Vol.2021-ARC-243 No.32
Vol.2021-SLDM-193 No.32
2021/1/26

Quantization Techniques for Small Number of Bits in Transformer based Natural Language Processing

YI DING[1] MASAHIRO FUJITA[1,2]
SHIN-ICHI O'UCHI[2]

Abstract: After decades of development, natural language processing is widely used in life and research. In recent years, the Transformer-based NLP model has achieved good results on many tasks, in which a model called BERT is a major progress in recent years. However, the BERT model is really large, which requires a lot of storage and computing resources. In order to implement the BERT model on hardware, Intel has built a project named Q8BERT to quantize the model parameters to save memory space and computing resources. This paper implements the Q8BERT, and makes some improvements by proposing clipping, which is to limit the range of weights, and piece-wise quantization, which is to divide the range of weights into several pieces to get higher resolutions. We show experimentally that the clipping can get higher accuracy while inference, which is better than the Q8BERT. And we get some intermedium results of non-linear experiments. Experiments for piece-wise quantization are in progress.

Keywords: NLP, Transformer, BERT, Quantization

1. Introduction

With the development of artificial intelligence, many NLP models based on neural networks like GPT [1] have been proposed. Many of them have good performance, show good results on many tasks, and are widely used in many places, such as automatic translation, AI voice, even medical care. But at the same time, in order to meet these performance requirements, the size of the model is getting larger and larger, taking up a lot of memory and computing resources, requiring GPUs or TPUs. In order to allow these models to operate on hardware such as FPGAs-ASICs with fixed point integer calculations, quantization techniques are proposed.

2. Background

2.1 Natural Language Processing

Natural language processing (NLP) usually refers to making computers understand and process languages that humans can understand, such as English, Chinese, Japanese, and even artificial languages. NLP has been developed for decades, and now with the rise of neural networks, many NLP models based on neural networks have been proposed, such as models based on RNN or LSTM [2] in the last several years. NLP has now been applied to a plenty of fields, providing many conveniences for our lives and work, such as automatic translation tools, AI speech recognition and so on.

2.2 Attention Mechanism

NLP tasks are usually processing a sequence of unknown length. It is different from image processing. Image processing can transmit an entire image to CNN as input at the same time, and perform parallel processing to obtain the results. However, the input of NLP is usually related to each other. Sequence to sequence architecture is one of the structures to solve this kind of sequence input. In the structure, Encoder translates the input sequence into intermediate semantic C, and Decoder decodes semantic C into the target text sequence. RNNs are often used for this mapping with their recursion mechanism.

Although RNN has some memory properties, in Encoder-Decoder architecture, no matter how the input changes, the encoder gives a vector of fixed dimensions, and in the decoder, the semantic vector C which is used to generate each target text. This is obviously not good, because for each target text, the weight of each word in the input sequence is different.

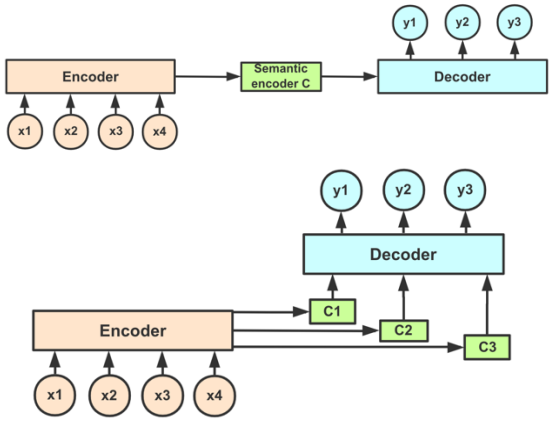


Figure 1: Encoder-Decoder model (Up) and with Attention (Down)

A mechanism called Attention [3] makes some optimization on such problems. The attention mechanism, as its name, is inspired by human attention. Imagine when we look at a portrait, although

1 Graduate School of Engineering, The University of Tokyo, Bunkyo, Tokyo 113-8654, Japan
2 AIST-UTokyo AI Chip Design Open Innovation Laboratory (AIDL), National Institute of Advanced Industrial Science and Technology, Bunkyo, Tokyo 113-8654, Japan

we can see the whole painting, our attention is first attracted by the portrait, and secondly the landscape behind the portrait, which means the weight of the portrait is larger than the weight of the background. The same is true for the Attention mechanism, and when generating each word, it gives different weights for different input words. Through the self-attention mechanism, we can also obtain the internal relationship among the input sentence itself.

