<PROJECT 최종보고서>

거대 딥러닝 언어 모델을 위한

Low-Precision & Quantization

이하린 이철희 서희수

신영길 교수님

Table of Contents

1. Abstract

본 프로젝트에서는 OpenAI 에서 2019 년에 발표한 거대 언어 모델인 GPT-2 에 최신의 low-precison & quantization 기법을 적용하여 모델의 경량화 가능성에 대해 살펴본다. 라이브러리 차원의 정수 자료형 계산 가속 기능의 부재로 수행 시간의 이점은 크게 이끌어내지 못했으나, LAMBADA dataset 벤치마크 결과를 모델 성능 측정의 지표로 활용하여 1%p 미만의 성능 차이로 GPT-2 small 은 기존 대비 35%, GPT-2 Extra Large 는 기존 대비 22%의 크기를 갖도록 모델 경량화 작업에 성공했다. 본 보고서에서는 GPU 에서 지원하지는 않는 자료형들에 대한 경량화 모델의 성능을 추정하기 위해 사용한 bit-bandwidth 조절 fake-quantization 기법 및 INT8/4 quantization, binary coding quantization 등 모델 경량화 작업에 사용된 기법들을 기술하였다.

2. Introduction

거대 모델들이 필요로 하는 많은 양의 전산 자원과 높은 연산 능력, 긴 학습과 추론 시간은 스마트폰 및 자동차, 로봇, 사물 인터넷 등의 임베디드 시스템의 제한된 환경에서의 기술 적용을 어렵게 하는 요인이 된다. 이러한 환경에서도 딥러닝 기술을 사용하기 위해 모델의 경량화를 통해 적은 용량으로 빠른 처리를 가능하게 하는 quantization 기술들이 최근 많이 연구되고 있다.

GPT-2는 2019년 OpenAI에서 발표한 자연어 처리 모델로, 15억개의 파라미터를 가진 거대 모델의 대표적인 예이다. 당시에는 자연어 처리 모델 중 가장 크고 가장 뛰어난 성능을 달성한 모델이었다. 기존 GPT 모델에서 큰 구조 변화 없이 파라미터의 수만 약 10배 증가시켰을 때에 기존의 모델들을 뛰어넘는 성능을 보여주며, 이 분야에서 모델의 크기를 키우는 것이 곧 성능의 증가로 이어질 수 있다는 사실을 직접적으로 보여주었다. 이후로 점점 더 큰 모델들이 등장하며 더욱 강력한 성능을 보여주고 있는 한편, 거대 모델들이 실용적으로 사용하는 것이 불가능한 테라 바이트 수준으로 커지며 정도에 이르자 거대 언어 모델들의 성능을 유지하며 크기를 줄이는 연구 또한 활발히 진행되고 있다.

모델 경량화의 방법으로는 여러 가지가 존재하는데, low-precision과 quantization은 연산에 사용하는 자료형을 바꾸어 모델의 크기와 추론 시간을 줄이는 기술이다. 일반적으로 딥러닝 모델들은 부동소수점 32비트 실수 자료형(FP32)을 사용하는데 low-precision의 경우 정밀도가 조금 떨어지는 부동소수점 16비트 실수 자료형(FP16)등을 사용하고, quantization의 경우 이산적인 8비트 정수 자료형(INT8)과 같은 자료형을 사용해 FP32 연산을 대체한다. 이 때 FP32보다 표현 능력이 떨어지는 자료형을 사용함에 따라 생기는 성능 저하를 최소화하는 방법이 주된 관심사가 되며, 성능 저하를 줄이기 위한 새로운 자료형을 개발하는 연구 역시 진행된다.

이 프로젝트의 목표는 GPT-2에 low-precision과 quantization 기술을 적용해 GPT-2의 성능을 유지하며 크기를 줄이는 것이다. GPT-2의 구조와 다양한 low-precision, quantization 기술에 대한 이해를 바탕으로 어떤 방법으로 GPT-2의 경량화를 이루어내는 것이 적합한지 판단한 후, 경량화 기술이 적용된 모델의 성능과 크기를 비교할 것이다.

3. Background Study

A. GPT-2

GPT-2는 2019년 OpenAI의 Language Models are Unsupervised Multitask Learners (Radford, et al)[1] 논문에서 발표된 자연어 처리 모델이다. 이 모델은 당시 여러 자연어 처리 벤치마크에서 최고 성능을 기록하였으며, 언어 모델이 fine-tuning 없이 바로 여러 태스크에 적용될 수 있다는 점과, 모델의 크기를 키울수록 더욱 뛰어난 성능을 얻을 수 있다는 점을 보여주었다.

GPT-2는 인터넷에서 수집된 광범위한 양의 글인 WebText를 대상으로 일정 양의 단어들 직후에 올 단어를 예측하도록 훈련되었다. 그 결과 GPT-2는 단어 예측에서 뛰어난 성능을 보인 것은 물론이고 예측을 반복해 높은 수준의 글을 쓸 수 있다는 점에서 큰 화제가 되었었다. 또한 단어 예측뿐만 아니라 한 번도 훈련된 적 없는 기계 번역, 질문에 대한 답변, 문단 요약 등의 태스크에서도 훌륭한 성능을 보였다. 기존의 언어 모델들은 지도 학습을 통해 fine-tuning을 거처야만 각 태스크에 적용할 수 있었지만, GPT-2는 fine-tuning을 하는 대신 입력에 무엇을 출력해야 하는지 구체적으로 명시함으로써 여러 태스크를 훈련 없이 수행하는 zero-shot learning에 성공하였다. 이를 통해 언어 모델은 각 용도에 맞는 별개의 모델들이 필요한 것이 아니라, 다양한 데이터로부터 훈련되어 언어의 특성과 용법을 상황에 맞게 잘 분석할 수 있는 모델만 있으면 충분하다는 것을 보였다.

