Towards Fully 8-bit Integer Inference for the Transformer Model

Ye Lin[1*], **Yanyang Li**[1*], **Tengbo Liu**[1], **Tong Xiao**[1,2†], **Tongran Liu**[3] and **Jingbo Zhu**[1,2]
[1]Natural Language Processing Lab., Northeastern University, Shenyang, China
[2]NiuTrans Research, Shenyang, China
[3]CAS Key Laboratory of Behavioral Science, Institute of Psychology, CAS, Beijing, China
{linye2015, blamedrlee}@outlook.com, tengboliu@stumail.neu.edu.cn,
{xiaotong, zhujingbo}@mail.neu.edu.cn, liutr@psych.ac.cn

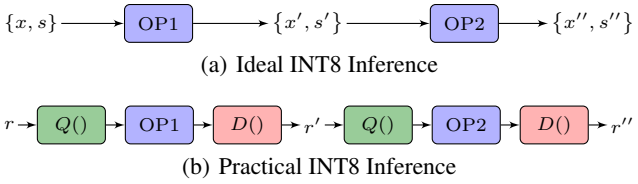
Abstract

8-bit integer inference, as a promising direction in reducing both the latency and storage of deep neural networks, has made great progress recently. On the other hand, previous systems still rely on 32-bit floating point for certain functions in complex models (e.g., Softmax in Transformer), and make heavy use of quantization and de-quantization. In this work, we show that after a principled modification on the Transformer architecture, dubbed *Integer Transformer*, an (almost) fully 8-bit integer inference algorithm *Scale Propagation* could be derived. De-quantization is adopted when necessary, which makes the network more efficient. Our experiments on WMT16 En↔Ro, WMT14 En↔De and En→Fr translation tasks as well as the WikiText-103 language modelling task show that the fully 8-bit Transformer system achieves comparable performance with the floating point baseline but requires nearly 4× less memory footprint.

1 Introduction

In recent years, the self-attention-based Transformer model [Vaswani *et al.*, 2017] has shown promising improvements in a wide variety of tasks, e.g., machine translation [Li *et al.*, 2020] and language modelling [Baevski and Auli, 2019]. The superior performance of these systems is mostly achieved by using very large neural networks, which are accompanied by the great demands on computation, storage and energy [Strubell *et al.*, 2019]. As a side effect, deploying such models on small devices is challenging as they have limited storage space and computation power. For example, practical systems often run on CPUs where the 32-bit floating point computation capability is much lower than that of GPUs.

One appealing solution to these issues is to reduce the numerical precision used in the model at hand [Hubara *et al.*, 2016; Micikevicius *et al.*, 2018], where both the parameters and the activations are represented with fewer bits. For instance, employing 8-bit integer (INT8) potentially consumes 4× less storage space but is up to 6× faster [Quinn and Ballesteros, 2018]. Beyond this, INT8 is 10× more energy efficient [Johnson, 2018] and saves much less chip area than the commonly used 32-bit floating point (FP32) in hardware design [Sze *et al.*, 2017]. Also, the low-precision approach is orthogonal to other existing compression and acceleration methods, e.g., efficient network design [Xiao *et al.*, 2019].



(a) Ideal INT8 Inference
(b) Practical INT8 Inference

Figure 1: Ideal vs. Practical INT8 inference (OP: operation).

In general, we need two additional components to adapt FP32 algorithms to INT8 algorithms: quantization and de-quantization [Gong *et al.*, 2018]. Quantization can be seen as a function that transforms a rational tensor r into an integer tensor x with the scale s [Wu, 2020]:

$$Q(r, s) = \lfloor s \cdot r \rceil \tag{1}$$

where $\lfloor \cdot \rceil$ represents rounding to the nearest integer. As a reverse process, de-quantization approximates the rational tensor r with its quantized form x:

$$D(x, s) = x/s \tag{2}$$

Ideally, the INT8-based inference process is as follow: the rational input (FP32) tensor r is first quantized to an INT8 tensor x with the scale s. Then all succeeding operations are performed on INT8 tensors and corresponding scales simultaneously. De-quantization is employed at the end of the process or there appears an overflow[1].

This method is efficient because quantization and de-quantization functions are used only when necessary. Unfortunately, previous INT8-based models are much more expensive, as every operation in it is sandwiched between a pair of quantization and de-quantization (see Fig. 1). The heavy use of quantization and de-quantization blocks the efficient flow

[*]Authors contributed equally.
[†]Corresponding author.

