Model Compression by Quantization for RoBERTa

By Zezhong Fan & Yiquan Li

Executive Summary

Problem Statement:

RoBERTa is an improved version of BERT and is a state-of-the-art language model that has become a popular choice in NLP field. Consider its large size, we would like to apply quantization on the model for model compression, and find a good deployment-accuracy tradeoff among different quantization method combinations.

Solution Approach:

- Use uncompressed pre-trained RoBERTa-Base model as baseline, fine-tune the model based on different tasks;
- Apply integer-only quantization on both linear and non-linear operations by quantization-aware training;
- Evaluate model performance through test accuracy, training time and inference time.

Value & Benefit:

We could explore the possibilities for integer-only quantization on non-linear operations in DL models, and find a good quantization technique that require appropriate training time while keeping the model accuracy.

Problem Motivation

- RoBERTa is an optimized BERT pretraining approach that has been widely used in NLP research and in industry. But it generally has a even larger model size than BERT since it is trained on a larger dataset. Quantization could be helpful in reducing memory footprint for the model;
- In real world scenarios, many applications of deep learning need to be applied on edge devices, so model compression is important for deploying DL models with constrained resources and having feasible inference time;
- Integer-only quantization can achieve larger model deployment cost reduction than floating-point quantization;
- Applying end-to-end quantization may cause increase in model training time. So instead of quantizing all layers, we could quantize part of them while maintaining the model accuracy.

Background Work:

1. **Roberta Model:** Proposed in RoBERTa: A Robustly Optimized BERT Pretraining Approach. Model Link: https://huggingface.co/docs/transformers/model_doc/roberta. It builds on BERT and modifies key hyperparameters, removing the next-sentence pretraining objective and training with much larger mini-batches and learning rates

2. **Simulated(Fake) Quantization:** Q-BERT: Hessian based ultra low precision quantization of bert(https://arxiv.org/pdf/1909.05840.pdf). Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference(https://arxiv.org/pdf/2005.03842.pdf)

3. **Integer-Only Quantization on Pytorch:** I-BERT: Integer-only BERT Quantization(https://arxiv.org/abs/2101.01321): The model stores all parameters with INT8 representation, and carries out the entire inference using integer-only arithmetic

Technical Challenges

Before Model Training:

- **Understand the I-BERT paper (including the Appendix)**
  - We need to learn about quantization and understand the mathematical theory/algorithm behind different quantization approaches mentioned in the I-BERT paper
- **The selection of quantization method combinations**
  - Due to limitation on hardware and time, it is not feasible to evaluate every single combination of quantization. We should choose the combinations that can lead to effective results.

During Model Training:

- **Intensive network training**
  - The network has 6 variations; each for a different quantization combination
  - For testing robustness, we perform comparisons for 4 NLP tasks (MNLI, SST-2, CoLA, RTE)
  - Quantization-aware training acquires longer training time. One epoch can take up to 2 hours.
  - Intensive need for disk memory and RAM due to large dataset and stored checkpoints

Technical Challenges

During Model Training (Continued):

- **Use of Fairseq**
  - We need to derive and modify the Fairseq toolkit so that it can support both non-quantized and quantized model fine-tuning

After Model Training (Limitation):

- **Model evaluation**
  - PyTorch does not support integer operations thus the Pytorch implementation does not achieve speedup for quantization on real hardware by itself, and the model size needs to be manually computed.

Quantization Approach

Basic Quantization Method:

- Uniform Symmetric Quantization:

$$q = \mathrm{Q}(x, b, S) = \mathrm{Int}\left(\frac{\mathrm{clip}(x, -\alpha, \alpha)}{S}\right)$$

where ***x*** is the real number we need to quantize, ***b*** specifies the quantization bit precision, ***S*** is the scaling factor $\alpha/(2^{b-1}-1)$, and ***clip*** is the truncation function that limit the values in which α is the clipping parameter.