GPT-2의 큰 특징 중 또 다른 하나는 크기이다. OpenAI에서는 4가지 크기의 GPT-2 모델을 제시하고 각각의 성능을 비교하였는데, 크기가 커질수록 성능이 확연히 좋아지는 것을 확인할 수 있다. 특히 가장 큰 모델인 GPT-2 Extra-Large는 15억개의 파라미터를 가지고 있는데, 논문에서 실험한 8개의 벤치마크 중 7개에서 당시 최고 성능을 기록하였다. 이를 통해 당시 자연어 처리 분야에서 모델의 크기를 키우는 것이 성능으로 직결된다는 것이 증명되었으며, 그 이후로는 점점 더 큰 모델을 통해 더더욱 뛰어난 성능을 달성하는 연구가 계속해서 진행되고 있다.

GPT-2는 자연어 처리 모델인 Transformer decoder의 구조를 거의 그대로 사용한다. Transformer는 2017년 Attention is All You Need (Vaswani, et al)[2] 논문에서 제시된 자연어 처리 모델로, 기존

의 자연어 처리 모델들인 RNN, LSTM과 다르게 attention mechanism만을 사용하여 뛰어난 성능을 보여주었다. Attention mechanism은 문단 내의 두 토큰 사이의 관계의 정도를 계산하여 토큰의 의미를 다르게 해석하는 방법이다. Transformer 구조에 대한 자세한 분석은 이 보고서의 범위를 넘어가며, GPT-2에서 보고서의 이해에 필요한 부분만 간략히 소개한다.

GPT-2의 구조는 크게 Input embedding, GPT2Block, output probabilities의 세 부분으로 나눌 수 있다. Input embedding에서는 입력 문장을 토큰 단위로 나눈 후, 각 토큰과 위치에 미리 배정된 임베딩 벡터를 대응시켜 시작 activation을 구성한다. GPT-2는 한 글자 단위 토큰부터 시작해 많이 등장하는 토큰 묶음을 다시 하나의 긴 토큰으로 정의하는 BPE(Byte-Pair Encoding) 토큰화 방식을 사용했으며, 훈련 데이터로부터 50,257 종류의 토큰을 미리 정의하고 훈련에 사용했다. 시작 activation이 구성된 후 주요한 계산은 GPT2Block을 여러 개를 쌓아 올린 부분에서 진행된다. 내부적으로 Layer normalizations, Linear layers, Softmax, Residual connections, Matrix multiplication 등 여러 연산이 사용되는데, 대부분의 파라미터는 linear layers 에 모여있다. 한 개의 GPT2Block은 4개의 linear layers를 가지며, GPT-2 Small의 경우 12개의 블록, Medium은 24개, Large는 36개의 블록으로 구성되어 있다. GPT-2 Extra-Large는 48개의 블록을 사용해 총 192개의 linear layers로 구성되어 있고 약 94%의 파라미터들이 이 부분에 저장되어 있다. 마지막으로 output probabilities 부분은 출력 벡터로부터 50,257 종류의 토큰에 각각 확률을 부여하는 한 개의 linear layer로 이루어져 있다.

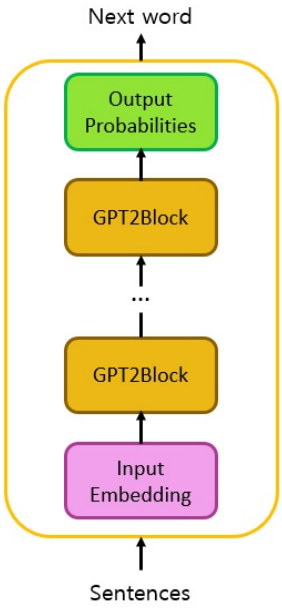

그림 1. 단순화된 GPT-2의 구조

B. Low-Precision

Low-precision은 비트 수가 더 적은 대신 정밀도가 더 낮은 자료형을 사용하여 모델의 경량화를 달성하는 방법이다. 가장 대표적으로 FP32 대신 half-precision, 혹은 FP16이라고 불리는 16비트 실수 자료형을 사용하는 방법이 있다. 그러나 FP32보다 지수 부분의 비트 수가 적은 FP16은 표현할 수 있는 크기가 작아져 딥러닝 훈련 시 비교적 좋지 않은 성능을 보여주었고, 이를 극복하기 위해 BFLOAT16이라는 자료형이 개발되었다. 지수 부의 크기에 민감한 신경망을 위해 개발된 BFLOAT16은 FP16 자료형 보다 지수부가 3비트 더 많은 커스텀 16비트 자료형으로, 구글의 TPU와 대부분의 GPU에서 연산을 지원하고 있으며, FP16보다 BFLOAT16를 사용하는 것이 딥러닝 훈련이 더 잘 된다는 연구가 있다[3]. 같은 맥락에서 만들어진 TF32(TensorFloat-32)는 FP32와 같은 크기의 지수부를 사용하지만 만티사 부분을 절반 정도만 사용하여 정밀도는 떨어지지만 더욱 빠른 연산을 가능하게 했으며, NVIDIA A100 GPU에서 연산을 지원한다[4].