[1]For intermediate tensors produced by these operations, we perform de-quantization and quantization with $s = \frac{2^p - 1}{\max(|r|)}$ immediately if the overflow happens. The bit-precision p is 7 for INT8.

of INT8 throughout the network, and somehow prevents fully 8-bit integer models. The problem lies in two facts:

- Scale Incompatibility: INT8 tensors with different scales are incomparable because we cannot use the same FP32-to-INT8 mapping to process them in a single operation. For example, let x_1 and x_2 be INT8 tensors that are quantized from FP32 tensors r_1 and r_2 with difference scales s_1 and s_2. Adding x_1 and x_2 is obviously problematic because $x_1 + x_2$ is not the INT8 form of $r_1 + r_2$, i.e., $r_1 + r_2 \neq (x_1 + x_2)/s_1 \neq (x_1 + x_2)/s_2$.
- INT8 Incompatibility: some functions in complex networks are not INT8 friendly and we have to resort to FP32 computation in this case. The most representative examples are the exponential function in the attention mechanism and the square root function in layer normalization [Vaswani *et al.*, 2017].

In this work, we take a further step towards fully INT8-based transformer models. We choose Transformer for study because it is one of the most popular models in natural language processing. We present *Scale Propagation*, which bounds INT8 tensors with associated scales, and propagates them throughout the network during inference. It addresses the scale incompatibility issue by matching the input scales if necessary, allowing each operation to manipulate the INT8 tensor and its scale simultaneously. Moreover, we propose *Integer Transformer* in responding to the INT8 incompatibility issue. To make full use of INT8 in Transformer, we replace the exponential function in the standard attention by the polynomial function, and replace the square root function in the layer normalization with the absolute value function. Our extensive experiments on several machine translation and language modelling tasks show that integer Transformer achieves competitive INT8 performance with approximately 4× less storage and 3.47× speed-up on average.

2 Background: Transformer

We start with the description of Transformer. Transformer [Vaswani *et al.*, 2017] is mainly composed of a stack of layers. Each layer consists of a self-attention and a feed-forward network. The self-attention takes three tensors, Q, K and V, as inputs and produces a tensor with the same size as the output. It is formulated as:

$$\mathrm{Attention}(Q, K, V) = \mathrm{SoftMax}(\frac{QK^T}{\sqrt{d_\mathrm{m}}})V \tag{3}$$

where d_m is the dimension of the hidden representation. $\mathrm{SoftMax}$ is a function that casts its input to a distribution:

$$\mathrm{SoftMax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{4}$$

The feed-forward network is built on top of two linear projections with the ReLU activation in between:

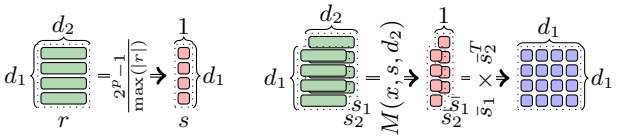
$$\mathrm{FFN}(x) = \mathrm{ReLU}(xW_1 + b_1)W_2 + b_2 \tag{5}$$

$$\mathrm{ReLU}(x) = \max(0, x) \tag{6}$$

These modules are coupled with the residual connection [He *et al.*, 2016], i.e., $y = f(x) + x$ where f is either the self-attention or the feed-forward network. The Layer Normalization is after the residual connection:

$$\mathrm{LN}(x) = g \odot (\frac{x - \mu}{\sqrt{\sigma^2 + \varepsilon}}) + b \tag{7}$$

where μ and σ are the mean and variance of x along the hidden dimension, and ε is a fixed small number to prevent dividing 0. g and b are two learnable parameters. For more details, we refer the reader to [Vaswani *et al.*, 2017].



(a) Scale Initialization (b) Scale MatMul (along d_2)

Figure 2: Examples of initializing scale and multiplying scale in MatMul.

3 Scale Propagation

3.1 Bounding Tensors & Scales

As discussed in Section 1, the necessity of de-quantization comes from the fact that input INT8 tensors might not be produced by the same mapping that converts FP32 to INT8, e.g., not multiplied by the same scale in our case, and therefore forces us to compute the correct result by rolling back to the FP32 mode.

Inspired by this fact, the ideal INT8-based inference should propagate not only the tensor but also the mapping. If multiple INT8 tensors are inputted, unifying their mappings is necessary so that we can perform the succeeding operation on INT8 tensors directly. In our case, the mapping is defined as the scale that indicates to what extent the current INT8 tensor deviates from its FP32 counterpart. This scale can be obtained through $s = \frac{2^p-1}{\max(|r|)}$ initially, where r is the network input. In practice, $\max(|r|)$ is performed along the hidden dimension, producing a scale with the same shape as r except the dimension that the maximization operates is 1. Fig. 2(a) shows how to initialize the scale from an FP32 tensor. Although we introduce extra operations to manipulate FP32 scales, it is cheap to maintain them and the cost is negligible.