2.3 BERT

With the introduction of Attention, researchers have proposed a new NLP model, completely abandoning the traditional RNN or LSTM structure, and only consisting of Attention., which is called Transformer [5]. Transformer divides the model into encoder and decoder. Each contains several layers of Attention structure.

BERT (Bidirectional Encoder Representations from Transformers) [4] is an improved and optimized model based on Transformer. Transformer [5] is a unidirectional connection. BERT uses a bidirectional architecture to improve the performance of the model and has achieved SOTA results on many NLP tasks.

3. Related Works

3.1 Linear quantization

Linear quantization [6] is a kind of uniform quantization. Symmetric linear quantization meanly maps 32-bit floating-point numbers to linear integer space of 8-bit integers ([-127, 127]). The symmetrical linear quantization calculation is simpler because there is no need to consider the offset caused by the position of 0. The quantization calculation can be simply represented by the following formula:

$$quantize(x) = clamp\big(-thresh, thresh, round(x * scale)\big)$$

Where the *scale* is the quantization conversion ratio, and *thresh* is the maximum allowable value after quantization, which in this case is 127. The *round* function obtains the nearest integer by rounding. *clamp* is to prevent data overflow. The quantized integer data can be restored to FP32 data by the following formula:

$$dequantize(x) = \frac{x}{scale}$$

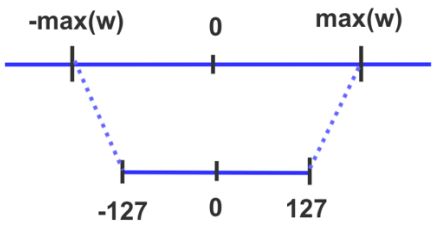


Figure 2: 8bit symmetric linear quantization

3.2 Post-training quantization

Post-training quantization is a relatively simple quantization method. After the training of the original floating-point number model is completed to obtain the FP32 model, the weights in the FP32 model can be directly quantized into integers by the above formula. By this, quantization can reduce the size of the model several times in a short time. However, in the inference process, we still need to use FP32 format for calculation. This is because the range of the input variable is not known, so it is still necessary to dequantize the quantized weight to FP32 for calculation.

Therefore, quantization after training is suitable to reduce the size of the model, but it cannot reduce the amount of calculation, and it is difficult to apply it to hardware. In addition, this method also brings some precision loss due to rounding when quantized to an integer.

3.3 Quantization-aware training [7]

In order to obtain higher precision, and can also perform calculations using only quantized integers during inference to reduce computing resources and be suitable for hardware implementation, it is necessary to perform quantization during training to continuously obtain and update the input range and quantization scale.

3.4 Q8BERT

Intel's AI laboratory combines Linear quantization and Quantization-aware training and implements it on the BERT model. It completes the 8-bit linear quantization for the BERT model, which is called Q8BERT [8].

As we introduced in 2.1, Q8BERT uses symmetric linear quantization. In the formula, *scale* determines the range and resolution after quantization, so Q8BERT learns the range and scale of input through fake-quantization training, so that all *scale* and *thresh* parameters of the quantization model after training can be fixed in order to use these fixed parameters to perform inference with only integers.

Fake-quantization is a training process that includes forward and backward propagation.

$$fakequantize(x) = dequantize(quantize(x))$$

In this process, because the *round* function is not derivable, in order to obtain gradient, STE [9] was introduced instead of the gradient calculation function. STE (straight-through estimator) is a method commonly used in neural network gradient calculations. It can replace some underivable functions in the training process to generate gradients so that the model can be trained and learned normally. After the training is completed, the inference can be performed using only integers.

4. Proposed Improvements

Q8BERT has achieved good results in 8bit quantization, but it still requires a lot of storage and computing resources. Once it is less than 8bit, the result becomes much worse. We hope to make some improvements and make comparisons to further reduce the number of bits while maintaining accuracy.