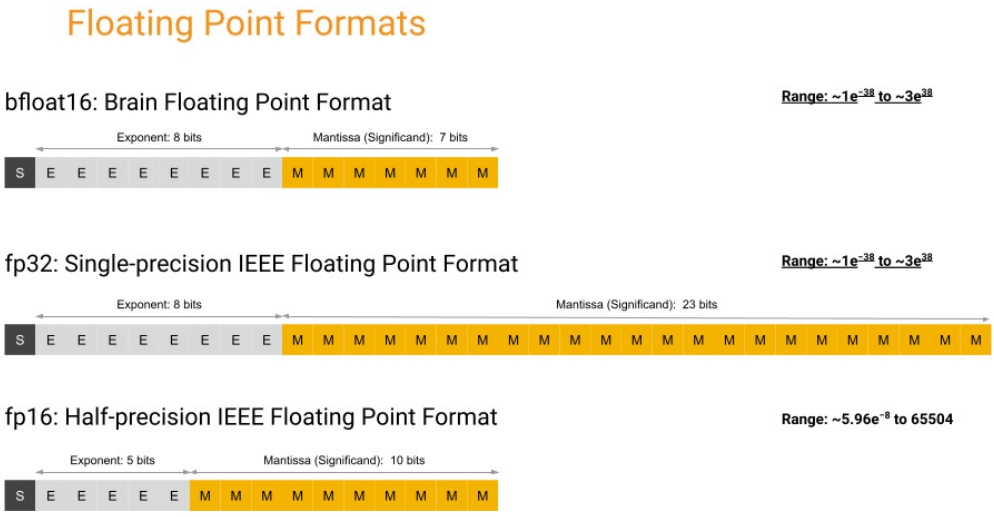

그림 2. BFLOAT16, FP32, FP16 자료형의 비트 비교

C. Quantization

Quantization은 실수 자료형 대신 정수 자료형 등의 이산적인 값을 사용해 모델의 경량화를 달성하는 방법이다. 일반적으로는 INT8이나 INT4 등의 자료형을 사용하는데, XNOR-net의 경우 이미지 처리 분야에서 weight를 극단적으로 1비트까지 줄일 수 있다는 것을 보여주었으며[5], DoReFa-

net의 경우에는 activation과 gradient까지도 적은 비트 수의 quantization을 적용해 훈련시킬 수 있다는 것을 보여주었다[6].

가장 기본적인 quantization 방법으로는 FP32 값을 선형 변환 후 반올림을 통해 INT8로 만드는 방법이 있다. 이 경우 자료형 변환이 단순하고 quantization이 적용된 행렬 간의 연산도 쉽게 가능하다는 장점이 있다. 정수 자료형이 아닌 1비트 quantization을 사용하는 XNOR-net은 weight를 +1, -1 두 가지 값으로 대응시켰으며 채널 단위로 적용되는 추가적인 scaling factor E를 사용했다. Weight가 +1, -1 두 가지 값만 가지므로 실질적으로 곱셈 없이 덧셈, 뺄셈만으로 계산을 처리할 수 있다는 장점이 있다. Binary coding quantization은 XNOR-net의 아이디어를 여러 비트를 사용할 수 있게 발전시킨 방법이다. 행렬을 적은 비트 수로 근사할 때에 scaling factor와 +1, -1로 이루어진 행렬을 여러 개 사용하여 1비트 이상의 quantization도 가능하게 하는 방법이다. 이 경우 정수 자료형을 사용하는 것과는 다르게 quantization 결과값이 non-uniform한 값들로 이산화되며, 비트 수가 자료형에 구애 받지 않는다는 장점이 있다.

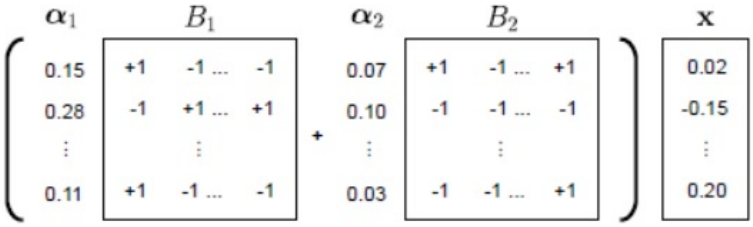

그림 3. 2비트 binary coding quantization의 한 예[7]

