LLaMA: Open and Efficient Foundation Language Models

Touvron, Hugo, et al. "Llama: Open and efficient foundation language models." *arXiv preprint arXiv:2302.13971* (**2023**).

간단 논세

2024.09.06 유하영

Introduction

적절한 규모의 모델을 사용하고 상대적으로 적은 토큰으로 훈련하는 방법을 제안

기존 연구

더 큰 모델을 만드는 데 초점

Chinchilla Scaling
: A replication attempt

작은 모델을 더 많은 토큰으로 오래 훈련

작은 모델을 사용하여
추론이 빠른 모델

무조건적으로 많은 토큰의 수 보다는,
모델의 크기와 데이터를 최적화하는 방법

LLaMA

적절한 규모의 모델과 상대적으로
적은 토큰으로 훈련

- 오픈소싱에 제약 X
- PaLM이나 Chinchilla에 준하는 성능

- 공개된 데이터 사용
- 기존 모델들을 활용한 효율적인 아키텍처 사용
- 추가연구 방향 → instruction tuning의 가능성

Data

다른 LLM을 학습할 때 사용한 데이터 재사용
- 이미 효과가 입증된 데이터
- 오픈소싱과 호환 : 나중에 오픈소스로 공개할 때 제약이 없어야하기 때문

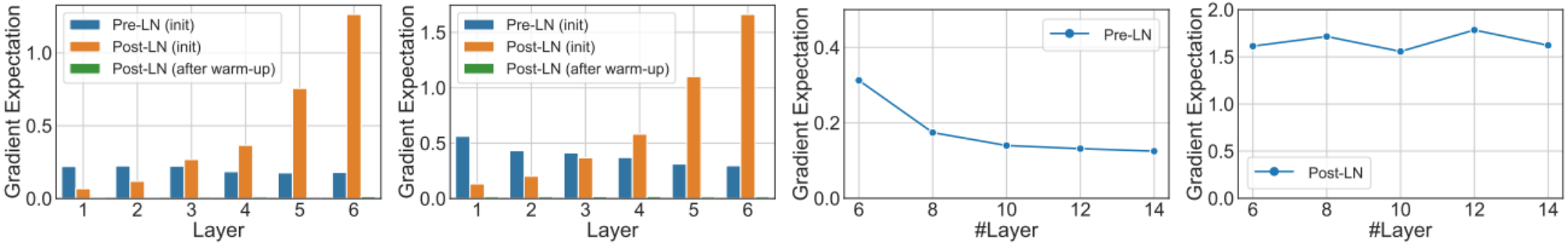
Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

Table 1: **Pre-training data.** Data mixtures used for pre-

CommonCrawl : **웹 크롤링 기반 대규모 데이터셋**
노이즈가 많고, 품질이 낮다.

C4 : CommonCrawl 데이터 기반 전처리된 데이터셋.
구글의 T5모델을 훈련하기 위해 개발된 데이터셋

Pre-normalization [GPT-3]



(a) W^1 in the FFN sub-layers (b) W^2 in the FFN sub-layers (c) Pre-LN Transformer (d) Post-LN Transformer

1. Post-LN 구조의 warm-up

Post-LN은 출력층 근처의 파라미터들이 큰 gradient를 가지기 때문에,
이를 직접 큰 learning rate로 학습하면 학습이 불안정하므로 warm up 필요 (거의 필수적)
* 학습률을 매우 작은 값에서 시작 → 점진적인 학습률 증가(warm up단계)
→ Warm-up기간이 끝나면, 일반적인 학습률로 조정됨

2. warm-up없이, Pre-LN 사용

Layer Normalization을 초기에 배치하여, warm up 없이도 안정적인 학습의 가능성을 제시
(순서) 레이어 정규화->서브레이어->잔차연결