Quantization for Non-linear Operations :

- Polynomial Approximation of Non-linear Functions:

  Interpolating polynomial $L(x) = \sum_{i=0}^{n} f_i l_i(x)$ where $l_i(x) = \prod_{\substack{0 \le j \le n \\ j \ne i}} \frac{x - x_j}{x_i - x_j}$

  where $\{(x_0, f_0), \ldots, (x_n, f_n)\}$are given data points. Then apply integer-only computation.

- Newton's Method

Quantization Approach

1. Polynomial Approximation of Non-linear Functions:

- **Integer-only GELU**
  - GELU is a non-linear activation function used in Transformer models:

$$\mathrm{GELU}(x) := x \cdot \frac{1}{2}\left[1 + \mathrm{erf}(\frac{x}{\sqrt{2}})\right], \text{ where } \mathrm{erf}(x) := \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2)dt.$$

    erf is the error function and is the nonlinear part in GELU.
  - Use a 2nd-order polynomial to approximate the ***erf*** function:

$$\min_{a,b,c} \frac{1}{2} \left\| \mathrm{GELU}(x) - x \cdot \frac{1}{2}\left[1 + \mathrm{L}(\frac{x}{\sqrt{2}})\right] \right\|_2^2, \text{ s.t. } L(x) = a(x+b)^2 + c,$$

$$\Rightarrow \quad \mathrm{L}(x) = \mathrm{sgn}(x)\left[a(\mathrm{clip}(|x|, \max = -b) + b)^2 + 1\right]$$, where ***a***=-0.2888, ***b***=-1.769

  - Final integer-only approximation for GELU:

$$\text{i-GELU}(x) := x \cdot \frac{1}{2}\left[1 + \mathrm{L}(\frac{x}{\sqrt{2}})\right]$$

Quantization Approach

- **Integer-only Softmax**
  - Softmax: $$\text{Softmax}(\mathbf{x})_i := \frac{\exp x_i}{\sum_{j=1}^{k} \exp x_j}, \quad \text{where } \mathbf{x} = [x_1, \ldots, x_k].$$

    exp is the nonlinear part in Softmax.
  - Use a 2nd-order polynomial to approximate ***exp***:

    Subtract *max(x)* for numerical stability: $$\text{Softmax}(\mathbf{x})_i = \frac{\exp(x_i - x_{\max})}{\sum_{j=1}^{k} \exp(x_j - x_{\max})}$$

    Let $\tilde{x}_i = x_i - x_{\max}$, then $\exp(\tilde{x}) = 2^{-z} \exp(p) = \exp(p) >> z$, >> is the bit shifting operation.

    $$\Longrightarrow \quad L(p) = 0.3585(p + 1.353)^2 + 0.344 \approx \exp(p)$$

  - Final integer-only approximation for Softmax:

    $$\text{i-exp}(\tilde{x}) := L(p) >> z \text{ where } z = \lfloor -\tilde{x} / \ln 2 \rfloor \text{ and } p = \tilde{x} + z \ln 2.$$

    and put the approximation ***i-exp*** back to the Softmax function.

Any non-positive real number ***x*** can be decomposed as:
x=(-ln 2)z+p, where
z: non-negative int,
p: real number in ***(-ln 2, 0]***.

2. Newton's Method:

- **Integer-only LayerNorm**
  - Normalization process: $\tilde{x} = \frac{x - \mu}{\sigma}$ where $\mu = \frac{1}{C}\sum_{i=1}^{C} x_i$ and $\sigma = \sqrt{\frac{1}{C}\sum_{i=1}^{C}(x_i - \mu)^2}$

