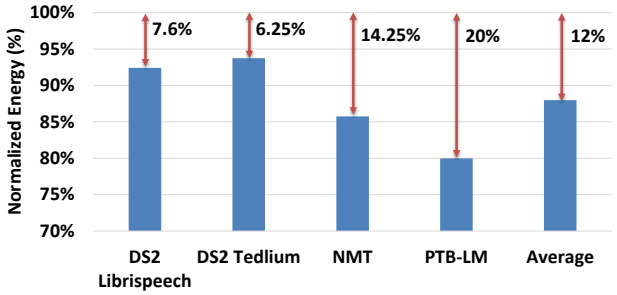


Fig. 9. Normalized energy for each RNN. Baseline configuration is the TPU-like accelerator without CGPA

connection, and not for all the cell gates. Furthermore, comparisons can be done in parallel with useful computations in the systolic array, hiding their latency by a large extent. As one output neuron normally requires hundreds of inputs, performing a comparison to detect whether the output of the neuron will be multiplied by an almost zero value can save hundreds of computations and memory accesses. NMT and PTB-LM exhibit the highest degree of activation values close to saturation (see Section 2) and, hence, they obtain the largest computation reduction and performance improvements.

Figure 9 reports normalized energy. On average, CGPA reduces energy consumption of the accelerator by 12%. The energy overhead of the accelerator is less than 0.001% for all the networks. The most energy consuming component is the on-chip SRAM memory which represents 60% of the overall energy of the accelerator while the array of PEs and the main memory consume 10% and 30% respectively. The energy savings are strongly correlated with the percentage of activation values close to saturation and the computation reduction reported in Figure 6. These energy savings are due to two main reasons. First, dynamic energy is reduced due to the savings in computations and memory accesses. Second, the performance improvements shown in Figure 8 provide a reduction in static energy. Again, NMT and PTB-LM obtain the largest benefits, achieving a reduction of 14% and 20% in energy respectively.

The above evaluation includes all the overheads due to CGPA: accesses to an small SRAM buffer of 512B and floating point comparisons of the activations. As it is shown in Figure 8 and Figure 9, these overheads are negligible in comparison to the savings in computations and memory accesses, and the net result is an improvement of 1.25x in energy-delay (1.12x in energy and 1.12x in delay). The overall area overhead of the accelerator is less than 0.003%, as it increases from 353.93 mm2 to 353.94 mm2.

5 Conclusions

In this paper, we show that modern RNNs exhibit a significant percentage of activation values saturated towards one or zero in different gates of LSTM and GRU cells. We propose CGPA, a technique that exploits these saturated activation values to save computations and memory accesses. As the output of neurons from different gates are multiplied element-wise, CGPA skips the evaluation of a neuron whenever the output of its peer neuron from a different gate is saturated. We implement CGPA on top of a TPU-like accelerator. Our experimental results show that, on average, our scheme provides 12% energy savings and 1.12x speedup, while it only requires a minor increase in the area of the accelerator (less than 0.003%). CGPA provides benefits in performance and energy consumption for RNNs from different applications, including speech recognition, machine translation and language modeling.

Acknowledgments

This work has been supported by the the CoCoUnit ERC Advanced Grant of the EUs Horizon 2020 program (grant No 833057), the Spanish State Research Agency under grant TIN2016-75344-R (AEI/FEDER, EU), and the Spanish Ministry of Education under grant FPU15/02294.

References

[1] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernndez-Lobato, G. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *ISCA*, 2016.

[2] A. Ardakani, C. Condo, and W. J. Gross, "Activation pruning of deep convolutional neural networks," in *GlobalSIP*, 2017.

[3] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *ISCA*, 2016.

[4] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *ISCA*, 2017.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv*, 2016.

[7] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, J. Chen, J. Chen, Z. Chen, M. Chrzanowski, A. Coates, G. Diamos, K. Ding, N. Du, E. Elsen, J. Engel, W. Fang, L. Fan, C. Fougner, L. Gao, C. Gong, A. Hannun, T. Han, L. Johannes, B. Jiang, C. Ju, B. Jun, P. LeGresley, L. Lin, J. Liu, Y. Liu, W. Li, X. Li, D. Ma, S. Narang, A. Ng, S. Ozair, Y. Peng, R. Prenger, S. Qian, Z. Quan, J. Raiman, V. Rao, S. Satheesh, D. Seetapun, S. Sengupta, K. Srinet, A. Sriram, H. Tang, L. Tang, C. Wang, J. Wang, K. Wang, Y. Wang, Z. Wang, Z. Wang, S. Wu, L. Wei, B. Xiao, W. Xie, Y. Xie, D. Yogatama, B. Yuan, J. Zhan, and Z. Zhu, "Deep speech 2 : End-to-end speech recognition in english and mandarin," in *ICML*, 2016.

[8] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv*, 2014.

[9] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, 2014.

[10] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," in *ACM/IEEE ISCA*, 2017.