- RMS Normalization 사용
layer 내의 평균 제곱근을 계산하고, 이를 사용해 정규화 진행

$$\bar{a}_i = \frac{a_i}{RMS(a)}, \quad where\ RMS(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$$

a_i is i_{th} input

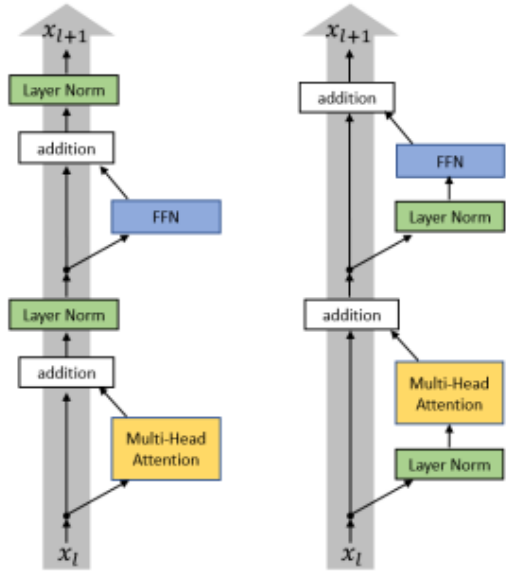


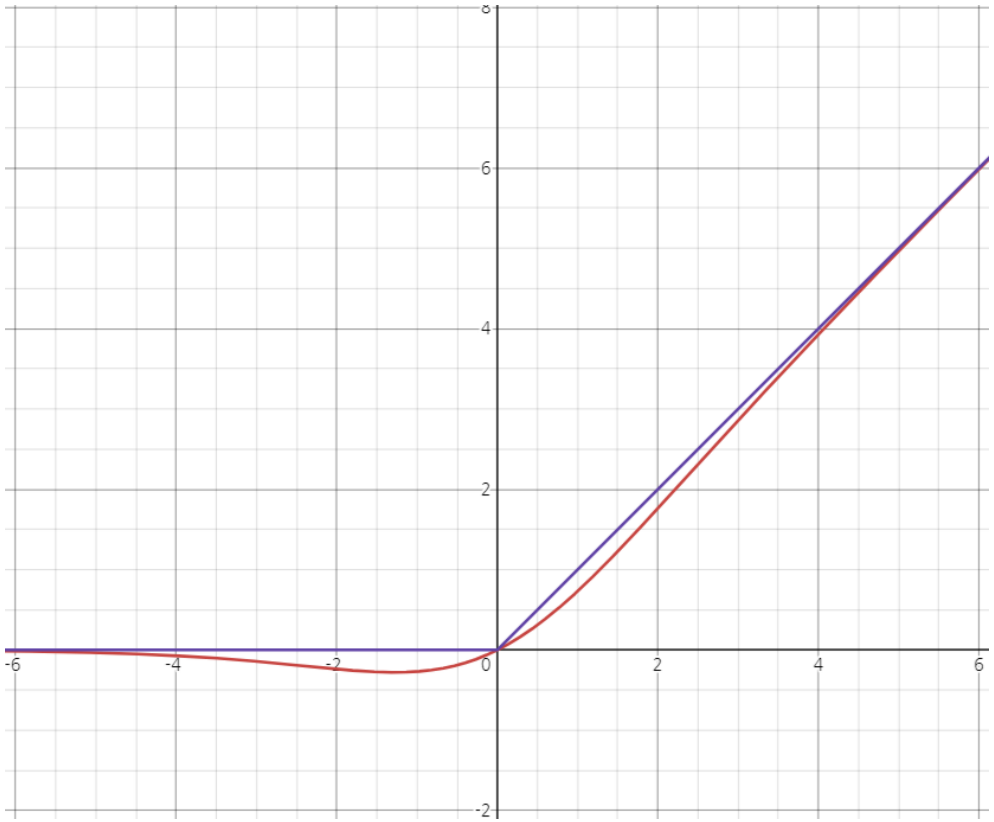
Figure 1. (a) Post-LN Transformer layer; (b) Pre-LN Transformer layer.

https://proceedings.mlr.press/v119/xiong20b

SwiGLU [PaLM]

SwiGLU => Swish함수 + GLU (Gated Linear Unit)