3.2 Manipulating Tensors & Scales

Extending the common FP32 operations to INT8 tensors and associated scales is non-trivial. Two questions naturally arise: 1) how can we calibrate mappings? 2) how to operate both the INT8 tensors and scales for a given FP32 operation?

For the first question, we note that the mapping here is a scale that gets multiplied in the quantization. Thus having an identical mapping across tensors is as to find a unique multiplier for every input tensor in our case. For each input tensor x_i with the scale s_i, we do *Scale Matching*:

$$M(x_i, s_i) = \{x_i / \lceil s_i/\bar{s} \rceil, \bar{s}\} \tag{8}$$

where $\bar{s} = \min(s_1, \cdots, s_n)$. Choosing the minimum of scales $\bar{s}$ as the unique multiplier guarantees that the result of Eq. 8 does not overflow.

FP32 OP	INT8 Equivalent
$[r_1, r_2]$	$\{[x_1, x_2], [s_1, s_2]\}$
r_1^T	$\{x_1^T, s_1^T\}$
$r_1 \cdot r_2$	$\{x_1 \cdot x_2, s_1 \cdot s_2\}$
$r_1 + r_2$	$\{x_i, \bar{s}\} = M(x_i, s_i), i \in \{1, 2\}$ $\{x_1 + x_2, \bar{s}\}, \bar{s} \in \mathbb{R}^{m \times n}$
$\text{MatMul}(r_1, r_2^T)$	$\{x_i, \bar{s}_i\} = M(x_i, s_i, d_2), i \in \{1, 2\}$ $\{x_1 \times x_2^T, \bar{s}_1 \times \bar{s}_2^T\}, \bar{s}_i \in \mathbb{R}^{m \times 1}$
r_1^n	$\{x_1^n, s_1^n\}$
$\lvert r_1 \rvert$	$\{\lvert x_1 \rvert, s_1\}$
$\text{ReLU}(r_1)$	$\{\text{ReLU}(x_1), s_1\}$

Table 1: FP32 operations in INT8. $r_i = D(x_i, s_i)$, $i \in \{1, 2\}$, $r_i \in \mathbb{R}^{m \times n}$, $x_i \in \mathbb{Z}^{m \times n}$, $s_i \in \mathbb{R}^{m \times n}$. [] denotes the concatenation. $\cdot$ denotes the element-wise multiplication.

Algorithm 1 SCALE PROPAGATION PROTOCOL

Input: Operation OP; INT8 Tensors $x_{1...n}$; Scales $s_{1...n}$
Output: INT8 Tensor x; Scale s

1: $\{x, s\} = \text{OP}(\{x_{1...n}, s_{1...n}\})$ {Store x in INT32}
2: **if** $x > 2^p - 1$ **then**
3: $\quad \{x, s\} = R(x, s)$ {Re-scaling}
4: **end if**
5: Convert (INT32) x to INT8
6: **return** x, s

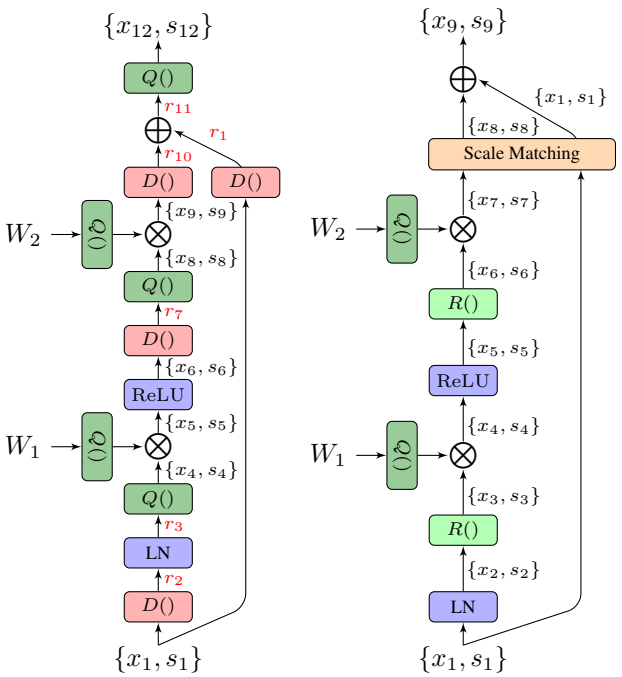
Having the scale matching, it is handy to induce the INT8 form for any FP32 tensor operation, as shown in Table 1. For tensor shape transformations, such as concatenation and transpose, the same transformation is applied to the INT8 tensor and its scale simultaneously, since they do not change the values. For element-wise multiplication, we multiply tensors and scales independently, as the quantization is just another element-wise multiplication. For addition, we first match the input scales via Eq. 8, then add tensors as usual.