Quantization은 시점에 따라 post training quantization(PTQ)와 quantization aware training(QAT)의 두 가지 방식이 존재한다. PTQ 방식은 FP32로 훈련이 완료된 모델에 quantization을 적용하는 방식이다. 이 때 발생하는 오차를 줄이기 위해 비교적 적은 양의 calibration set으로 추가적인 훈련을 시키기도 한다. PTQ 방식은 QAT 방식보다 빠르지만 quantize된 모델의 성능이 더 떨어진다는 단점이 있다. QAT 방식은 quantization이 적용된 상태로 훈련을 하는 방식이다. 훈련 시에는 fake quantization이라는 방법 사용하는데, quantization을 적용하였을 때 가지게 될 값을 FP32 자료형을 사용해 simulation하여 사용하고, 훈련이 끝난 후 실수 자료형을 정수 자료형으로 대체하는 방법이다. Back propagation 시의 gradient와의 연산이 FP32를 사용하기 때문에 높은 정밀도로 훈련을 할 수 있고 오차가 이미 고려된 상태이기 때문에 더 좋은 성능을 얻을 수 있다는 장점이 있지만, 시간이 오래 걸린다는 단점이 있다. 여기서 Fake quantization은 QAT 훈련뿐만 아니라 아직 하드웨어적 지원이 되지 않거나 구현이 복잡한 quantization 방법들의 성능을 미리 확인하기 위해서도 많이 사용된다. 초기 Transformer-based generative Pretrained Language Model(PLN) 과 같이 크기가 큰 모델들의 단순 경량화에서 시작해서, 최근에는 transformer등과 같은 모델 내부의 중간

연산 결과와 구조적 이해를 바탕으로 성능 하락을 최소화하는 model wise Quantization들이 연구되고 있다.

4. Goal/Problem & Requirements

이 프로젝트의 목표는 기존 GPT-2 모델에 low-precision과 quantization 기술을 적용해 크기를 줄이되, 성능의 감소는 최소화하는 것이다. 경량화 된 모델의 성능은 LAMBADA dataset을 활용해 측정한다. 초기 목표는 가장 크기가 작은 GPT-2 Small 기준 크기를 50% 이상 줄이고 정확도 40% 이상, PPL 40 이하를 얻는 것이 목표였으나, 이는 쉽게 달성되어 GPT-2 Small과 GPT-2 Extra-Large에 대해 정확도 차이 2%p 내외로 모델의 크기를 최대한 줄이는 것을 목표로 하였다.

5. Approach

A. Benchmark

모델을 경량화 했을 때에 달라지는 성능 측정의 지표로 자연어 처리 분야의 벤치마크 중 하나인 LAMBADA dataset을 선택하였다. LAMBADA dataset은 여러 문장이 주어졌을 때 가장 마지막 문장의 마지막 단어를 모델이 정확히 예측할 수 있는지 테스트하는 벤치마크이다. 정확도(ACC)는 테스트 케이스들에 대해 기계가 정답을 맞춘 비율로 정의되고, perplexity(PPL)은 모델이 다음에 올 정답 토큰들에 대해 배정한 확률들의 역수의 기하 평균으로 정의된다. ACC 수치는 높을수록, PPL 수치는 낮을수록 자연어 처리 성능이 뛰어남을 의미한다.

LAMBADA dataset은 공백, 대소문자 등의 전처리가 다른 Huggingface dataset 라이브러리의 버전과 OpenAI에서 제공하는 버전 두 가지가 존재한다. GPT-2 논문에서 구체적으로 어느 dataset에 대해 어떻게 테스트했는지 밝히지 않아 추가적인 조사가 필요했다. 그 결과 OpenAI의 데이터셋을 사용하고 마지막 단어가 아닌 마지막 BPE 토큰의 정답 여부를 비교해야 논문과 같은 정확도가 나오는 것을 확인하였고, 이 보고서의 모든 실험은 논문과 동일한 설정에서 진행되었다. PPL 수치는 논문의 수치보다 더 성능이 좋게 나왔는데, 각 테스트케이스에 대해 모델이 가장 마지막 정답 BPE 토큰에 배정한 확률의 역수들의 기하평균으로 계산했다.

	LAMBADA (PPL)	LAMBADA (ACC)
SOTA	99.8	59.23
117M	**35.13**	45.99
345M	**15.60**	55.48
762M	**10.87**	**60.12**
1542M	**8.63**	**63.24**

그림 4. 네 가지 크기의 GPT-2 모델에 대한 LAMBADA dataset 벤치마크 결과[1]

B. Low-Precision and Quantization Methods

거대 언어 모델들의 훈련 시간이 매우 길다는 것을 고려했을 때, QAT 방식보다는 PTQ 방식을 시도하는 것이 이번 프로젝트에 더 적합하다고 판단하였다. Pre-train된 파라미터들에 바로 quantization을 적용한 후 추가적인 훈련 없이 바로 성능을 확인하는 방식으로 진행되었다. 또 Pytorch에서 빠른 정수 행렬 연산이 지원이 되지 않는 관계로(7.B Implementation Issues 참조) 정수형 연산을 통해 시간적 단축을 목표로 하는 activation quantization은 진행하지 않고 FP16으로 바꾸는 low-precision만 적용하였다. Linear layers의 weight 를 제외한 나머지 파라미터들에도 역시 FP32에서 FP16으로의 low-precision을 적용하였다. 그 후 linear layers의 weight에는 추가적인 quantization을 진행했는데, INT8/4 quantization과 binary coding quantization의 별개의 두 가지 방법을 각각 적용하고 테스트하였다.

INT8/4 quantization에서는 원래의 FP32 행렬을 INT8 또는 INT4 자료형을 사용하는 정수형 행렬과 행 단위의 FP16 scaling factor들을 통해 근사하였다. Scaling factor는 행에서 절댓값이 가장 큰 원소가 정수 자료형에서도 절댓값이 가장 큰 값 (INT8의 경우 -128, INT4의 경우 -8)에 대응되게 결정하였고, 정수형 행렬은 원소를 scaling factor로 나눈 후 자료형의 범위로 clip한 후 반올림을 하여 결정하였다.