ReLU 활성화 함수 대신 사용



Swish함수

$$Swish(x)=x\cdot\sigma(x)$$

GLU

입력 벡터를 두 부분으로 나누어, 각각의 값을 다르게 처리하는 활성화 함수 (더 복잡한 비선형성 모델링 가능)

$$GLU(x)=x\_1\cdot\sigma(x\_2)$$

x_1 : 입력벡터의 첫 번째 절반, 선형변환 후 활성화 함수 없이 그대로 남음

x_2 : 입력벡터의 두 번째 절반, 시그모이드 함수를 적용하여 게이트 역할 수행

Rotary Positional Embeddings(RoPE) [GPTNeo]

상대적인 위치 임베딩 파악을 위해, **회전 행렬**을 사용하여 **토큰 간의 상대적 위치정보를 인코딩** 하는 것

Absolute Position Embedding

sinusoid 사용
시퀀스 내에서 **각 토큰의 고유한 위치**가 모델에 제공
→ **절대적인 위치**에 기반한 임베딩을 제공

Rotary Positional Embeddings (RoPE)

θ(sinusoid 사용)
q,k에 대해서 벡터를 회전하는 방식으로 **위치 정보를 삽입**

1.단어를 임베딩하여 벡터화
2.2차원의 하위벡터로 나눔
3.회전 변환 적용(회전 행렬사용, *θ 는 sinusoid 사용)*
4.기존 벡터는 위치정보가 반영된 임베딩 벡터로 변환(q,k에 대해서만)

쿼리와 키 벡터를 위치에 따라 각기 다른 각도로 회전 → 위치에 기반한 상대적인 정보를 인코딩
→ q벡터인 m과 k벡터인 n과의 차이를 통해 서로 다른 각도로 회전하여 연산. **회전 행렬은 각도만 변경하므로, 벡터의 크기나 방향 손상X, 단어 임베딩이 가진 의미는 유지된다.**