Handling matrix multiplication (MatMul) is more sophisticated. MatMul is an element-wise multiplication with an addition along the last dimension. We therefore first match the input scales along that dimension, then perform MatMul to the tensors and scales independently. To match the input scale along a specific dimension, we employ the same idea of scale matching by treating it as matching scales of multiple sub-tensors splitted from that dimension. It is denoted as $M(x, s, d)$, where x is the INT8 tensor, s is its scale and d is the dimension that we would like to match scales. Fig. 2(b) shows an example of how MatMul works on scales, which matches the scales on the dimension d_2 and multiplies them.

For element-wise non-linear functions, we assume that they satisfy the *distribution law*, i.e., $\text{OP}(r) = \text{OP}(x/s) = \text{OP}(x)/\text{OP}(s)$. Then, we have:

$$\text{OP}(\{x, s\}) = \{\text{OP}(x), \text{OP}(s)\} \tag{9}$$

where x is the INT8 tensor and s is its scale. This assumption holds for the polynomial function r^n where n is a fixed integer, since $r^n = (x/s)^n = x^n/s^n$. It also holds for the absolute value function, because $|r| = |x/s| = |x|/s$ as $s > 0$.



(a) Canonical Inference (b) Scale Propagation

Figure 3: The comparison of INT8 inferences in the FFN layer.

The same is for $\text{ReLU}(\{x, s\})$ when entries of x have the maximum value(s) exceeded 0, which is always true otherwise it will face the 'dying ReLU' problem [He *et al.*, 2015].

3.3 The General Protocol

Note that addition and multiplication operations may produce results that are out of the INT8 range. These results are thereby stored in data types with more bits in practical implementations, e.g., INT32. We need to project the result back to INT8 before the succeeding operations. We call it *Re-scaling*:

$$R(x, s) = \{x/\hat{s}, s/\hat{s}\} \tag{10}$$

where $\hat{s} = \left\lceil \frac{\max(|x|)}{2^p - 1} \right\rceil$. The protocol of extending an FP32 operation to INT8 tensors and their scales is summarized in Alg. 1: we directly apply the INT8 form of this operation to update $\{x, s\}$, and then use re-scaling to project x back to INT8 if necessary. Once this protocol for INT8 operations is defined, the routine for the INT8 forward propagation is as straightforward as the FP32 one, except that FP32 operations are replaced by their INT8 equivalents. This gives us the *Scale Propagation*. As shown in Fig. 3, scale propagation gets rid of de-quantization and only INT8 tensors are propagated in the whole forward propagation.

4 Integer Transformer

4.1 Polynomial Attention

Applying scale propagation to the Transformer model is not immediately available. As discussed in the previous section,

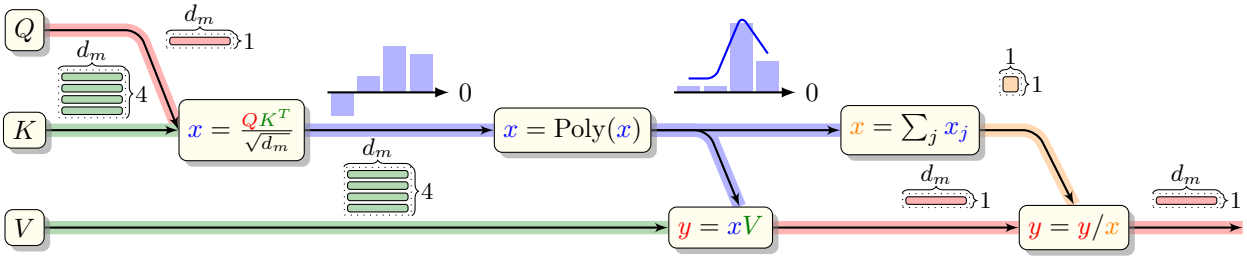


Figure 4: A running example of Polynomial Attention, where $Q \in \mathbb{R}^{1\times d_m}$, $K, V \in \mathbb{R}^{4\times d_m}$. Different colors indicate different shapes.

scale propagation assumes the element-wise functions satisfy the distribution law, which is not held for the exponential function e^x in the SoftMax of the attention functions, as $e^{x/s} \neq e^x/e^s$. Besides, the exponential function does not produce an integer output given an integer input.