4.1 Clipping

We analyzed the reason why the accuracy is greatly reduced when directly quantizing into lower bits. 8bit means there are 256 quantized integers which can be mapped, while there are only 16 integers for 4bit, the number is only 1/16 of 8bit. This means that the interval between the two numbers has increased by 16 times and the resolution has dropped by 16 times. The weights of BERT follow a normal distribution, so smaller weights have higher frequencies, but too large intervals make these small weights all quantized to the same value (such as 0), and too many values are quantized to 0, the performance of the model becomes terrible. This is because in the Attention structure, the output passed to the next layer is the matrix multiplication of the input and weights. As shown in the figure, in the 4-bit model, 75% of the weights are quantized to integer 0, which is only 12% in 8bit model. Therefore, as the results show, the performance descending of the low-bit model is very serious.

$$w_i = \begin{cases} w_i, & w_i < threshold \\ threshold, & w_i \geq threshold \end{cases}$$

As mentioned above, when the number of bits is determined, the resolution is determined by the maximum value. If the maximum value is quite large, most weights can only obtain very low resolution, which greatly reduces the accuracy. At the same time, the greater the weights, the lower the frequency. If the errors generated by ignoring these low-frequency large values are small enough, then we can reduce the maximum by ignoring them, and the resolution can be greatly improved, resulting in higher accuracy. This is the idea of clipping [10].

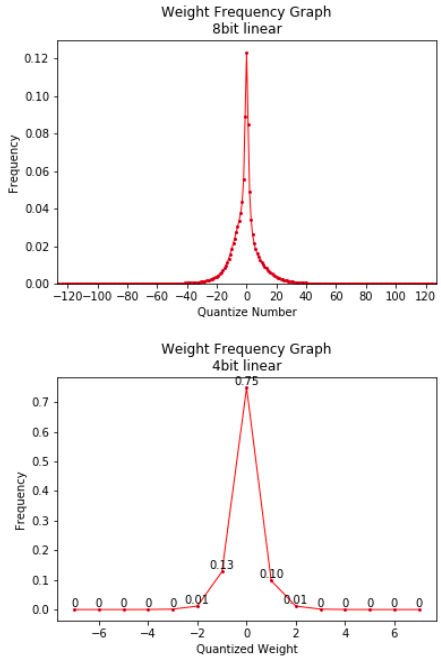


Figure 3: Weight frequency of 8bit-linear (Up) and 4bit-linear quantization (Down)

But we need to find a suitable clipping position. If the value of the position is too large, it has little effect on improving the resolution. If this value is too small, the accuracy may be reduced because of limiting too many large values. A very popular but effective method is to calculate the MSE (Mean Squared Error) of quantization after clipping [12].

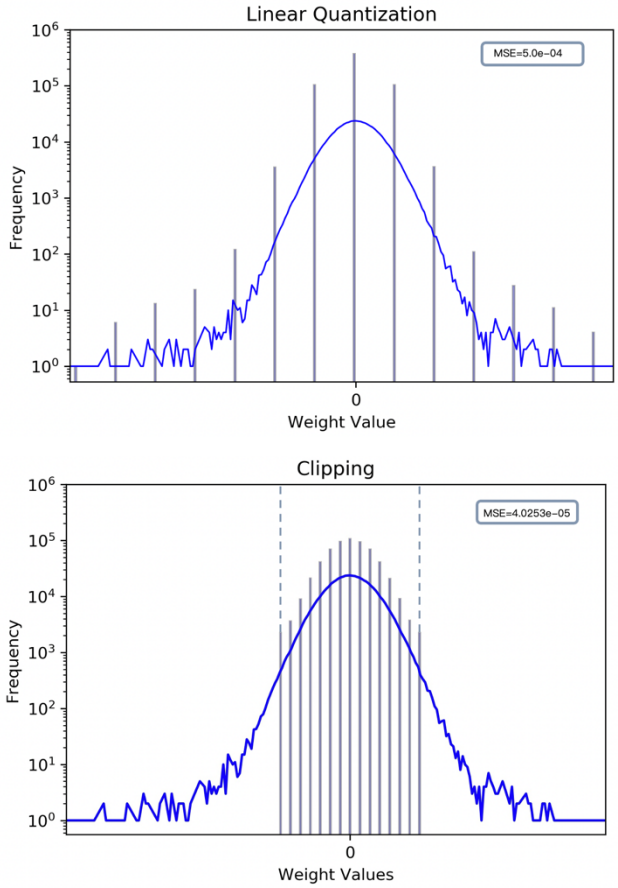


Figure 4: Weights in one layer of BERT showing clipping shortens the interval and increases the resolution