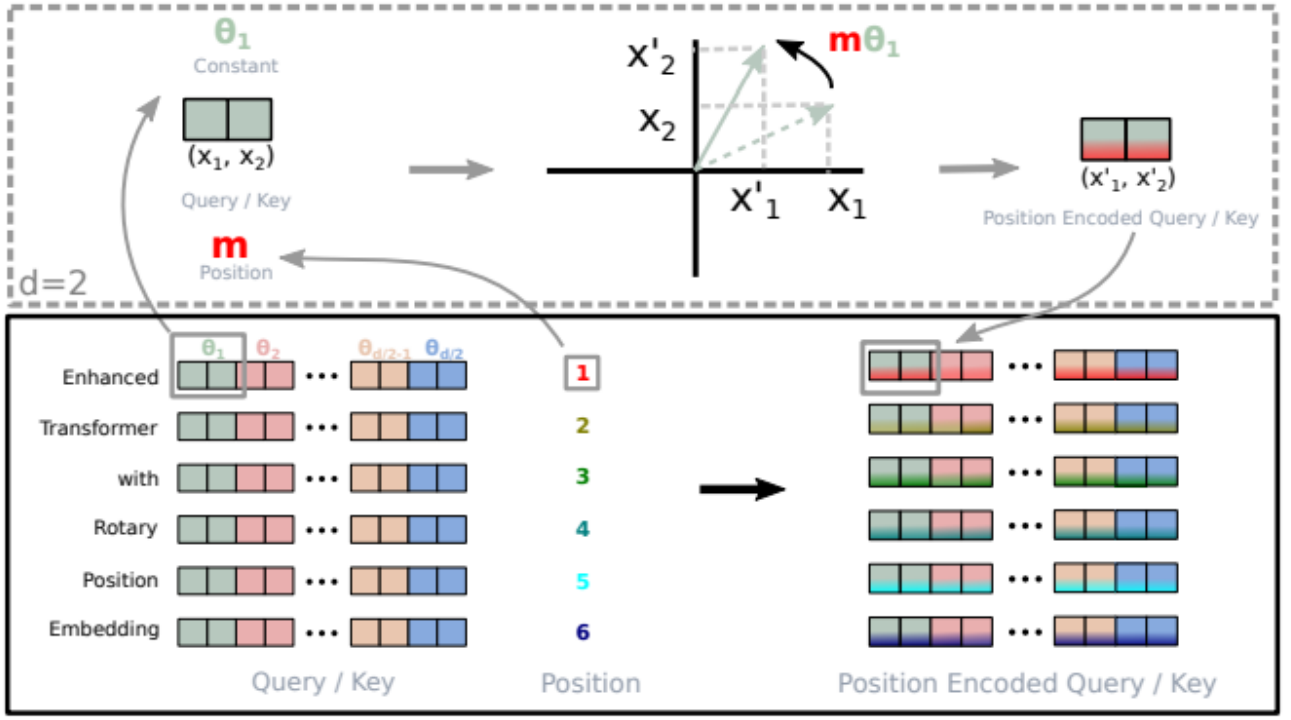


Figure 1: Implementation of Rotary Position Embedding(RoPE).

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$

https://www.sciencedirect.com/science/article/abs/pii/S0925231223011864
https://velog.io/@alstjsdlr0321/Chapter-8.-LLaMA-2-Part1#rotary-positional-embedding-rope

Main results

zero-shot과 few-shot 태스크에서의 성능을 측정한 방법과 결과에 대한 설명

Zero-shot
모델에 대한 아무런 예시를 주지 않고, 주어진 작업에 대한 텍스트 설명과 예제를 제공
Few-shot
작업에 대해 몇 개의 예시를 제공

		BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA
GPT-3	175B	60.5	81.0	-	78.9	70.2	68.8	51.4	57.6
Gopher	280B	79.3	81.8	50.6	79.2	70.1	-	-	-
Chinchilla	70B	83.7	81.8	51.3	80.8	74.9	-	-	-
PaLM	62B	84.8	80.5	-	79.7	77.0	75.2	52.5	50.4
PaLM-cont	62B	83.9	81.4	-	80.6	77.0	-	-	-
PaLM	540B	**88.0**	82.3	-	83.4	**81.1**	76.6	53.0	53.4
LLaMA	7B	76.5	79.8	48.9	76.1	70.1	72.8	47.6	57.2
	13B	78.1	80.1	50.4	79.2	73.0	74.8	52.7	56.4
	33B	83.1	82.3	50.4	82.8	76.0	**80.0**	**57.8**	58.6
	65B	85.3	**82.8**	**52.3**	**84.2**	77.0	78.9	56.0	**60.2**

상식추론 task에서의 Zero-shot performance

Table 3: **Zero-shot performance on Common Sense Reasoning tasks.**

“ LLaMA-13B 모델은 GPT-3보다 10배 이상 작은 크기에도 불구하고 이를 능가하는 성능을 보였으며,
LLaMA-65B 모델은 Chinchilla-70B 및 PaLM-540B와 경쟁할 만한 성능을 보였습니다. ”

Instruction Finetuning

좌측 - 기존 모델들로 MMLU task 성능 (가장 오른쪽 내용이 평균)
우측 - 파인튜닝된 모델을 바탕으로 MMLU task 성능(평균만 제시)

		Humanities	STEM	Social Sciences	Other	Average
GPT-NeoX	20B	29.8	34.9	33.7	37.7	33.6
GPT-3	175B	40.8	36.7	50.4	48.8	43.9
Gopher	280B	56.2	47.4	71.9	66.1	60.0
Chinchilla	70B	63.6	54.9	79.3	**73.9**	67.5
PaLM	8B	25.6	23.8	24.1	27.8	25.4
	62B	59.5	41.9	62.7	55.8	53.7
	540B	**77.0**	**55.6**	**81.0**	69.6	**69.3**
LLaMA	7B	34.0	30.5	38.3	38.1	35.1
	13B	45.0	35.8	53.8	53.3	46.9
	33B	55.8	46.0	66.7	63.4	57.8
	65B	61.8	51.7	72.9	67.4	63.4

Table 9: **Massive Multitask Language Understanding (MMLU).** Five-shot accuracy.