To enable scale propagation, we choose ReLU as an alternative of the exponential function here, since it not only produces positive results as the exponential function but also is compatible with INT8. A bias term is added in advance to rule out entries that are below a learnt threshold.

One downside of ReLU is its linear nature. The exponential function has the property that larger input values become more significant after the transformation, as its gradient e^x is larger than 1 in the positive number field. To achieve a similar effect, we introduce the polynomial function x^n, whose gradient $n \cdot x^{n-1}$ is exponential while it always produces integer outputs given integer inputs. Note that we only apply this polynomial function after ReLU, since an even degree n will mess up with the order of scores: a large negative number will be ranked in front instead of behind.

Putting all these pieces together, we have:

$$\text{Poly}(x) = [\text{ReLU}(x+b)]^n + |\delta| \tag{11}$$

where b is the bias term, n is the degree of the polynomial function and δ is another learnable parameter. $|\delta|$ ensures that the worst case of the attention, i.e., producing all 0 results, is a simple average instead of nothing.

Lastly, we multiply $\text{Poly}(x)$ with V and then divide the result by $\sum_j \text{Poly}(x_j)$, otherwise the integer division will incur all 0 results because $\text{Poly}(x_i) \leq \sum_j \text{Poly}(x_j)$:

$$\text{PolyAttn}(Q,K,V) = \frac{\text{Poly}(\frac{QK^T}{\sqrt{d_m}})V}{\sum_j \text{Poly}(\frac{QK_j^T}{\sqrt{d_m}})} \tag{12}$$

This way sidesteps the previous issue as the multiplication results are usually not smaller than the sum. We call Eq. 12 *Polynomial Attention*. Fig. 4 shows a running example of it.

4.2 L1 Layer Normalization

Another component that hinders Transformer INT8 inference is the square root function for computing the standard deviation inside the layer normalization, which does not guarantee the integer outputs given the integer inputs. Hoffer *et al.* [2018] proposes *L1 Batch Normalization*, which approximates the standard deviation with its L1-norm equivalent:

$$\text{L1LN}(x) = g \odot \left(\frac{x-\mu}{C \cdot \| x-\mu \|_1 / n}\right) + b \tag{13}$$

where g and b are two parameters, μ is the mean of x along the batch dimension, $C = \sqrt{\pi/2}$ and n is the batch size. This way replaces the square root function in the L2-norm by the absolute value function in the L1-norm.

We extend a similar idea of L1 batch normalization to our case, that we compute the mean μ along the hidden dimension instead of the dimension along the batch. We call this *L1 Layer Normalization*. The replacement of layer normalization as well as the attention gives us the *Integer Transformer* that supports fully INT8 inference.

5 Experiments

5.1 Setup

We evaluate our methods on three machine translation (MT) tasks and a language modelling (LM) task, including the WMT16 English-Roman (En↔Ro), the WMT14 English-German (En↔De), the WMT14 English-French (En→Fr) and the WikiText-103 LM tasks. For En↔Ro (610K pairs), we use *newsdev-2016* and *newstest-2016* as the validation and test sets respectively. For En↔De (4.5M pairs), *newstest-2013* is the validation set and *newstest-2014* is the test set. For En→Fr (36M pairs), we validate the system on the combination of *newstest-2012* and *newstest-2013*, and test it on *newstest-2014*. We tokenize every sentence using a script from Moses and segment every word into subword units using byte-pair encoding. The number of the BPE merge operations is set to 32K. We report case-sensitive tokenized BLEU scores. In addition, the results are the average of three identical runs with different random seeds for En↔Ro and En↔De. The WikiText-103 dataset contains a training set of 103 million words. Both the validation and test sets contain 0.2 million words. For the LM task, we report the perplexity.

For the machine translation tasks, we experiment with the Transformer-base (base) setting [Wang *et al.*, 2019]. We additionally run the Transformer-big (big) setting on En↔De and En→Fr. Both settings consist of a 6-layer encoder and a 6-layer decoder. The embedding size is set to 512 for Transformer-base and 1,024 for Transformer-big. The number of heads is 8/16 for Transformer-base/big. The hidden size equals to 4× embedding size in both settings. For training, we use Adam optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.997$. We adopt the inverse square root learning rate schedule with 8K warmup steps and the learning rate $= 0.001/0.0007$ for Transformer-base/big.