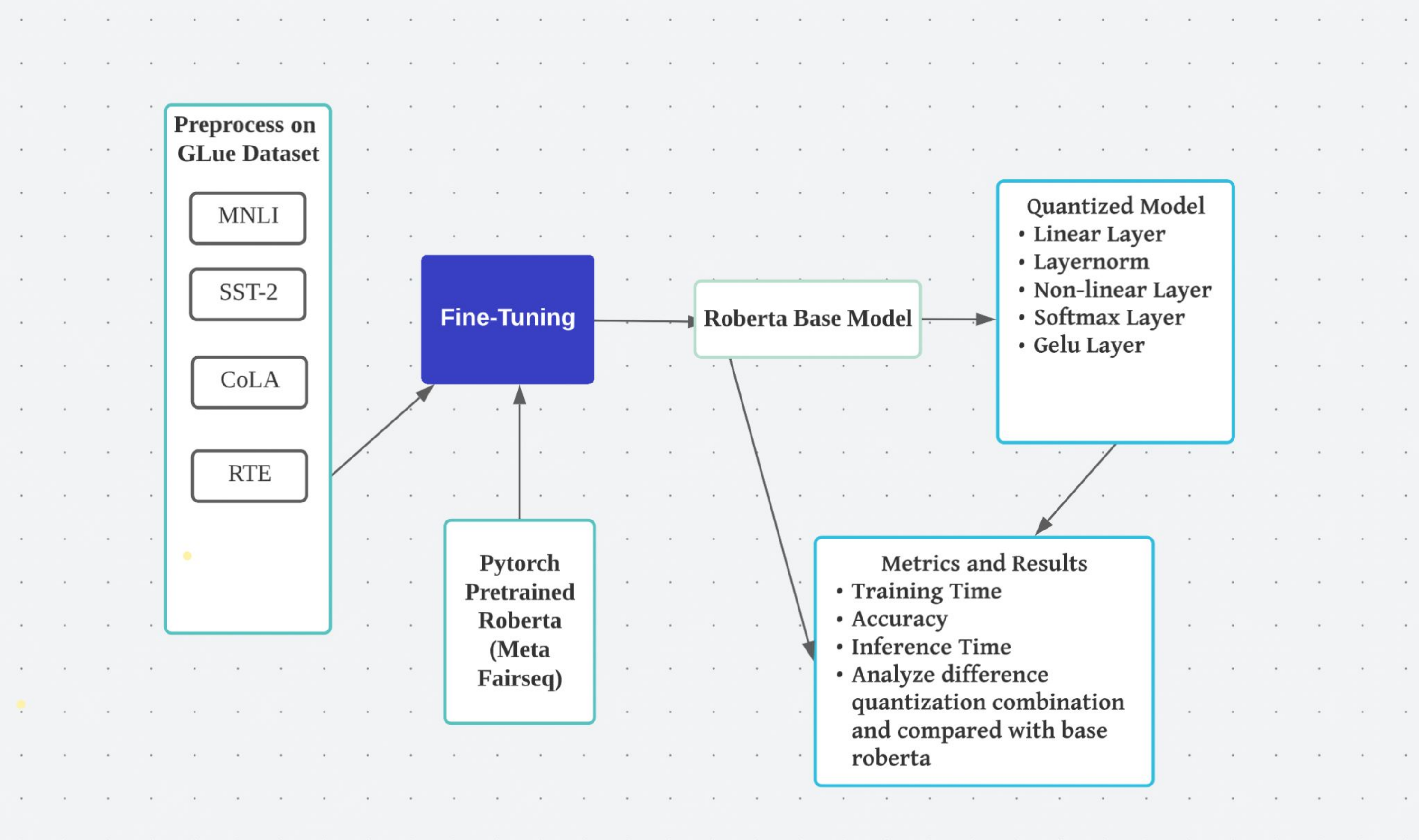
    sqrt in standard deviation is the nonlinear part in LayerNorm.
  - Use of Newton's Method to approximate the ***sqrt*** of a value:

    Integer-only square root:

Input: n: input integer
Output: integer square root of n, i.e., $\lfloor \sqrt{n} \rfloor$

function I-SQRT(n)
  if $n = 0$ **then return** 0
  Intialize x_0 to $2^{\lceil Bits(n)/2 \rceil}$ and i to 0
  repeat
    $x_{i+1} \leftarrow \lfloor (x_i + \lfloor n/x_i \rfloor)/2 \rfloor$
    if $x_{i+1} \geq x_i$ **then return** x_i
    else $i \leftarrow i + 1$
end function

Solution Diagram/Architecture



Implementation Details:

- **Hardware**: Nvidia T4 GPU
- **Platform**: Google Colab
- **Framework**: Pytorch
- **Dataset**: GLUE datasets
- **Functionalities**:
  - *Model*: Fairseq Roberta
  - *Tasks*: CoLA (Corpus of Linguistic Acceptability), MNLI (Multi-Genre Natural Language Inference), SST-2 (Sentiment Analysis), RTE (Recognizing Textual Entailment)
  - *Quantization Combinations*: End-to-end quantization, Linear quantization, Linear+GELU+Softmax, Linear+GELU+LayerNorm, Linear+Softmax+LayerNorm
  - *Metrics*: Test Accuracy, Training Time, Inference Time
- **Limitation:**
  - The the current PyTorch implementation could not achieve latency reduction on real hardware by itself due to PyTorch not supporting integer operations

Experiment Design Flow:

Collect and preprocess GLUE Datasets on four tasks: MNLI, CoLA, SST-2, and RTE

↓

Fine-tuning Roberta base models without quantization on the 4 tasks

↓

Convert the model to quantized model and quantization-aware fine-tuning on the 4 tasks

↓

Test the performance of the best checkpoints for each model using validation/test data

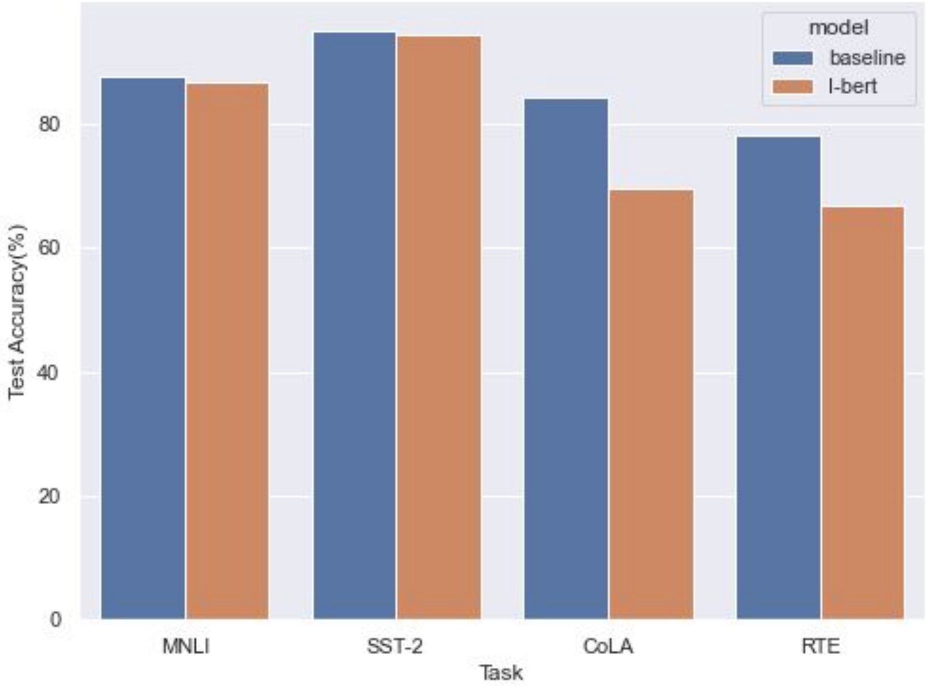
↓

Experimental Evaluation:

- Compare accuracy and training time on base and quantized model
- Compare accuracy and training time for different quantization combinations
- Explore training/inference time-accuracy tradeoffs for different quantization combinations on different tasks.