Binary coding quantization은 FP32 행렬을 +1, -1로만 이루어진 행렬들과 행마다의 FP16 scaling factor를 통해 근사하였다. Scaling factor와 +1, -1 행렬은 한 번에 하나씩 순서대로 남은 근사치의 특정 계산을 최소화하는 greedy approximation을 사용했다.

$$W_0 = W, W_p = W - \sum_{i=1}^{p} \alpha_i B_i$$

$$\alpha_p, B_p = \underset{\alpha_p, B_p}{\operatorname{argmin}} \left\| W_{p-1} - \alpha_p B_p \right\|^2 \ (1 \le p \le q)$$

그림 5. l2-norm을 최소화하는 greedy approximation

원래 행렬을 근사하기 위한 계산식으로 일반적으로는 MSE(Mean Squared Error) 혹은 l2-norm, 즉 각 원소의 오차의 제곱의 평균을 최소화하는 방식이 많이 채택되었고, 그것을 달성하기 위한 휴리스틱도 많이 연구되고 있다[7]. 그러나 반드시 해당 MSE를 최소화해야 할 이유는 없다고 판단하여 supnorm, 즉 오차의 절댓값의 최대값을 최소화하는 방식을 새로 제안하고 사용해보았다. 이 보고서에서는 l2-norm을 최소화하는 방식을 l2 방법, supnorm을 최소화하는 방식을 sup 방법이라고 지칭한다. l2 방법과 sup 방법 모두 한 개의 scaling factors와 +1, -1 행렬을 결정할 때에는 근사치가 아닌 정확한 해가 구해지기 때문에 greedy approximation을 사용하기 좋다는 장점이 있다.

6. Methods

A. Experiment Details

Huggingface의 Transformer 라이브러리에서 다운받을 수 있는 pre-trained GPT-2 Small, GPT-2 Extra-Large 모델 두 가지에 대해 동일한 실험을 진행했다. FP32를 사용하는 기존 모델과 모든 파라미터들을 FP16으로 변환한 모델에 대해 먼저 벤치마크 결과를 측정했다. 이후 FP16 모델의 linear layers의 weight에 추가적인 quantization을 적용하였다. Quantization 방법으로는 INT8/4 quantization과 binary coding quantization을 각각 적용하고 테스트하였다.

INT8/4 quantization은 INT8을 사용해 quantization 했을 때와 INT4를 사용해 quantization을 했을 때의 성능을 테스트하였다. 추가로 최종 단어 선택 linear layer가 quantization에 민감한 정도를 확인하기 위해 다른 linear layers와 함께 마지막 linear layer에도 INT8 quantization을 적용했을 때의 성능 역시 테스트하였다. 그 후 여러 블록의 각 레이어에 INT8 자료형과 INT4 자료형을 섞어서 quantization하고 성능을 테스트하며 정밀도에 더 민감한 레이어와 아닌 레이어를 구분해 최소한의 성능 저하로 최대한의 경량화를 이룰 수 있는 지점을 찾으려 하였다.

Binary coding quantization은 8, 6, 4비트를 사용한 sup 방법과 l2 방법 quantization을 진행하고 결과를 테스트하였다. 그 후 GPT-2 Extra-Large에 대해서 몇 가지 실험을 더 진행하였다. 먼저 2비트 sup 방법과 l2 방법을 추가적으로 비교했으며, 8, 6비트 quantization에서 scaling factors의 결정

시 greedy approximation으로 최소화하는 계산식을 일부는 supnorm을, 나머지는 l2-norm으로 섞어서 선택했을 때의 결과를 테스트하였다.

벤치마크로는 OpenAI의 LAMBADA dataset을 사용했으며, 32개 texts를 한 개의 배치로 묶어 테스트하였다. 각 모델이 주어진 문장들의 마지막 BPE 토큰을 예측하게 하고, 그 때의 accuracy와 perplexity를 기록하였다.

B. Development Environment

RTX 3090 GPU 4개가 사용 가능한 Ubuntu 20.04.4 LTS 연구실 서버에서 실험이 진행되었다. PyTorch 머신러닝 라이브러리를 사용했으며, Huggingface의 Transformer 라이브러리의 GPT-2 모델과 pre-trained 파라미터를 사용했다. 사용한 라이브러리들의 버전은 Transformer 4.4.2, Tokenizer 0.10.3, PyTorch 1.12.1, CUDA 11.3이다.

7. Solution

A. Implementations Details

GPT-2의 linear layer 모듈과 동일한 인터페이스를 갖는 quantized linear layer 모듈 클래스를 작성하고 GPT-2 모델 객체의 linear layer 객체들을 새로 만든 모듈의 객체들로 대체하였다.

INT8/4 quantization에 사용한 quantized linear layer 모듈은 초기화 시에는 대체할 linear layer 객체와 사용할 비트 수를 입력 받아 scaling factors와 INT8 행렬을 계산해 저장한다. forward 함수에서는 INT8 행렬을 FP16으로 변환해(7.B. implementations Issues 참조) 입력 텐서와 행렬 곱을 한 후 scaling factors를 elementwise 곱셈 후 반환한다.