For the language modelling task, we follow the lm-base and lm-big architectural choices and training details de-

Entry		System	BLEU		Storage	Estimated Speed-up
			FP32	INT8		
base	En→Ro	Baseline	32.55	-	318M	1×
		Ours	**32.60**	32.54	80M	3.53×
	Ro→En	Baseline	32.85	-	306M	1×
		Ours	**33.04**	32.95	77M	3.59×
	En→De	Baseline	26.95	-	302M	1×
		Ours	**27.08**	26.91	76M	3.24×
	De→En	Baseline	32.19	-	302M	1×
		Ours	**32.43**	32.26	76M	3.31×
	En→Fr	Baseline	**40.88**	-	425M	1×
		Ours	40.64	40.00	107M	3.03×
big	En→De	Baseline	28.72	-	939M	1×
		Ours	**28.93**	28.71	236M	3.60×
	De→En	Baseline	33.07	-	939M	1×
		Ours	**33.53**	33.46	236M	3.68×
	En→Fr	Baseline	42.37	-	1243M	1×
		Ours	**42.46**	41.59	311M	3.51×

Table 2: BLEU scores [%], storage (megabytes) and speed-up.

Entry		valid		test		Storage	Estimated Speed-up
		FP32	INT8	FP32	INT8		
base	Baseline	29.61	-	31.18	-	596M	1×
	Ours	**29.49**	30.28	**30.79**	31.61	150M	3.43×
big	Baseline	18.22	-	18.86	-	944M	1×
	Ours	**17.49**	17.55	**18.16**	18.23	280M	3.78×

Table 3: WikiText-103 PPL, storage (megabytes) and speed-up.

scribed in [Baevski and Auli, 2019]. The embedding size is 512 for lm-base and 1024 for lm-big. The hidden size equals to 4× embedding size. The number of heads is 8 for both lm-base and lm-big. The number of layers is set to 6/16 for lm-base/big. For the lm-base model, we train it with the same setting as in the machine translation tasks. As for the lm-big training, we use the Nesterov's accelerated gradient. We adopt the cosine learning rate schedule with 16K warmup steps and the maximum learning rate 1. All experiments are run on 8 NVIDIA TITAN V GPUs.

5.2 Results

Table 2 summarizes the results on various translation tasks. Compared to the vanilla Transformer, integer Transformer obtains competitive or even superior FP32 performance by 0.1∼0.4 BLEU points in either the base or big setup. When integer Transformer is decoded with INT8, it shows only about a decrease of 0.3 BLEU points on average except in En→Fr, where it underperforms the baseline by more than 1 BLEU point. In Section 5.3, we will show that it is mainly due to the last residual connection and layer normalization, which suffer from greater loss with lower bits representations. Experiments on the WikiText-103 language modelling task in Table 3 show a similar trend as those in machine translation tasks, where integer Transformer beats the baseline with the same setup as in MT.

Both Table 2 and Table 3 show that using INT8 indeed saves nearly 4× storage space. Since we need to store both the parameters and their scales, we are unable to reach exactly 4× less storage. Employing INT8 also runs about 3.5× faster on average. Note that we estimate this speed-up by collecting the time consumption of each operation and their corresponding speed-up (6×) in INT8, as modern CPUs have limited supports of INT8 arithmetics, e.g., MatMul only. We find that this speed-up is more obvious if the output sequence is longer, e.g., translations in Ro→En is longer than those in En→Fr and thus higher speed-up in Ro→En is observed. This phenomenon arises from the fact that operations that benefit from INT8 such as MatMul occupy a higher portion when generating long sequences, while other fixed time operations such as data preparation become marginal.

System	BLEU			PPL	
	En→Ro	En→De	En→Fr	valid	test
Baseline	32.55	26.95	**40.99**	29.58	31.28
+Poly	**32.56**	**27.13**	40.90	**29.54**	31.20
+L1LN	32.55	26.94	40.67	29.61	**31.18**

Table 4: The ablation study of Integer Transformer.

System	BLEU			PPL	
	En→Ro	En→De	En→Fr	valid	test
Ours (FP32)	32.60	27.08	40.99	29.49	30.79
+B Scale	**32.54**	21.84	39.45	30.88	32.25
+$B \times T$ Scale	**32.54**	**26.91**	**40.00**	**30.28**	**31.61**

Table 5: INT8 performance vs. different sized scales.

5.3 Analysis

We show an ablation study of integer Transformer in Table 4. We can see that replacing the standard attention by polynomial attention generally improve the FP32 result and L1 layer normalization has the close performance to standard layer normalization. These observations imply that either the polynomial attention or the L1 layer normalization is a good alternative to its counterpart in the baseline transformer model.