OPT	30B	26.1
GLM	120B	44.8
PaLM	62B	55.1
PaLM-cont	62B	62.8
Chinchilla	70B	67.5
LLaMA	65B	63.4
OPT-IML-Max	30B	43.2
Flan-T5-XXL	11B	55.1
Flan-PaLM	62B	59.6
Flan-PaLM-cont	62B	66.1
LLaMA-I	65B	**68.9**

Table 10: **Instruction finetuning – MMLU (5-shot).** Comparison of models of moderate size with and without instruction finetuning on MMLU.

- **Instruction Finetuning** : LLaMA 모델을 추가로 명령어 데이터를 사용해 미세 조정하여, 모델이 주어진 명령어를 더 잘 이해하고 따를 수 있게 만드는 과정
- fine tuning된 LLaMA-65B 모델(LLaMA-I)은 **MMLU(대규모 멀티태스크 언어 이해)** 벤치마크에서 더 나은 성능을 보였다.

Conclusion

1. LLaMA-13B 모델은 GPT-3보다 10배 이상 작은 크기에도 불구하고 이를 능가하는 성능을 보였으며, LLaMA-65B 모델은 Chinchilla-70B 및 PaLM-540B와 경쟁할 만한 성능을 보였다.

2. 기존 연구들과는 달리, 우리는 독점 데이터셋을 사용하지 않고도 **공개적으로 사용 가능한 데이터**로만 학습하여 더 나은 성능을 달성했음을 관찰
오픈소스로 제공했다는 점에서, 추후 많은 대규모 LM들의 개발을 가속화하는데 기여했다는 점에서 높은 가치가 있다.

3. 모델들을 명령어 데이터로 미세 조정하는 것이 유망한 결과를 낳는다는 것을 관찰

7주차

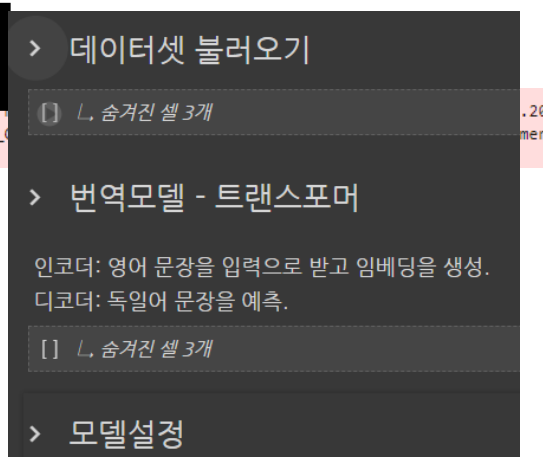
WMT 2016 번역모델 구현

WMT2016 번역 모델 구현

1. 제한된 자원/메모리 부족

```
OutOfMemoryError: CUDA out of memory. Tried to allocate 5.88 GiB. GPU 0 has a total capacity of 14.74 GiB of which 1.22 GiB is free. Process 2975 [illegible].20
GiB is allocated by PyTorch, and 5.12 GiB is reserved by PyTorch but unallocated. If reserved but unallocated memory is large try setting PYTORCH_[illegible]ment
ation.  See documentation for Memory Management  (https://pytorch.org/docs/stable/notes/cuda.html#environment-variables)
```

Colab



2. 텐서 변환.. 문제..

Trainer O –〉 컴퓨팅 자원 문제, **텐서 변환 문제**

```
First sample 'input_ids': [32, 582, 287, 4077, 6622, 257, 10047, 981, 262, 584, 582, 34526, 465, 10147, 13, 50256, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
First sample 'labels': [36, 259, 20291, 287, 1036, 9116, 77, 289, 11033, 2528, 304, 500, 402, 7940, 260, 11, 266, 11033, 71, 10920, 4587, 290, 567, 20291, 384, 259, 15617,
```