We divide the area of weights [0, maximum] into several parts, set the clipping value as the point in the end of each part, then do the quantization and compare the MSE generated after clipped quantization, and select the point with the smallest MSE as the clipping position. Through this method, the MSE generated by quantization can be significantly reduced, and the resolution can be improved several times, thereby obtaining higher accuracy.

$$MSE = \frac{1}{n}\sum_{1}^{n}(qw_i - w_i)^2$$

4.2 Non-linear quantization

As we analyzed in section 4.1, the normal distribution of weights results in smaller weights having higher frequencies, while larger weights have lower frequencies. The Clipping result validates the importance of the resolution with a smaller weight to a certain extent. The demand for resolution with small weights is far larger than that with large weights. Therefore, we associate the Log function in the non-linear function [11].

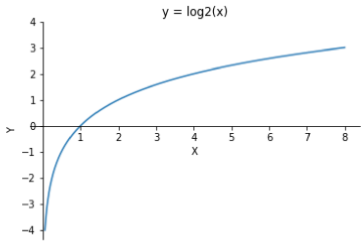


Figure 5: Log function

As shown in the figure, the gradient of the log function quickly becomes smaller as the input becomes larger. This just meets our needs. With the same number of bits, the Log function has higher resolution, and has a larger representation range due to its non-linearity. Therefore, Log scale quantization may be a good solution.

After the above transformation, the matrix multiplication of the original linear space becomes the following operation.

The reason why 2 is selected as the bottom of the log function is that it can change the exponential operation into a simple shift operation. For hardware, the complex multiply-add operation is simplified for addition and shift operations, which should save a lot of hardware resources.

$$w^T x \cong \sum_{i=1}^{n} 2^{Quantize(w_i)+Quantize(x_i)}$$
$$= \sum_{i=1}^{n} 1 \ll (Quantize(w_i) + Quantize(x_i))$$

But the log function has a problem that it can only handle positive numbers. This means that negative numbers must be absolute values before they can be processed by the log function. But in this case, there will be a problem that the original sign of the variable cannot be restored during the dequantization process and the inference process, which causes the result to be completely wrong in the accumulation operation.

A simple solution is to create an additional sign matrix for each weight matrix to store FP32 real number signs. In the experiment of this article, in order to make the experiment easier and avoid creating additional storage space, another similar method is adopted. In order to retain both the signs before and after quantization, an additional bit is used as the sign bit of original FP32 value, but the specific implementation method is to add an extra bit as the sign bit of the quantized data, which becomes 9bit integers. During the operation, the sign of the original data is restored by judging the state of this bit.

4.3 Piece-wise quantization

Piece-wise is a multi-segment linear quantization, which divides the original data into two parts, we call them ***w1*** and ***w2***. ***w1*** and ***w2*** both occupy 7 bits, which together occupy 8 bits in total. If it is single quantization, then the split point of ***w1*** and ***w2*** is at the midpoint of the range ***max(x)/2*** (equivalent to a special piece-wise). When we move the position of the segmentation from the middle to the left, the range of ***w1*** gradually shrinks. Therefore, when it is quantized into a 7-bit integer, the data in ***w1*** gets a greater resolution. At the same time, the data in ***w2*** obtains a smaller resolution. Since the weights of the neural network conform to the normal distribution, more data near ***0***. Therefore, moving the split point to the left effectively makes the resolution of most data better.

It would be bad if we completely ignore the accuracy of the right half, so we need to find a suitable split position. The piece-wise quantization can be equivalent to a double linear quantization problem, that is, ***w1*** is mapped to a 7-bit integer ***qw1*** through linear quantization, and ***w2*** is mapped to a 7-bit integer ***qw2*** through linear quantization, and then they are combined to occupy just 8 bits.

Since ***w1*** and ***w2*** are actually in the same matrix, they are the results of 7bit quantization, so there may be duplication, so we need to process to distinguish them.