Section 3.1 has described how to obtain the initial scale by taking the maximum of the hidden dimension in the FP32 input. This method can be extended to the case of multiple dimensions. In Table 5, we test it on maximizing on $T \times C$ and C given the input of the size $T \times B \times C$, resulting a sized B and $B \times T$ scale respectively. Here T is the input sequence length, B is the batch size and C is the number of the hidden units. The results reveal that using a scale with more entries better preserves the performance, yet the one with fewer entries lowers the computation budget.

Also, we plot how hyper-parameters relate to performance. We can see from the left of Fig. 5 that $n > 1$ results in much better performance than $n = 1$ in all tasks, indicating the necessity of non-linearity in the attention. But higher n does not necessarily lead to better results, where $n = 3$ performs the best in En→De and WikiText-103. The right of Fig. 5 shows that adding a few bits can recover most of the performance, especially for those suffer from great loss in INT8 inference, e.g., En→Fr. Moreover, we observe that the performance of En→Ro decreases slightly with 6 bit, which suggests that further speed-up might be available.

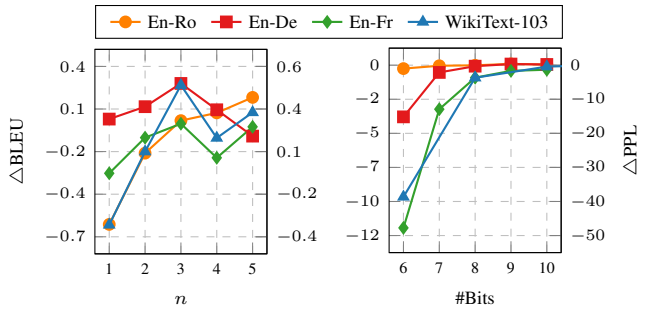


Figure 5: Sensitivity analysis (n: the degree of the polynomial function; #Bits: the number of bits used in the inference).

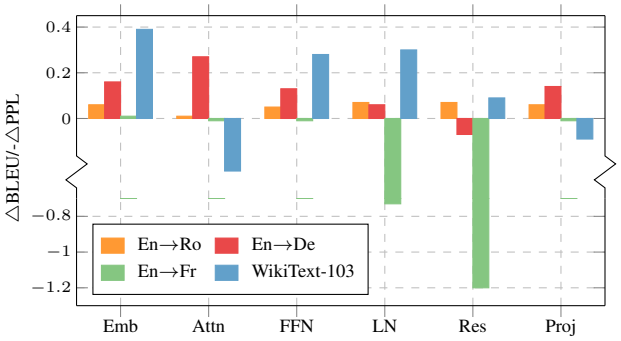


Figure 6: Performance improvement (> 0) and loss (< 0) of applying INT8 to modules (Emb: the embedding; Attn: the attention; FFN: the feed-forward network; LN: the layer normalization; Res: the residual connection; Proj: the output projection.)

We next investigate which factor has a significant impact on the performance by presenting details on which module is responsible for the INT8 performance loss. As can be seen in Fig. 6, if the performance drop is not significant, each module contribute similarly, otherwise a few modules should be blamed for. This fact suggests that poor INT8 performance is mainly led by one or two crucial points, e.g., the layer normalization and the residual connection in En→Fr.

As implied by Fig. 6, we make an in-depth analysis to see whether the high precision loss connects to the poor performance of applying INT8 to the layer normalization and the residual connection in En→Fr. To evaluate the precision loss, we choose the mean square error between the FP32 activations and the de-quantized INT8 ones as the proxy. Fig. 7 shows that there exists a positive relationship between precision loss and performance loss, i.e., △BLEU. Interestingly, most loss occurs in the last layer. Noting that the residual connection is the sum of all outputs of the residual branches in previous layers, the last residual connection will produce the result with large values, which might suffer from greater precision loss through the quantization.

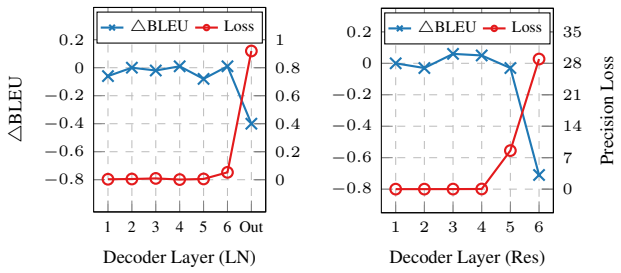


Figure 7: Precision loss vs. Performance loss (En→Fr, Out: the last layer normalization before the output projection).