Experimental Evaluation:

Test accuracy for baseline Roberta and fully-quantized Roberta

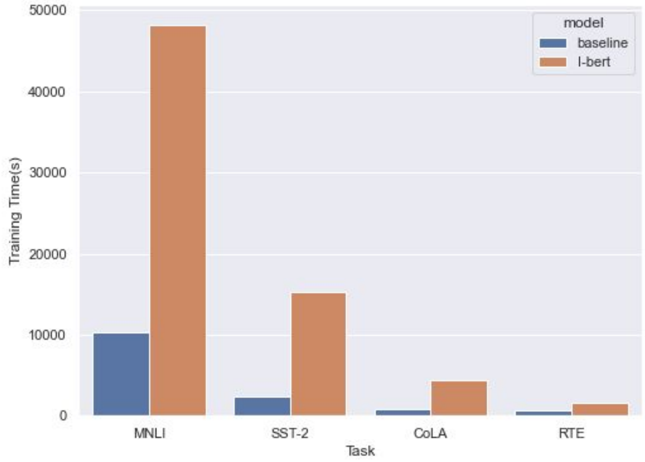


We compare the accuracy between baseline Roberta base model and quantized-aware fine-tuning Roberta on four different tasks.

For MNLI and SST-2, the degradation of accuracy is not obvious. However, for CoLA and RTE, the degradation of accuracy is obvious.

Experimental Evaluation:

Training time for baseline Roberta and fully-quantized Roberta



We then compare the training time(fine-tuning) between the baseline Roberta and quantized Roberta on four different tasks.

For MNLI, SST-2, and Cola, the fine-tuning time of fully quantized model is around 5x more than baseline Roberta. For RTE, quantized model consume 2.5x times than base model.

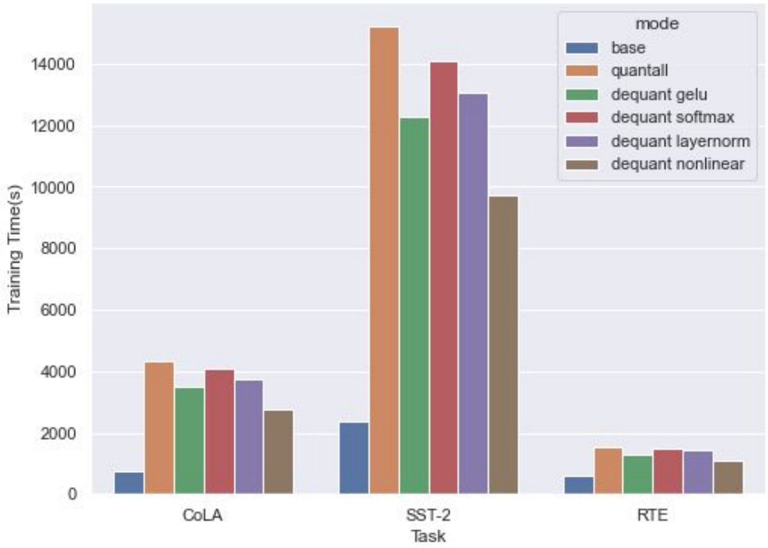
Task	Baseline		I-bert	
	Test Accuracy(%)	Training Time(s)	Test Accuracy(%)	Training Time(s)
MNLI	87.6	10348.4	86.6	48243.7
SST-2	95	2362.5	94.3	15238
CoLA	84.1	766.8	69.4	4349.6
RTE	78	607.3	66.8	1553.5

Table1. Training time and test accuracy for Roberta base and quantized Roberta base.

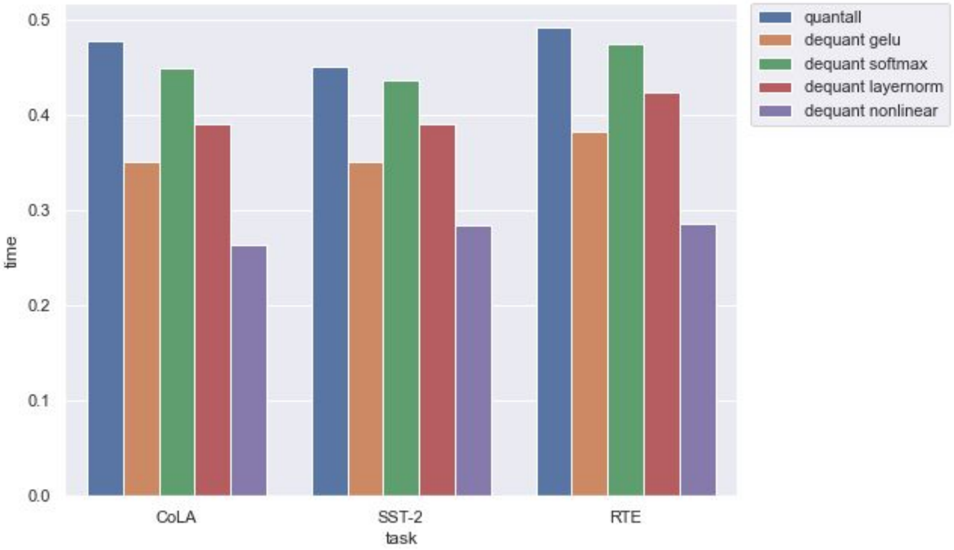
Experimental Evaluation:

Explore training time and inference time for different quantization layers

Training time for different quantization layers and tasks



Inference time for different quantization layers and tasks

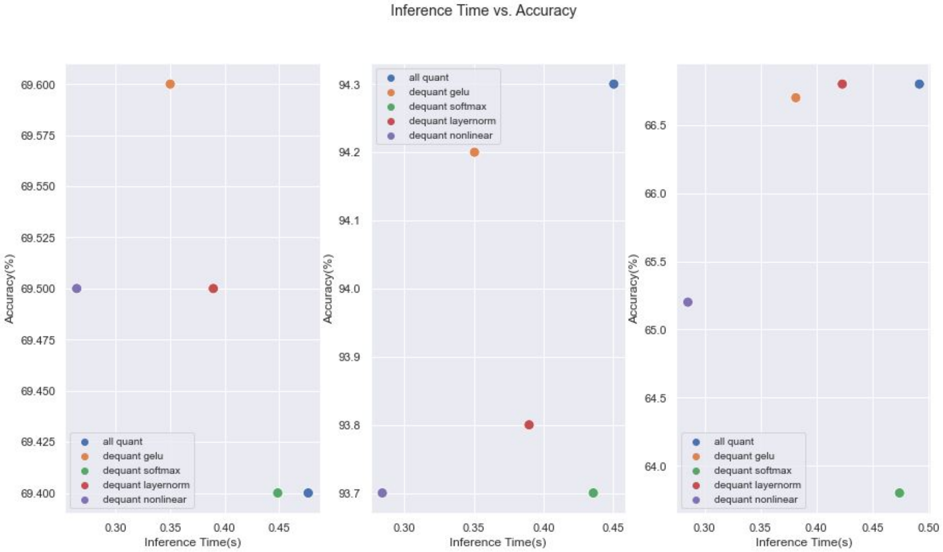


Experimental Evaluation:

Trade-offs of different dequant options

Trade off between accuracy and training time

Trade off between accuracy and inference time



From the plot, we can discover that end-to-end Quant always needs more training time and lower accuracy than base. Dequant all nonlinear layer have less training but relatively low accuracy.

Dequantizing GELU operation seems to be a good trade-off.

Conclusion: (Github repo: https://github.com/Maggieli99/RoBERTA_quantization)

- When training the tasks with very large dataset (e.g., MNLI, SST-2), quantized RoBERTa can achieve very similar accuracy as compared to the full-precision baseline. For smaller datasets (e.g., CoLA, RTE), the degradation in accuracy is more significant.
- **Quantization-aware training leads to larger training time** because it requires multiple forward and backward passes through the network with different quantization levels; in general, larger datasets leads to larger training time difference between quantized and base models
- Since Pytorch does not support integer operation, the inference times we get have significant bias. So inference on other computing units is needed.
- Only **dequantizing GELU operation** (Quantize linear+Softmax+LayerNorm) seems to be a good trade-off. In this case, it achieves a relatively high accuracy compared to other quantization combinations while having a moderate training time.

Future Work:

- Pytorch does not support integer operation so in order to deploy integer-quantized model on GPU or CPU and achieve speedup during inference, we need export the integer parameters along with the model architecture to other frameworks that support deployment on integer processing units(TVM and TensorRT).
- Manually computing model size base is needed to further evaluate efficiency of quantization.

Thank You!

PHOTO: DARCY PETERKA