INT8 Transformers for Inference Acceleration

Andy Rock
Untether AI
andrew@untether.ai

Omar Khalil
Untether AI
omark@untether.ai

Ofer Shai
Untether AI
ofers@untether.ai

Paul Grouchy
MDA*
paul.grouchy@mail.utoronto.ca

Abstract

Given the general trend towards large models in the deep learning community (particularly in the space of Transformers), much work has been done with the goal of reducing the cost associated with inference. In this work, we reach a new low, quantizing all weights and activations of BERT to 8-bit integers. GELU and $\exp$ are implemented with integer lookup tables, achieving optimal INT8 quantization error. We introduce a generalized technique to compute operations frequently missing on integer-only hardware (e.g. divisions, roots) via an efficient instantiation of binary search. By applying it to intermediate computations in Softmax and LayerNorm, we obtain accurate implementations of these layers as well. We evaluate our approach on several GLUE tasks, demonstrating minimal accuracy degradation.

1 Introduction

Over the last several years, Transformers have revolutionized the field of deep learning, successfully replacing recurrent models as the de facto standard in NLP. There are numerous reasons for this - chief among them is the ability of transformer models to scale with a large number of parameters. For this reason, efficient storage and execution of such models is required for practical applications.

One approach that has proved fruitful is quantization, a technique to reduce the bit-widths of neural network activations and weights, saving memory and accelerating computation. There have been several attempts to quantize transformer-based architectures, with varying goals. [7], [15] and [3] focused on quantizing matrix multiplication layers. Mixed ([11]) and sub 8-bit ([2], [16], [6]) quantization schemes have also been attempted.

However, all of the aforementioned papers leave some operations in floating point arithmetic. [3] and [15] keep nonlinearities in FP32, citing the risk of accuracy degradation. [9] replace nonlinearities with integer friendly variants, e.g. $\exp$ with ReLU. I-BERT [8] was the first fully integer quantization of a transformer, with GELU, Softmax and LayerNorm operating directly on the INT32 outputs of previous layers.

To our knowledge, we are the first to propose a quantization of BERT ([5]) which assumes INT8 inputs and outputs for all layers - including LayerNorm, SoftMax and GELU, while suffering minimal accuracy degradation. Our contributions are as follows:

In (2), we describe a generalized SIMD-friendly approach for computing functions such as integer divisions, roots, and beyond, using a specialized form of branchless binary search. This enables the computation of complex functions without requiring intrinsics. In (3), we quantize GELU to INT8

*Work done while at Untether AI

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

via a lookup table, achieving zero accuracy degradation in excess of the inevitable quantization error. In (4), we quantize Softmax to INT8 via two components - a LUT for the e^x step, and an application of our binary search technique for the normalization step. In (5), we quantize LayerNorm to INT8 by applying our binary search technique to precisely compute the inverse of the standard deviation.

In (6), we quantize instances of BERT with our INT8 kernels, demonstrating minimal accuracy degradation on a variety of GLUE tasks.

2 Binary searching for complex operations

2.1 Background

Integer-only inference accelerators often possess a very limited instruction set. This is unsurprising for two reasons: i) there are many operations which are not naturally defined over the integers: division, roots, transcendental functions, etc., and ii) AI accelerators are often massively parallel, which means that control flow instructions (loops, conditionals, etc.) may not be present. To this end, we describe a technique which enables SIMD computation of complex operations, using only existing (generally simpler) intrinsics. Suppose we have some function f, with arguments $\vec{x}$. The result $f(\vec{x})$ is known to belong to $I_k = \{0, 1, \ldots, 2^k - 1\}$. We attempt to apply the following procedure:

2.2 Method

1. Define the function $g(y) = H(y - f(\vec{x}))$, where $H(z) = \mathbb{1}_{z \leq 0}$. Observe that g is monotone, and serves as an evaluation function for a "guess" y to $f(\vec{x})$. In particular, if $y \leq f(\vec{x})$, then $g(y) = 1$. Otherwise, $y > f(\vec{x})$, so $g(y) = 0$. Note that since f is inevaluable, so is g.
2. Attempt to reparameterize g into a form g' that does not contain the parts of f that make it inevaluable. Note that this process is not well-defined, since it depends on the specific set of operations that are supported by the relevant machine. Nonetheless, it is generally quite intuitive, and is best demonstrated by the examples in Appendix A. If this process succeeds, we end up with a function $g' = g$ with the helpful property that (unlike g) g' is evaluable.
3. Define the function $f'(\vec{x}) = \max\{y \in I_k \mid g'(y) = 1\}$. Recalling that $f(\vec{x}) \in I_k$, we have:

$$f'(\vec{x}) = \max\{y \in I_k \mid g'(y) = 1\} = \max\{y \in I_k \mid y \leq f(\vec{x})\} = f(\vec{x}) \tag{1}$$