Binary coding quantization에 사용한 모듈은 대체할 linear layer 객체와 사용할 비트 수, scaling factors를 결정할 방법을 입력으로 받아 scaling factors와 +1, -1 행렬을 계산한 후 행렬을 +1을 1, -1을 0으로 대응시키고 한 열 내에서 8개의 원소 씩 묶어 INT8 행렬에 저장한다. forward 함수에서는 INT8 행렬을 다시 +1, -1 행렬로 되돌리고 scaling factors를 각 행렬에 곱한 후 모두 더한 하나의 FP16 행렬을 구성해 입력 텐서와의 행렬 곱을 반환한다(7.B. implementations Issues 참조).

B. Implementations Issues

표 1. Pytorch에서 자료형에 따른 1000x1000x1000 크기의 두 텐서의 matmul 연산 지원 여부와 소요 시간.

시간(s)	FP32	FP16	INT32	INT8
CPU	1.92	지원 x	832.79	55.25
GPU	0.001 미만	0.001 미만	지원 x	지원 x

먼저 quantization은 정수 연산을 통해 시간적 단축을 이루어내야 하지만, Pytorch에서는 아직 정수 자료형의 행렬 연산이 제대로 지원되지 않는다는 문제가 있다. CPU의 경우 정수 연산이 FP32 연산보다 더 느리며, GPU의 경우 정수 연산이 아예 지원이 되지 않는다. 그리고 서로 다른 자료형의 행렬 간의 곱 연산은 전혀 지원되지 않는다. 그런 문제로 weight quantization을 통한 크기 압축만 실험할 수 있었으며 activation quantization을 통한 시간 단축은 실험하는 것이 불가능하다고 판단하여 진행하지 않았다. 모델 내의 계산에도 정수 자료형을 모두 FP16으로 변환하여 사용하는 fake quantization 방식으로 진행해야 했다. 또한 Pytorch에서는 INT4 자료형을 지원하지 않기 때문에 INT8 자료형에 INT4 범위의 값을 저장한 후, 크기 계산에는 INT4 자료형이 사용되었을 때의 이론적인 크기를 계산하였다.

Binary coding quantization의 경우 반복되는 덧셈/뺄셈을 lookup table을 활용해 연산의 수를 줄이는 방법이 연구되었다[8]. 이 방법을 구현하기 위해서는 먼저 table을 만들고 만들어진 table의 특정 인덱스들의 값의 합을 구하는 연산이 필요한데, 이를 직접적으로 지원하는 Pytorch 연산은 없었고, 특정 인덱스의 값들을 모으는 부분과 합을 구하는 부분을 따로 구현하자 시간이 오히려 더 오래 걸린다는 단점이 있었다. 여러 방법을 시도해봤을 때 현재 Pytorch 버전과 주어진 GPU에서는 scaling factors와 비트 행렬로부터 원래의 행렬을 FP16으로 재구성해 입력과 한 번의 행렬 곱을 하는 것이 가장 빠른 것으로 나타났다. 이는 행렬 곱에 여러 최적화가 적용되는 점, Pytorch 텐서를 인덱스 별로 접근하는 것이 비교적 느린 점, 그리고 lookup table은 입력에 따라 다른 table이 만들어지지만 행렬을 재구성하는 것은 항상 동일한 연산을 하기 때문에 caching등이 가능한 점 등에 의한 것으로 추측된다.

시간적 단축을 위한 연산들이 현재 Pytorch 버전의 지원되지 않아 일부 구현에서는 이론상 더 비효율적인 방법을 택해야 했지만, 필요한 연산이 지원되는 환경에서는 시간 단축도 크게 이루어낼 수 있을 것으로 예측된다.

8. Results

A. INT8/4 Quantization

표 2. GPT-2 Small 기본 모델과 low-precision, INT8/4 quantization이 적용된 모델들의 LAMBADA dataset 벤치마크 결과와 모델 크기, 실행 시간을 비교. INT8+Last는 output layer도 INT8 quantization을 적용했을 때, INT8+INT4는 GPT-2의 첫 두 블록과 마지막 두 블록에는 INT8, 나머지 블록에는 네 개의 linear layers에는 각각 INT4, INT4, INT4, INT8 quantization을 적용했을 때의 결과이다.

Small	ACC(%)	PPL	SIZE(MB)	크기 비(%)	시간(s)
FP32	46.67	17.56	633.94	100.0	13.11
FP16	46.28	17.63	322.97	50.9	6.83
INT8	46.30	17.48	242.13	38.2	8.42
INT8+Last	44.93	inf	205.41	32.4	7.21
INT4	40.68	29.69	201.63	31.8	7.01
INT8+INT4	45.35	20.61	224.13	35.3	8.40

표 3. GPT-2 Small 기본 모델과 low-precision, INT8/4 quantization이 적용된 모델들의 LAMBADA dataset 벤치마크 결과와 크기, 실행 시간을 비교. INT8+Last는 output layer도 INT8 quantization을 적용했을 때, INT8+INT4는 GPT-2의 첫 두 블록과 마지막 네 블록에는 INT8, 나머지 블록에는 네 개의 linear layers를 각각 INT4, INT4, 에는 INT4, INT8 quantization을 적용했을 때의 결과이다.