문장 + 〈EOS〉 + zero padding

문장 + zero padding + 〈EOS〉 –〉 X

문장 + 〈EOS〉 + zero padding –〉 텐서 불일치

```python
def tokenize_function(examples):
    max_length = 128

    #토크나이즈
    inputs=tokenizer(examples['en'], padding=False, truncation=True, max_length=max_length-1)
    targets= tokenizer(examples['de'], padding=False, truncation=True, max_length=max_length-1)

    #EOS 토큰추가
    inputs['input_ids'] =[input_ids + [tokenizer.eos_token_id] for input_ids in inputs['input_ids']]
    targets['input_ids'] =[target_ids + [tokenizer.eos_token_id] for target_ids in targets['input_ids']]

    #zero 패딩
    def pad_to_max_length(seq):
        return seq +[tokenizer.pad_token_id] * (max_length - len(seq))

    inputs['input_ids'] = [pad_to_max_length(seq) for seq in inputs['input_ids']]
    targets['input_ids'] = [pad_to_max_length(seq) for seq in targets['input_ids']]
```

016 번역 모델 구현

```python
import torch.nn as nn

def train(model, train_loader, val_loader, optimizer, loss_fn, epochs, batch_size, device):
    train_losse_list=[]
    val_loss_list = []
    val_bleu_score_list = []

    for epoch in range(epochs):
        model.train()
        epoch_loss = 0
        preds_list = []
        targets_list = []

        loop = tqdm(train_loader, leave=True, desc=f"Epoch {epoch+1}/{epochs}")

        for batch in loop:
            src = batch['input_ids'].to(device) # GPU로 이동
            tgt = batch['labels'].to(device)

            output = model(src, tgt)
            optimizer.zero_grad()

            # 손실 계산을 위한 출력과 라벨 크기 맞추기
            output = output.reshape(-1, output.shape[-1])  # (
            tgt = tgt.reshape(-1)  # (batch_size * seq_len)
            tgt = tgt.long()

            #loss
            loss = loss_fn(outpu
            loss.backward()
            optimizer.step()
            epoch_loss += loss.i

            #예측값 추출 및 저장
            preds = torch.argma
            preds_list.extend(pr
            #실제값도 리스트에 저
            targets_list.extend(t
            loop.set_postfix(loss

        #에포크가 끝나면 손실 값 출력
        epoch_loss = epoch_loss /
        train_losse_list.append(epoch_loss)#loss 저장

        bleu_score = calculate_bleu(preds_list, targets_list)
        print(f'Epoch {epoch+1}: Loss = {epoch_loss}, BLEU-4 Score = {bleu_score}')


        model.eval()  #평가 모드로 전환
        val_avg_loss, val_bleu_score = evaluate(model, val_loader, loss_fn, batch_size, device)

        val_loss_list.append(val_avg_loss)#loss 저장
        val_bleu_score_list.append(val_bleu_score)#BLEU score 저장

        model.train()  #학습 모드로 복귀

    return train_losse_list, val_loss_list, val_bleu_score_list
```

```python
# 모델 초기화
def initialize_model(input_vocab_size, target_vocab_size, smoothing=0.1, eos_weight=0.1):
    # 모델 정의
    model = Transformer(input_vocab_size=input_vocab_size,
                        target_vocab_size=target_vocab_size,
                        embed_size=512,
                        num_heads=4,
                        ff_hidden_dim=1024,
                        num_layers=4,
                        dropout=0.1)  # .to(device)

    optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
```

```python
#평가
def evaluate(model, val_loader, loss_fn, batch_size, device):
    model.eval()
    preds_list = []
    targets_list = []
    total_loss = 0.0

    with torch.no_grad():#그래디언트 계산 비활성화로 메모리 절약
        for batch in val_loader:
            src = batch['input_ids'].to(device)
                                                    ut.shape
                                                    ze * seq_len)
                                                    .reshape(batch_size, -1).detach().cpu().numpy()
                                                    .detach().cpu().numpy()
                                                    e(batch_size, -1).detach().cpu().numpy())
            targets_list.extend(tgt.detach().cpu().numpy())
            #loop.set_postfix(loss=loss.item())

    #평균 validation loss
    avg_loss = total_loss / len(val_loader)

    bleu_score = calculate_bleu(preds_list, targets_list)
    print(f'Validation Loss: {avg_loss}, Validation BLEU-4 Score: {bleu_score}₩n')

    return avg_loss, bleu_score
```