An obvious way is to use the most significant bit. ***qw1*** occupies -0000000~+0111111, while ***qw2*** occupies -1000000~-1111111, and +1000000~+1111111. When processing data, we can judge whether they belong to ***qw1*** or ***qw2*** through the most significant bit. In the subsequent processing, reset the highest bit of ***qw2*** to ***0***.

If we use ***thresh*** to represent the split position, the *quantize* and *dequantize* operations can be expressed by the following formula:

$$qw = \begin{cases} round(w * scale1) & , abs(w) \le thresh \\ round(((abs(w) - thresh) * scale2 + offset) * sign(w)), & others \end{cases}$$

$$w = \begin{cases} round\left(\frac{qw}{scale1}\right) & , abs(w) < offset \\ \frac{round(abs(qw) - offset + q_{thresh})}{scale2} * sign(qw)), & others \end{cases}$$

where the ***offset*** means how much the difference led by the most significant bit.

$$offset = 2^{bits-2}$$

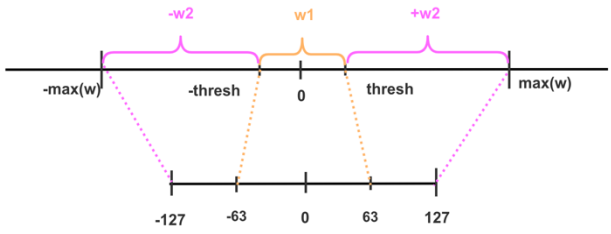


Figure 6: Piece-wise quantization

Another problem is that, this is equivalent to two 7-bit linear quantization, so they have different scales. This brings trouble to the subsequent matrix multiplication and addition operations. Different scales make the final multiplication and addition incorrect, and cannot be dequantized and mapped back to FP32 values.

Table 1: The results of original Q8BERT and after Clipping on different Benchmarks and bis

Benchmark	Metric	Bits	Direct	Clipping	Relative Improvements
MRPC	F1	8	**89.16**	89.12	-0.04%
		7	85.14	**89.86**	5.54%
		6	83.2	**86.61**	4.10%
		5	79.53	**83.2**	4.61%
		4	**81.22**	80.5	-0.89%
RTE	Accuracy	8	64.98	**65.7**	1.11%
		7	58.48	**63.18**	8.04%
		6	53.43	**59.21**	10.82%
		5	52.35	**58.12**	11.02%
		4	**52.7**	47.65	-9.58%
STS-B	Pearson	8	88.52	**88.67**	0.17%
		7	87.27	**87.6**	0.38%
		6	82.27	**86.76**	5.46%
		5	63.77	**81.4**	27.65%
		4	11.56	**25.52**	120.76%
CoLA	Matthew's	8	56.24	**57.06**	1.46%
		7	50.24	**51.55**	2.61%
		6	0.0	**27.52**	-
		5	-	-	-
		4	-	-	-

qw1 and ***qw2*** are distinguished by the most significant bit, so the quantized matrix ***qw*** can actually be decomposed into two matrices ***qw1*** and ***qw2*** (extract the part of ***qw1***, and the rest fill with ***0***; extract the part of ***qw2***, the rest Add ***0***). Therefore, the multiplication and addition operations are equivalent to two parts: ***sum(x*qw1)***, ***sum(x*qw2)***. But these two parts cannot be added directly, because their different scales make the scale of the product impossible to calculate and lead to incorrect results. Because the scale of the product is equal to ***scale_x * scale_w***. But there are two different ***scale_w*** here. Should it be ***scale1*** or ***scale2***? Both are wrong. Let's see some mathematical operations. Since ***qw1 = w*scale1***, ***qw2 = w*scale2***, if you want to get the correct result at the end, you only need to do an extra multiplication before the final accumulation. That is,

$$q_{product} = (\sum qx * qw1) * scale_2 + (\sum qx * (qw2 + qthresh)) * scale_1$$

The ***qthresh*** means the quantized ***thresh***, which represents the position of dividing. ***scale1*** is the scale of ***qw1***, and ***scale2*** is the scale of ***qw2***. In this way, the scale of the output activations can be calculated.