Extra Large	ACC(%)	PPL	SIZE(MB)	크기 비(%)	시간(s)
FP32	61.42	6.18	6296.56	100	128.35
FP16	61.34	6.19	3172.28	50.4	46.16
INT8	61.30	6.19	1767.34	28.1	50.79
INT8+Last	61.30	6.20	1690.75	26.9	50.28
INT4	58.87	7.22	1064.22	16.9	49.77
INT8+INT4	61.26	6.35	1357.19	21.6	50.10

FP32를 사용하는 기존 모델과 FP16 low-precision, INT8/4 quantization이 순차적으로 적용된 모델들의 LAMBADA dataset 벤치마크 결과를 비교하였다. FP16 low-precision과 INT8 quantization까지 적용된 모델은 GPT-2 Small과 GPT-2 Extra-Large 둘 다 성능의 하락이 매우 근소한 것을 볼 수 있

다. 마지막 output probability linear layer까지 quantization을 시도했을 때는 GPT-2 Small의 경우 크기가 많이 줄었지만 큰 성능 하락을 보인 반면 GPT-2 Extra-Large는 크기가 많이 줄지 않고 성능이 거의 떨어지지 않은 것을 볼 수 있다. 일반적으로 입력, 출력에 가까운 레이어는 quantization 시 크기 압축 정도에 비해 성능에 많은 영향을 미친다는 것이 알려져 있어[6, 9], output layer의 quantization은 진행하지 않는 것이 더 낫다고 판단하였다. INT4 quantization까지 진행했을 때에는 성능이 비교적 크게 떨어지는 것을 볼 수 있다. 이 때 전체적으로 GPT-2 Small에 비해 GPT-2 Extra-Large가 적은 비트 수를 사용함에 따라 성능이 떨어지는 정도가 더 적은 것을 확인할 수 있고, 모델의 크기가 클수록 quantization에 의한 오차에 덜 민감하다는 결론을 얻을 수 있다.

마지막으로 GPT-2의 각 linear layer에 INT8 quantization과 INT4 quantization을 섞어서 성능을 측정해본 결과 블록 내의 네 개의 linear layers 중 마지막 레이어가 quantization 시 성능 하락이 가장 큰 것을 확인하였고, 입력과 출력에 가까운 block들이 특히 성능에 큰 영향을 미친다는 것을 확인하였다. GPT-2 Small의 경우 첫 두 블록과 마지막 두 블록, 그리고 나머지 블록들의 마지막 linear layers에 INT8 quantization을 적용하고 나머지 linear layers에 INT4 quantization을 적용했을 때 정확도 약 1.3%p 저하와 크기 비 35.3%를 달성할 수 있었고, GPT-2 Extra-Large의 경우 앞과 동일하지만 마지막에서 세 번째, 네 번째 블록에 INT4 대신 INT8 quantization을 적용했을 때 정확도 저하 약 0.2%p 저하와 크기 비 21.6%를 달성할 수 있었다.

B. Binary Coding Quantization

표 4. GPT-2 Small 기본 모델과 low-precision, binary coding quantization이 적용된 모델들의 LAMBADA dataset 벤치마크 결과와 크기, 실행 시간을 비교.

Small	ACC(%)	PPL	SIZE(MB)	크기 비(%)	시간(s)
FP32	46.67	17.58	633.93	100.0	13.10
FP16	46.28	17.63	322.97	50.9	6.83
8 bits(sup)	45.99	17.91	243.23	38.4	11.96
6 bits(sup)	45.62	18.11	222.67	35.1	10.59
4 bits(sup)	37.65	51.59	202.10	31.9	9.71
8 bits(l2)	28.95	inf	243.23	38.4	11.33
6 bits(l2)	25.67	inf	222.67	35.1	11.14
4 bits(l2)	17.68	inf	202.10	31.9	9.72

표 5. GPT-2 Extra-Large 기본 모델과 low-precision, binary coding quantization이 적용된 모델들의 LAMBADA dataset 벤치마크 결과와 크기, 실행 시간을 비교.

Extra Large	ACC(%)	PPL	SIZE(MB)	크기 비(%)	시간(s)
FP32	61.42	6.17	6296.56	100.0	128.35
FP16	61.34	6.18	3172.28	50.4	46.16
8 bits(sup)	61.24	6.19	1776.57	38.4	117.48
6 bits(sup)	60.85	6.28	1422.37	22.6	104.03
4 bits(sup)	49.52	13.64	1068.17	16.96	87.70
2 bits(sup)	0.09	inf	713.97	11.34	73.26
8 bits(l2)	60.45	6.44	1776.57	38.4	116.92
6 bits(l2)	59.69	6.74	1422.37	22.6	102.89
4 bits(l2)	57.75	7.71	1068.17	17.0	87.73
2 bits(l2)	15.23	inf	713.97	11.34	73.24

FP32를 사용하는 기존 모델과 FP16 low-precision, binary coding quantization이 순차적으로 적용된 모델들의 LAMBADA dataset 벤치마크 결과를 비교하였다. GPT-2 Small의 경우 sup 방법 6비트까지 성능 저하가 적은 것을 볼 수 있고, sup 방법 4비트는 비교적 성능 저하가 큰 것을 볼 수 있다. l2 방법은 항상 sup 방법보다 안 좋은 결과를 내는 것을 볼 수 있다. GPT-2 Extra-Large의 경우 sup 방법 6비트까지는 성능 저하가 적고, sup 방법 4비트부터는 성능이 크게 떨어지는 것을 확인할 수 있다. 반면 l2 방법의 경우 6비트까지는 sup 방법보다는 성능 저하가 크지만 4비트, 2비트에서는 sup 방법보다 훨씬 좋은 결과를 얻는 것을 볼 수 있다.