2.3 Implementation

This alternate definition for f directly yields a straightforward algorithm: iterate in reverse (at compile-time) over I_k, successively querying g' until an index y is found satisfying $g'(y) = 1$. By the definition of f', this is $f(\vec{x})$.

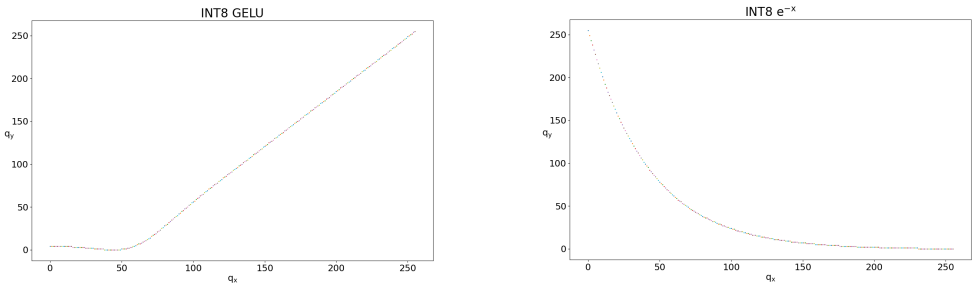
However, this can be quite slow, running in $O(2^k)$ time (we assume g' can be evaluated in $O(1)$ time). Since $g' = g$ is monotone, we can instead proceed via binary search, seeking the maximal element in I_k satisfying g'. Note that a binary search over the range $[0, 2^k)$ takes on a very specific form - it runs for exactly k iterations, and on each iteration determines whether to toggle a bit, from high to low. This runs in $O(k)$ time, and can be expressed branchlessly. It is given in Appendix B and enables the computation of $f(\vec{x})$ - using only a few simple operations and the instructions required to compute g'. Appendix C gives the instantiations of the algorithm for the two examples described previously.

2.4 A note on utility

While this technique may be applicable to many functions, its utility is dependent on hardware. In particular, it is *not* intended to provide a speed-up over intrinsics - if they exist, they should be used. The benefit is only realized when none exist, in which case this provides a reasonable alternative.

2.5 A note on novelty

This method of binary searching via bit twiddling is somewhat of a "folklore" algorithm - vaguely well known, but difficult to track down the origins of (see [4] and [10] for similar ideas). The specific



(a) GELU (b) e^{-x}

Figure 1: LUTs corresponding to optimal INT8 quantization of GELU and e^{-x}

instantiations are also not new - our integer division is isomorphic to restoring division [12] and our integer square root is isomorphic to a technique whose lineage can be traced back thousands of years to the Chinese abacus [14].

Our contribution here is thus the mathematical formalization laid out in Section 2.2. It is this which enables these algorithms to fall out "for free", as well as arbitrarily more complicated ones.

3 GELU

The GELU activation function is used in many transformer models, including BERT. In our setting, we assume asymmetric and per-tensor quantization to the set of 8-bit unsigned integers $I_8 = \{0, 1, \ldots, 255\}$. That is, we have input quantization parameters (S_z, Z_z) and output quantization parameters (S_y, Z_y) such that an input $q_x \in I_k$ corresponds to the real value $r_x = \text{dequant}_x(q_x) = S_x(q_x - Z_x)$, and an output $q_y \in I_k$ corresponds to the real value $r_y = \text{dequant}_y(q_y) = S_y(q_y - Z_y)$.