Transformer를 활용한 번역 모델 구현

TEST 결과

```
Epoch 1/5: 100%| 1813/1813 [06:27<00:00,  4.68it/s, loss=0.179]
Epoch 1: Loss = 0.7193282372074205, BLEU-4 Score = 9.338672068856056e-234
Validation BLEU-4 Score: 0.20827334697358543
Epoch 1: Validation BLEU-4 Score = None
Epoch 2/5: 100%| 1813/1813 [06:27<00:00,  4.68it/s, loss=0.0984]
Epoch 2: Loss = 0.0774451351609755, BLEU-4 Score = 5.689361740571835e-234
Validation BLEU-4 Score: 0.1722808700397511
Epoch 2: Validation BLEU-4 Score = None
Epoch 3/5: 100%| 1813/1813 [06:25<00:00,  4.70it/s, loss=0.0164]
Epoch 3: Loss = 0.03472983379569881, BLEU-4 Score = 5.161633751637872e-234
Validation BLEU-4 Score: 0.17167888213630944
Epoch 3: Validation BLEU-4 Score = None
Epoch 4/5: 100%| 1813/1813 [06:26<00:00,  4.69it/s, loss=0.0216]
Epoch 4: Loss = 0.0178024125648177, BLEU-4 Score = 4.87645738061956e-234
Validation BLEU-4 Score: 0.15770328625746347
Epoch 4: Validation BLEU-4 Score = None
Epoch 5/5: 100%| 1813/1813 [06:26<00:00,  4.69it/s, loss=0.00064]
Epoch 5: Loss = 0.00889529055370923, BLEU-4 Score = 5.509180443788247e-234
Validation BLEU-4 Score: 0.1645353103171813
Epoch 5: Validation BLEU-4 Score = None
Evaluating on test data...
Validation BLEU-4 Score: 0.153249745408643
```

Translated Sentence

```
Model output logits: tensor([[[ 1.6959, -0.4013,  0.4710,  ...,  0.2187,  0.1952, 13.2880],
         [12.7492, -0.3853, -0.2550,  ..., -0.4283, -0.4328, -0.3742],
         [12.7712, -0.3811, -0.3528,  ..., -0.4193, -0.4176, -0.3491],
         ...,
         [12.7376, -0.5471, -0.5061,  ..., -0.4110, -0.4059, -0.3789],
         [12.7379, -0.5399, -0.5019,  ..., -0.4086, -0.4049, -0.3592],
         [12.7362, -0.5306, -0.4962,  ..., -0.4090, -0.4071, -0.3275]]],
       device='cuda:0')
Next token probabilities: tensor([[8.4129e-06, 1.0331e-06, 2.4714e-06,  ..., 1.9204e-06, 1.8757e-06,
         9.1055e-01]], device='cuda:0')
Next token: 50256
Model output logits: tensor([[[ 1.6959, -0.4013,  0.4710,  ...,  0.2187,  0.1952, 13.2880],
         [ 1.8194, -0.4494,  0.3786,  ...,  0.2266,  0.1859, 13.3382],
         [12.7712, -0.3811, -0.3528,  ..., -0.4193, -0.4176, -0.3491],
         ...,
         [12.7376, -0.5471, -0.5061,  ..., -0.4110, -0.4059, -0.3789],
         [12.7379, -0.5399, -0.5019,  ..., -0.4086, -0.4049, -0.3592],
         [12.7362, -0.5306, -0.4962,  ..., -0.4090, -0.4071, -0.3275]]],
       device='cuda:0')
Next token probabilities: tensor([[9.0861e-06, 9.3982e-07, 2.1510e-06,  ..., 1.8477e-06, 1.7739e-06,
         9.1396e-01]], device='cuda:0')
Next token: 50256
Translated Sentence:
```