표 6. GPT-2 Extra-Large에 다양한 scaling factor 결정 방법을 사용해 binary coding quantization이 적용된 모델들의 LAMBADA dataset 벤치마크 결과 비교. 1-4 sup + 5-8 l2는 1-4번째 비트의 scaling factors는 supnorm을 최소화하도록 greedy하게 결정되었고, 5-8번째 비트의 scaling factors는 l2-norm을 최소화하도록 greedy하게 결정되었다는 뜻이며 나머지도 같은 방식으로 해석하면 된다.

Extra Large	ACC(%)	PPL
1-8 sup	61.24	6.19
1-8 l2	60.45	6.44
1-4 sup + 5-8 l2	61.28	6.19
1-4 l2 + 5-8 sup	61.23	6.30
1-6 sup	60.85	6.28

1-6 l2	59.69	6.74
1-4 sup + 5-6 l2	60.93	6.27
1-3 sup + 4-6 l2	60.83	6.36
1-3 l2 + 4-6 sup	55.93	8.47

Scaling factors를 결정하는 계산식을 비트마다 다르게 적용하였을 때의 결과도 관찰하였다. 일반적으로 앞쪽의 비트들을 sup 방식으로 scaling factor를 결정하고 뒤쪽의 비트들을 l2 방식으로 결정하는 것이 근소하게 더 좋은 결과를 내는 것을 확인할 수 있고, 반대로 l2 방식을 먼저 적용한 후 sup 방법을 적용하는 것은 오히려 더 안 좋은 결과를 내는 것을 확인할 수 있다. 이는 l2 방법이 남은 비트 수가 적을 때에 더 좋은 성능을 보여주는 것으로 해석할 수 있다. 첫 네 비트는 sup 방법, 그 후 두 비트는 l2 방법을 사용하는 6비트 quantization은 정확도 저하 0.5%p와 크기비 22.6%를 달성할 수 있었다.

9. Division & Assignment of Work

표 7. Division & Assignment of Work

항목	담당자
선행 연구 조사	이철희, 이하린, 서희수
기존 Low-precision/Quantization 기술 적용	이철희, 이하린, 서희수
자체적 Low-precision/Quantization 기술 적용	이철희, 이하린, 서희수
발표 자료 및 보고서 작성	이철희, 이하린, 서희수
Demo 코드 작성	이철희, 이하린, 서희수

10. Conclusion

이 프로젝트에서는 GPT-2에 적용할 수 있는 low-precision과 quantization을 탐구하였다. LAMBADA dataset을 성능 측정의 지표로 삼았고, INT8/4 quantization과 binary coding quantization을 적용하여 벤치마크 결과가 거의 유지되는 상태로 크기를 대폭 줄일 수 있다는 것을 보였다. GPT-2 Extra-Large 기준 INT8과 INT4 자료형을 적절히 사용하거나 특정한 방식으로 scaling factors를 결정한 6비트 binary coding quantization을 통해 정확도 0.2%p~0.5%p 차이로 기존 대비 약

21~23%의 크기를 갖는 모델을 만들 수 있음을 보였다. Pytorch의 정수 연산 기능 부재로 시간적 단축을 이루어내지 못한 점은 아쉬우나, 추후에 하드웨어와 소프트웨어 차원에서의 작은 자료형들의 연산 가속 지원으로 실제 low-precision과 quantization이 적용된 모델들이 상용화 될 것이라 예상한다.

References

[1] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. In *OpenAI Blog*, 2019.

[2] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[3] Kalamkar, D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S., et al. A study of BFLOAT16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

[4] Kharya, P. TensorFloat-32 in the A100 GPU accelerates AI training, HPC up to 20x. In *NVIDIA Blog*. 2020. (https://blogs.nvidia.com/blog/2020/05/14/tensorfloat-32-precision-format/)

[5] Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv preprint arXiv:1603.05279*, 2016.

[6] Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[7] Chung, I., Kim, B., Choi, Y., Kwon, S., J., Jeon, Y., Park, B., Kim, S., Lee, D., Extremely low bit transformer quantization for on-device neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4812–4826, 2020.

[8] Jeon, Y., Park, B., Kwon, S. J., Kim, B., Yun, J., Lee, D. BiQGEMM: Matrix multiplication with lookup table for binary-coding-based quantized dnns. *arXiv preprint arXiv:2005.09904*, 2020.

[9] Bondarenko, Y., Nagel, M., Blankevoort, T. Understanding and overcoming the challenges of efficient transformer quantization. *arXiv preprint arXiv:2109.12948*, 2021.

[Appendix]

1. Additional Results

표 9. GPT-2 Small과 GPT-2 Extra-Large의 블록들의 네 개의 linear layers에 INT4 quantization을 적용하고 하나의 linear layer를 선택해 INT8 quantization을 적용했을 때의 LAMBADA dataset 벤치마크 결과

	ACC(%)	PPL
Small 8-4-4-4	43.82	22.80
Small 4-8-4-4	40.97	28.05
Small 4-4-8-4	40.04	29.39
Small 4-4-4-8	44.03	22.58
Extra-Large 8-4-4-4	58.86	7.09
Extra-Large 4-8-4-4	58.88	7.22
Extra-Large 4-4-8-4	59.87	6.94
Extra-Large 4-4-4-8	60.76	6.50