We would like to compute q_y such that $\text{dequant}_y(q_y)$ is as close as possible to $\text{GELU}(\text{dequant}_x(q_x))$. With the standard quantization function $\text{quant}_y(r_y) = \text{clip}(\lfloor r_y/S_y + Z_y \rceil, 0, 255)$, this optimal quantization is given by the "fake-quant" implementation $q_y = \text{quant}_y(\text{GELU}(\text{dequant}_x(q_x)))$. This leads to the following straightforward algorithm for perfect INT8 quantization of GELU:

1. Precompute $T \in {I_8}^{|I_8|}$ defined by $T_i = \text{quant}_y(\text{GELU}(\text{dequant}_x(i))$ for $i \in I_8$ (Figure 1a).
2. At run-time, for an input $q_x \in I_8$, simply lookup the index q_x in T, returning T_{q_x}.

4 Softmax

The softmax activation function is defined by $\text{softmax}(\vec{x})_i = \frac{e^{x_i}}{\Sigma_j e^{x_j}}$. This can be viewed as two separate steps: 1) compute $\vec{y} = e^{\vec{x}}$, 2) normalize $\vec{y}$ to sum to 1. We quantize each step separately:

4.1 Exponential function

For the purpose of numerical stability, we apply the standard trick of replacing $e^{\vec{x}}$ with $e^{\vec{x} - \max(\vec{x})} = e^{-(\max(\vec{x}) - \vec{x})}$. The subsequent normalization step ensures that this is mathemetically equivalent.

As was done for GELU, we optimally quantize e^{-x} with a static fake-quant lookup table (Figure 1b). This yields a quantized representation $\vec{q_y}$ for $\vec{y}$ with associated quantization parameters (S_y, Z_y).

4.2 Normalization

We want to compute $\vec{q_z}$ with quantization parameters (S_z, Z_z) corresponding to $\vec{z}$, the normalization of $\vec{y}$. Noting that $\vec{y}, \vec{z} \geq \vec{0}$, we take $Z_y = Z_z = 0$. Since $\vec{0} \leq \vec{z} \leq \vec{1}$, we take $S_z = \frac{1-0}{2^8-1-0} = \frac{1}{255}$.

Then optimal quantization is given by $\vec{q_z} = \text{quant}_z\left(\frac{\text{dequant}_y(\vec{y})}{\Sigma\text{dequant}_y(\vec{y})}\right) = \lfloor \vec{q_y} \frac{255}{D} \rceil$, where $D = \Sigma \vec{q_y}$.

	Baseline	FQ (GELU, Softmax, LayerNorm)	INT8
mnli	0.87	0.87	0.87
mrpc(f1)	0.94	0.92	0.92
qnli	0.93	0.92	0.91
sst-2	0.94	0.93	0.93
sts-b(spearman)	0.89	0.91	0.90
rte	0.79	0.78	0.77

Table 1: GLUE inference degradation comparison - all results obtained with post-training quantization

We can easily compute $\lfloor \vec{q_y} \frac{255}{D} \rfloor$ using our bitwise binary search. The full algorithm for INT8 softmax is given in Appendix D.

5 LayerNorm

Layer Normalization operates on an input vector $\vec{x}$ and produces $\vec{y} = \frac{\vec{x} - \mathrm{E}[\vec{x}]}{\sqrt{\mathrm{Var}[\vec{x}]}}\gamma + \beta$.

As usual, we start with the equation for optimal quantization, and solve for $\vec{q_y}$. We have:

$$S_y(\vec{q_y} - Z_y) = \frac{S_x(\vec{q_x} - Z_x) - \mathrm{E}[S_x(\vec{q_x} - Z_x)]}{\sqrt{\mathrm{Var}[S_x(\vec{q_x} - Z_x)]}}\gamma + \beta = \frac{S_x(\vec{q_x} - Z_x) - S_x(\mathrm{E}[\vec{q_x}] - Z_x)}{\sqrt{S_x^2 \mathrm{Var}[\vec{q_x}]}}\gamma + \beta \tag{2}$$

$$\implies \vec{q_y} = \frac{\vec{q_x} - \mathrm{E}[\vec{q_x}]}{\sqrt{\mathrm{Var}[\vec{q_x}]}}\gamma' + \beta' \qquad (\text{with } \gamma' = \tfrac{\gamma}{S_y} \text{ and } \beta' = \tfrac{\beta}{S_y} + Z_y)$$

Hence, quantized LayerNorm can be computed identically to float LayerNorm.

