Towards Fully 8-bit Integer Inference for the Transformer Model

Ye Lin1,*, Yanyang Li1,†, Tengbo Liu1, Tong Xiao1,2†, Tongran Liu3 and Jingbo Zhu1,2
1Natural Language Processing Lab., Northeastern University, Shenyang, China
2NiuTrans Research, Shenyang, China
3CAS 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].

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]:

\[ Q(r, s) = \left\lfloor \frac{s}{r} \right\rfloor \]  

(1)

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

\[ D(x, s) = \frac{x}{s} \]  

(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 overflow1.

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 of inference.

1For 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:

$$
\text{Attention}(Q, K, V) = \text{SoftMax}(\frac{QK^T}{\sqrt{d_m}})V
$$

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

$$
\text{SoftMax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}
$$

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

$$
\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2
$$

$$
\text{ReLU}(x) = \max(0, x)
$$

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:

$$
\text{LN}(x) = g \circ \left( \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \right) + b
$$

where $\mu$ and $\sigma$ are the mean and variance of $x$ along the hidden dimension, and $\epsilon$ 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].

## 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 $s = 2^{p-1} \max(|r|)$ 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:

$$
M(x_i, s_i) = \{x_i/\bar{s} s, \bar{s}\}
$$

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.

![Figure 2: Examples of initializing scale and multiplying scale in MatMul.](image-url)
Table 1: FP32 operations in INT8. \( r_i = D(x_i, s_i), \) \( i \in \{1, 2\}, r_i \in \mathbb{R}^{m \times n}, x_i \in \mathbb{Z}^{m \times n}, s_i \in \mathbb{R}^{m \times 1}. \| \) denotes the concatenation. \( \cdot \) denotes the element-wise multiplication.

**Algorithm 1 Scale Propagation Protocol**

**Input:** Operation \( \text{OP}; \) INT8 Tensors \( x_{1, ..., n}; \) Scales \( s_{1, ..., n} \)

**Output:** INT8 Tensor \( x; \) Scale \( s \)

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

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:

\[
R(x, s) = \frac{x}{s} \quad s > 0
\]

where \( s = \left\lfloor \frac{\max(|x|)}{2^n-1} \right\rfloor \). 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,
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 $e^{x+y} \neq e^x e^y$. 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:

$$
Poly(x) = [ReLU(x + b)]^n + |\delta| 
$$

where $b$ is the bias term, $n$ is the degree of the polynomial function and $\delta$ 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 $Poly(x)$ with $V$ and then divide the result by $\sum_j Poly(x_j)$, otherwise the integer division will incur all 0 results because $Poly(x) \leq \sum_j Poly(x_j)$:

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

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:

$$
L1LN(x) = g \odot (\frac{x - \mu}{C \|x - \mu\|_1/n}) + b
$$

where $g$ and $b$ are two parameters, $\mu$ 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 $\mu$ 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 02 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-
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.

### 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 data preparation become marginal. 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 equals to 4× 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 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

<table>
<thead>
<tr>
<th>System</th>
<th>BLEU</th>
<th>PPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>En→Ro</td>
<td>32.55</td>
<td>99.58</td>
</tr>
<tr>
<td>+Poly</td>
<td>32.56</td>
<td>99.59</td>
</tr>
<tr>
<td>+L1LN</td>
<td>32.57</td>
<td>99.60</td>
</tr>
</tbody>
</table>

Table 4: The ablation study of Integer Transformer.

### 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.

<table>
<thead>
<tr>
<th>Entry</th>
<th>System</th>
<th>BLEU</th>
<th>PPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>En→Ro</td>
<td>32.55</td>
<td>99.58</td>
</tr>
<tr>
<td></td>
<td>+Poly</td>
<td>32.56</td>
<td>99.59</td>
</tr>
<tr>
<td></td>
<td>+L1LN</td>
<td>32.57</td>
<td>99.60</td>
</tr>
</tbody>
</table>

Table 5: INT8 performance vs. different sized scales.
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.

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