6 Related Work

Employing the low precision data type for neural networks to accelerate the network computation or save storage space has a long history. Early work has shown that training and inference with the ternary (2-bit) or even binary (1-bit) network is possible [Hubara *et al.*, 2016]. But these results have restricted to simple architectures, such as the feed-forward networks. Recent work mainly focuses on training a sophisticated network with higher precision, such as 32-bit (FP32) and 16-bit floating point (FP16) [Micikevicius *et al.*, 2018] but attempts to inference with fewer bits, such as 8-bit fixed point (INT8) [Jacob *et al.*, 2018]. However, most of them have limited to computer vision and only a few of them discuss how to leverage low precision to infer the complicate Transformer model in natural language processing.

Bhandare *et al.* [2019] first demonstrates that Transformer can be inferred with INT8. But some operations are still performed in FP32 and its INT8 performance is not evaluated by common metrics, e.g., BLEU. Though more recent work [Prato *et al.*, 2019; Wu, 2020] share the same limitation of partially relying on FP32, they report better INT8 results by tailoring the training as well as the quantization method to the Transformer model. This work, on the other hand, takes a step toward fully INT8 inference without any FP32 operation for the Transformer model. The forward propagation flows purely on INT8 and shows competitive performance without modifying the training process.

7 Conclusion

In this work, we present an (almost) fully INT8 inference algorithm *Scale Propagation*, which propagates the INT8 tensor and its scale to resolve the scale incompatibility problem. Moreover, we propose *Integer Transformer* to address the INT8 incompatibility issue in the Transformer model, which replaces the exponential function and the square root function by the polynomial function and the absolute value function respectively. Our experiments show that our method achieves competitive INT8 performance in machine translation and language modelling tasks.

Acknowledgments

This work was supported in part by the National Science Foundation of China (Nos. 61876035 and 61732005), the National Key R&D Program of China (No. 2019QY1801) and the Opening Project of Beijing Key Laboratory of Internet Culture and Digital Dissemination Research. The authors would like to thank anonymous reviewers for their comments.

References

[Baevski and Auli, 2019] Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[Bhandare *et al.*, 2019] Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. Efficient 8-bit quantization of transformer neural machine language translation model. *CoRR*, abs/1906.00532, 2019.

[Gong *et al.*, 2018] Jiong Gong, Haihao Shen, Guoming Zhang, Xiaoli Liu, Shane Li, Ge Jin, Niharika Maheshwari, Evarist Fomenko, and Eden Segal. Highly efficient 8-bit low precision inference of convolutional neural networks with intelcaffe. In *Proceedings of the 1st on Reproducible Quality-Efficient Systems Tournament on Co-designing Pareto-efficient Deep Learning, ReQuEST@ASPLOS 2018, Williamsburg, VA, USA, March 24, 2018*, page 2, 2018.

[He *et al.*, 2015] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034, 2015.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778, 2016.

[Hoffer *et al.*, 2018] Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 2164–2174, 2018.

[Hubara *et al.*, 2016] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4107–4115, 2016.

[Jacob *et al.*, 2018] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 2704–2713, 2018.

[Johnson, 2018] Jeff Johnson. Rethinking floating point for deep learning. *CoRR*, abs/1811.01721, 2018.

[Li *et al.*, 2020] Yanyang Li, Qiang Wang, Tong Xiao, Tongran Liu, and Jingbo Zhu. Neural machine translation with joint representation. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[Micikevicius *et al.*, 2018] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[Prato *et al.*, 2019] Gabriele Prato, Ella Charlaix, and Mehdi Rezagholizadeh. Fully quantized transformer for improved translation. *CoRR*, abs/1910.10485, 2019.

[Quinn and Ballesteros, 2018] Jerry Quinn and Miguel Ballesteros. Pieces of eight: 8-bit neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 3 (Industry Papers)*, pages 114–120, 2018.

[Strubell *et al.*, 2019] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3645–3650, 2019.

[Sze *et al.*, 2017] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[Wang *et al.*, 2019] Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. Learning deep transformer models for machine translation. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 1810–1822. Association for Computational Linguistics, 2019.

[Wu, 2020] Ephrem Wu. Learning accurate integer transformer machine-translation models. *CoRR*, abs/2001.00926, 2020.

[Xiao *et al.*, 2019] Tong Xiao, Yinqiao Li, Jingbo Zhu, Zhengtao Yu, and Tongran Liu. Sharing attention weights for fast transformer. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 5292–5298. ijcai.org, 2019.