We compute $S_1 = \sum(\vec{q_x})$, $S_2 = \sum(\vec{q_x}^2)$, followed by $E = \frac{S_1}{N} = \mathrm{E}[\vec{q_x}]$, $V = \frac{N * S_2 - S_1^2}{N^2} = \mathrm{Var}[\vec{q_x}]$. Note that N and N^2 are constants, and thus each division can be implemented as a multiplication followed by a shift ([1]).

For the term $\frac{\vec{q_x} - E}{\sqrt{V}}$, we proceed via binary search, applying the method laid out in Section 2.2:

$$\begin{aligned} g(y) = 1 &\iff H\left(y - \frac{\vec{q_x} - E}{\sqrt{V}}\right) = 1 \\ &\iff y \le \frac{\vec{q_x} - E}{\sqrt{V}} \iff y^2 V \le \vec{q_x} - E \\ &\iff H(y^2 V - (\vec{q_x} - E)) = 1 \end{aligned} \tag{3}$$

The subsequent multiplication by γ' and addition of β' are quantized to fixed point. The full algorithm for INT8 LayerNorm is given in Appendix E.

6 Results

Table 1 contains experimental results across several tasks from the GLUE benchmark ([13], CC BY 4.0). The baseline column contains the original scores for the FP32 models. The FQ column refers to an experiment where the three layers of interest were implemented "optimistically", with fake-quantization. The relatively small deviation from the baseline scores indicates that INT8 is sufficient, provided good integer algorithms. The INT8 column contains results from our full quantization of BERT. The relatively small deviation from the FQ scores indicates that our INT8 implementations of GELU, Softmax and LayerNorm are precise, nearly matching fake-quantization.

7 Conclusion

In this work, we have introduced a novel INT8 quantization scheme for several common transformer layers - GELU, Softmax and LayerNorm. This was made possible by two ideas: 1) the optimal

quantization of point-wise functions via static LUTs, and 2) a formulation of binary search that enables efficient synthesis of complex operations from simple intrinsics. We produced a fully INT8 BERT model, with minimal accuracy degradation across a variety of tasks.

References

[1] R. Alverson. Integer division using reciprocals. In *[1991] Proceedings 10th IEEE Symposium on Computer Arithmetic*, pages 186–190, 1991.

[2] H. Bai, W. Zhang, L. Hou, L. Shang, J. Jin, X. Jiang, Q. Liu, M. R. Lyu, and I. King. Binarybert: Pushing the limit of BERT quantization. *CoRR*, abs/2012.15701, 2020.

[3] A. Bhandare, V. Sripathi, D. Karkada, V. Menon, S. Choi, K. Datta, and V. Saletore. Efficient 8-bit quantization of transformer neural machine language translation model. *CoRR*, abs/1906.00532, 2019.

[4] F. Cannizzo. Fast and vectorizable alternatives to binary search. *SSRN Electronic Journal*, 06 2015.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. cite arxiv:1810.04805Comment: 13 pages.

[6] J. Jin, C. Liang, T. Wu, L. Zou, and Z. Gan. KDLSQ-BERT: A quantized bert combining knowledge distillation with learned step size quantization. *CoRR*, abs/2101.05938, 2021.

[7] M. Junczys-Dowmunt, K. Heafield, H. Hoang, R. Grundkiewicz, and A. Aue. Marian: Cost-effective high-quality neural machine translation in C++. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 129–135, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[8] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer. I-bert: Integer-only bert quantization. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5506–5518. PMLR, 18–24 Jul 2021.

[9] Y. Lin, Y. Li, T. Liu, T. Xiao, T. Liu, and J. Zhu. Towards fully 8-bit integer inference for the transformer model. In C. Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 3759–3765. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Main track.

[10] B. Mastripolito, N. Koskelo, D. Weatherred, D. A. Pimentel, D. Sheppard, A. P. Graham, L. Monroe, and R. Robey. Simd-optimized search over sorted data. *CoRR*, abs/2112.03229, 2021.

[11] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer. Q-bert: Hessian based ultra low precision quantization of bert. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8815–8821, Apr. 2020.