$$scale_{out} = scale_x * scale_1 * scale_2$$

A remained problem is that if we implement on hardware, ***scale1*** and ***scale2*** are both floating-point numbers, which are not easy to calculate. This also has a corresponding solution. We can convert scale1 and ***scale2*** in the additional multiplications into integer multiplications according to their ratios. The principle is very simple, just find the common multiple of scale1 and ***scale2***, and then divide to get their ratio. For example, if we select ***10%*** of the entire range as the split point, it means ***scale1 = 9 * scale2***. Therefore, the formula can be changed into:

$$q_{product} = \sum qx * qw_1 * 1 + \sum qx * qw_2 * 9$$

$$scale_{out} = scale_x * scale_1 * 1$$

$$scale_{out} = scale_x * scale_2 * 9$$

Through the above operations, we can apply this method to the experiment and compare with the results of clipping. In addition, we can combine piece-wise and clipping together.

The final thing we would like to do is to find proper split positions, which can lead to the highest accuracy. One possible way is similar to the method in the clipping, which is to divide the range of weights into n pieces and choose one which leads to the minimum MSE.

5. Experiment

We conducted clipping and non-linear experiments respectively. Experiments show that the clipping is effective in the BERT quantization model, and improvements have been made in most of the results of the experiments. In the experiments for non-linear quantization, we tested and compared the relative error generated by Non-linear and the linear quantization for different bits. However, in the experiment applying the non-linear to the BERT model, we did not achieve the ideal results. We will analyze about this in section 5.2, and improve this part of the experiment in future work.

Table 2: The results of original 4bit-linear quantization and out 5bit-non-linear on different Benchmarks and bits

Benchmark	Metric	4bit-Linear	5bit-Non-linear	Relative Improvements
MRPC	F1	81.22	81.22	0.00%

5.1 Clipping Experiment

In the clipping experiment, in order to retain the same conditions, we used pre-trained BERT model released by Google, and Glue Benchmark, same as the Q8BERT. Glue contains several different training and test data.

The experimental results show that the model after clipping has results better than the original Q8BERT in most cases. Among them, the precision improvement produced by 7bit, 6bit and 5bit quantization is particularly obvious.

But in the 4bit of RTE, the results of clipping have dropped significantly. We analyzed the reasons for such results. Because in RTE, 5bit and 4bit have almost reached the prediction result of all "1" (the correct prediction answer has about 52.7% of "1"). This means that there are too many "0"s in the quantized model, which leads to poor model performance and almost no classification ability. After the clipping, more parameters in the model have non-zero values, which makes the prediction results no longer all "1".

In short, clipping at 5bit, 6bit and 7bit can effectively improve the prediction accuracy. But in the 4bit, it is very unstable. We think this is because even if we do the clipping, 4bit still can only have 16 quantized values available for mapping, so the resolution is still too low. If we want to further ensure accuracy with lower bit, we need further improvements.

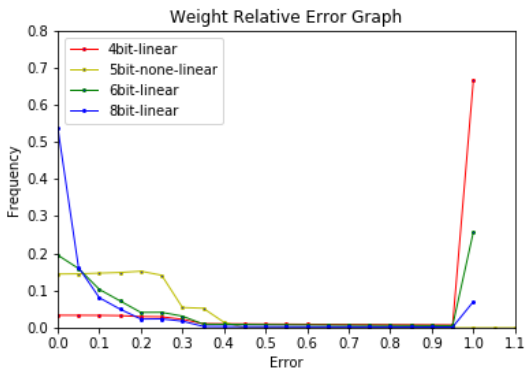
5.2 Non-linear Experiment

In order to verify the feasibility of non-linear, we did some comparative experiments to judge the error performance of non-linear quantization.

Because the range of 5bit Log quantization can reach $[2^{-15}, 2^{15}]$, which covers all the weight values, we conduct experiments on 5-bit Log quantization. The first figure is the relative error produced by different quantization methods on the weights. It can be seen that as the relative error increases, the frequency gradually decreases, but there is an abnormal protrusion at the place where the relative error is 100%. This is because some of the smaller values are quantized to 0, so the relative error becomes 100%. 4bit linear quantization has 70% of the values with a relative error of 100%, meaning that these values are almost all quantized to 0, and 6-bit is less, but even with 8-bit linear quantization, there are still many relative errors with weight values reaching 100%. Non-linear quantization does not decrease in frequency like linear quantization under a small relative error, because the quantized value and error distribution of non-linear quantization are not uniform. But there is no relative error of 100%, which is one of the advantages of non-linear quantization. Overall, nonlinear quantization seems to be more stable than linear quantization, with most values low error.

Because BERT does not have a traditional CNN convolutional layer, the Attention calculation method is similar to the FC (full-connection) layer, which is a General Matrix Multiply operation (GEMM). Therefore, we did another experiment to compare the relative errors generated after GEMM. The 4-bit linear quantization has a peak at 1.0 because the frequency of 0 is too high, and other quantization methods are similar decline curves. 5bit non-linear is slightly better than 6bit linear quantization. This looks not as good as expected. The reason for this is still because of the non-uniformity of the log function, the error generated on the larger value is larger. But in general, non-linear should be feasible and can get relatively good results.

In addition, we also applied non-linear quantization to the BERT, but there is some trouble in the experiment, resulting in poor results, which does not meet our expectations. The 5bit-Non-linear is the same as 4bit-Linear, while 4bit-linear has no classification ability because all the predictions of 4bit-linear is “0”.



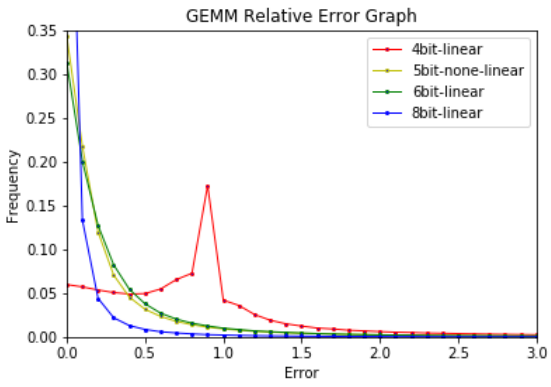


Figure 7: Weight relative error (Up) and GEMM relative error (Down) of different methods

At present, we suspect that the STE used in the backward propagation of linear quantization is not suitable for nonlinear quantization. We need more information about it.

5.3 Piece-wise Experiment

We are currently conducting the experiments of piece-wise quantization and combination of the clipping and piece-wise quantization.

6. Conclusion

In this work, we implemented Clipping, none-linear quantization and Piece-wise quantization to the original quantization. The results show that under the same bit, Clipping has higher accuracy in most cases. This confirms that clipping has an improvement in the current quantization of the BERT model. But there are still some bad results. Reducing the error caused by clipping [12] and also clipping on quantized activations [13] may make further improvements. We conducted some non-linear experiments and obtained some intermediate results, which showing non-linear is potentially a good method for improvement. And we are conducting the experiments for the piece-wise quantization.

Reference

[1] Radford, Alec, et al. "Improving language understanding by generative pre-training." (2018): 12.
[2] Olah, Christopher. "Understanding lstm networks." (2015).
[3] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *arXiv preprint arXiv:1409.0473* (2014).
[4] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).
[5] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017): 5998-6008.
[6] Krishnamoorthi, Raghuraman. "Quantizing deep convolutional networks for efficient inference: A whitepaper." *arXiv preprint arXiv:1806.08342* (2018).
[7] Jacob, Benoit, et al. "Quantization and training of neural networks for efficient integer-arithmetic-only inference." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.
[8] Zafrir, Ofir, et al. "Q8bert: Quantized 8bit bert." *arXiv preprint arXiv:1910.06188* (2019).
[9] Bengio, Yoshua, Nicholas Léonard, and Aaron Courville. "Estimating or propagating gradients through stochastic neurons for conditional computation." *arXiv preprint arXiv:1308.3432* (2013).
[10] Shin, Sungho, Kyuyeon Hwang, and Wonyong Sung. "Fixed-point performance analysis of recurrent neural networks." *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016.
[11] Miyashita, Daisuke, Edward H. Lee, and Boris Murmann. "Convolutional neural networks using logarithmic data representation." *arXiv preprint arXiv:1603.01025* (2016).
[12] Zhao, Ritchie, et al. "Improving neural network quantization without retraining using outlier channel splitting." *arXiv preprint arXiv:1901.09504* (2019).
[13] Choi, Jungwook, et al. "Pact: Parameterized clipping activation for quantized neural networks." *arXiv preprint arXiv:1805.06085* (2018).