[12] K. D. TOCHER. TECHNIQUES OF MULTIPLICATION AND DIVISION FOR AUTOMATIC BINARY COMPUTERS. *The Quarterly Journal of Mechanics and Applied Mathematics*, 11(3):364–384, 01 1958.

[13] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. 2019. In the Proceedings of ICLR.

[14] C. C. Woo. *The Fundamental Operations in Bead Arithmetic. How to Use the Chinese Abacus.* 1922.

[15] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat. Q8BERT: quantized 8bit BERT. *CoRR*, abs/1910.06188, 2019.

[16] D. Zhang, J. Yang, D. Ye, and G. Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. *CoRR*, abs/1807.10029, 2018.

Appendix

A Reparameterization Examples

- Let $f(n, d) = \lfloor \frac{n}{d} \rfloor$, where $n \geq 0$ and $d > 0$ are integers. Suppose division is unsupported, but subtraction and multiplication are. For $y \in I_k$, we have:

$$\begin{aligned} g(y) = 1 &\iff H(y - f(n,d)) = 1 \\ &\iff y \leq \left\lfloor \frac{n}{d} \right\rfloor \iff y \leq \frac{n}{d} \iff dy \leq n \\ &\iff H(dy - n) = 1 \end{aligned} \tag{4}$$

  Thus, by letting $g'(y) = H(dy - n)$, we can compute $g(y)$ via $g'(y)$.

- Let $f(x) = \lfloor \sqrt{x} \rfloor$, where $x \geq 0$ is an integer. Suppose $\sqrt{}$ is unsupported, but subtraction and multiplication are. For $y \in I_k$, we have:

$$\begin{aligned} g(y) = 1 &\iff H(y - f(x)) = 1 \\ &\iff y \leq \lfloor \sqrt{x} \rfloor \iff y \leq \sqrt{x} \iff y^2 \leq x \\ &\iff H(y^2 - x) = 1 \end{aligned} \tag{5}$$

  Thus, by letting $g'(y) = H(y^2 - x)$, we can compute $g(y)$ via $g'(y)$.

B Bitwise Binary Search

```
1:  function BITWISE_BINARY_SEARCH(g', k)
2:      y ← 0
3:      for b = k − 1 to 0 do
4:          y' ← y | (1 ≪ b)
5:          if g'(y') then
6:              y ← y'
7:          end if
8:      end for
9:      return y
10: end function
```

C Bitwise Binary Search Application

Instantiations of the technique described in Section 2.2 for truncated integer division and square root.

```
1:  function I_DIVIDE(n, d, k)                          ▷ computes ⌊n/d⌋ ∈ I_k)
2:      function CONDITION(y)
3:          return d * y ≤ n
4:      end function
5:      return BITWISE_BINARY_SEARCH(CONDITION, k)
6:  end function
                                                        ▷ I_DIVIDE(76, 7, 4) → 10
7:  function I_SQRT(x, k)                               ▷ computes ⌊√x⌋ ∈ I_k)
8:      function CONDITION(y)
9:          return y * y ≤ x
10:     end function
11:     return BITWISE_BINARY_SEARCH(CONDITION, k)
12: end function
                                                        ▷ I_SQRT(2022, 8) → 44
```

D Softmax

function SOFTMAX($\vec{x}$)
    $\vec{y} \leftarrow \text{LUT}[\max(\vec{x}) - \vec{x}]$
    $D \leftarrow \sum(\vec{y})$
    function CONDITION($\vec{z}$)
        return $\vec{z} * D \leq \vec{y} * 255$
    end function
    return BITWISE_BINARY_SEARCH(CONDITION, 8)
end function

E LayerNorm

function LAYERNORM($\vec{x} \in {I_8}^N$)
    $S_1 \leftarrow \sum(\vec{x})$
    $S_2 \leftarrow \sum(\vec{x}^2)$
    $E \leftarrow \lfloor \frac{S_1}{N} \rceil$
    $V \leftarrow \lfloor \frac{NS_2 - {S_1}^2}{N^2} \rceil$
    function CONDITION($\vec{y}$)
        return $\vec{y} * \vec{y} * V \leq \vec{x} - E$
    end function
    $\vec{y} \leftarrow$ BRANCHLESS_BINARY_SEARCH(CONDITION, 8)
    return $\vec{y} * q_{\gamma'} + q_{\beta'}$
end function