Cross-Entropy-Loss

① How to teach a Language Model
② Cross-Entropy-Loss
③ Generate text from the GPTs

How to teach a language model

Language Model

- **Probability of a Word(token) Sequence** (주어진 sequence가 나올 확률) :
  - Given a sequence of words $W = w_1, w_2, \ldots, w_t$,
  - $P(W) = P(w_1, w_2, \ldots, w_t)$:

$$P(W) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \cdots P(w_t|w_1, w_2, \ldots, w_{t-1})$$

- **주어진 Corpus에 대하여 어떻게 Language Model을 학습하는가?**

① **통계적 방식 – n-gram 방식(예: kenlm)**

② **신경망 학습 방식 – RNN(Recurrent Neural Network), Transformer Network**

Language Model 학습 방법

▪ n-gram 방식

a. 긴 문맥 처리 어렵고

b. 통계적 예측만 하기 때문에 새로운 단어/유사어, 단어 의미 해결 못함

▪ RNN (Recurrent Neural Network) 방식

a. word를 vector로 학습, 단어에 내재된 의미 파악 가능(Word2Vec)

b. RNN 구조의 한계로 long context 처리 한계

c. 학습 시 토큰 시퀀스를 순차적으로 처리하기 때문에 속도가 느려져서 대용량 데이터 학습에 불리

- RNN은 입력 token을 한 번에 하나씩 순서대로 입력 받아 연산 – token의 시간 순서를 자동으로 유지함

Language Model 학습 방법

▪ Transformer Network 방식

a. Attention으로 long context 처리할 수 있으며,

b. 토큰 시퀀스를 한 번에 병렬로 처리 가능하여 대용량 데이터 학습에 유리

- Transformer Network은 최대 토큰 수만큼 한 번에 입력 받아 병렬로 연산

① Position Embedding이 순서를 표현함

② Masked Causal Attention을 사용하여 현재 토큰 이후의 토큰을 미리 참조해서 계산하지 못하도록 장치 마련

c. 컴퓨팅 리소스를 많이 필요로 하는 단점이 있으나 장점들이 단점을 무색케 함

신경망 기반 Language Model 학습 방법

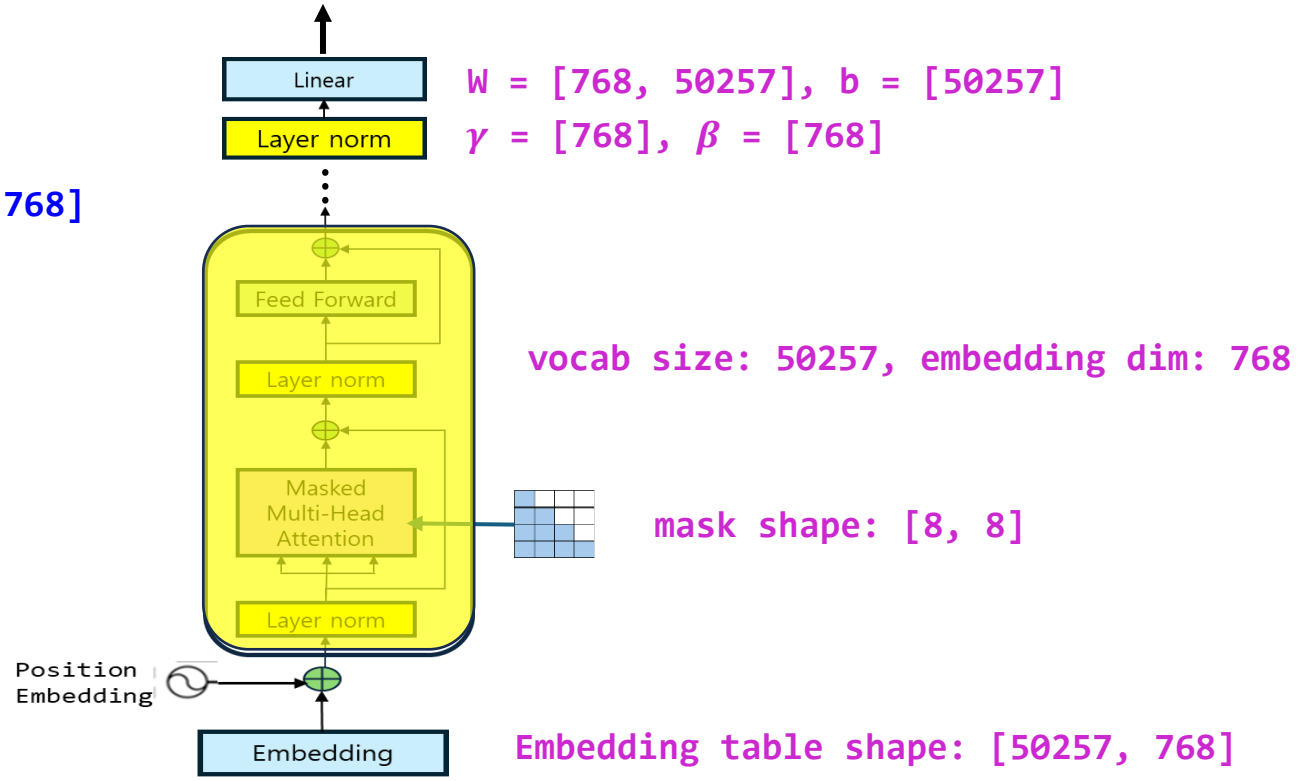
- **Train the network Θ to maximize log-likelihood $log\ P(W;\Theta)$:**
  - $\log P(W;\Theta) = \log P(w_1;\Theta) + \log P(w_2|w_1;\Theta) + \cdots + \log P(w_t|w_1, w_2, \cdots, w_{t-1};\Theta)$
  - 각 log-term은 지난 단어들 다음에 현재 단어가 나올 log-probability
- **Cross-entropy-loss를 minimize하는 것으로 위 목적 달성**
  - $E[-P_{target} log\ P_{model}]$
- **Perplexity를 minimize하는 것과 같음(why?)**
  - $PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$

Input & Target for Transformer Network in Learning LM

⑤ 타겟 크기: [1, 8]

345	257	1692	393	257	9379	30	220

④ 최종 출력 크기: [1, 8, 50257]



W = [768, 50257], b = [50257]

γ = [768], β = [768]

③ 트랜스포머 블록 출력 크기: [1, 8, 768]

vocab size: 50257, embedding dim: 768

mask shape: [8, 8]

② 임베딩 후 입력 크기: [1, 8, 768]

Embedding table shape: [50257, 768]

① 입력 크기: [1, 8]

8491	345	257	1692	393	257	9379	30

Are	you	a	human	or	a	robot	?

Cross-Entropy-Loss, Conditional Probabilities, Causal Attention

$$\log P(W;\Theta) = \log P(w_1;\Theta) + \log P(w_2|w_1;\Theta) + \cdots + \log P(w_t|w_1, w_2, \cdots, w_{t-1};\Theta)$$

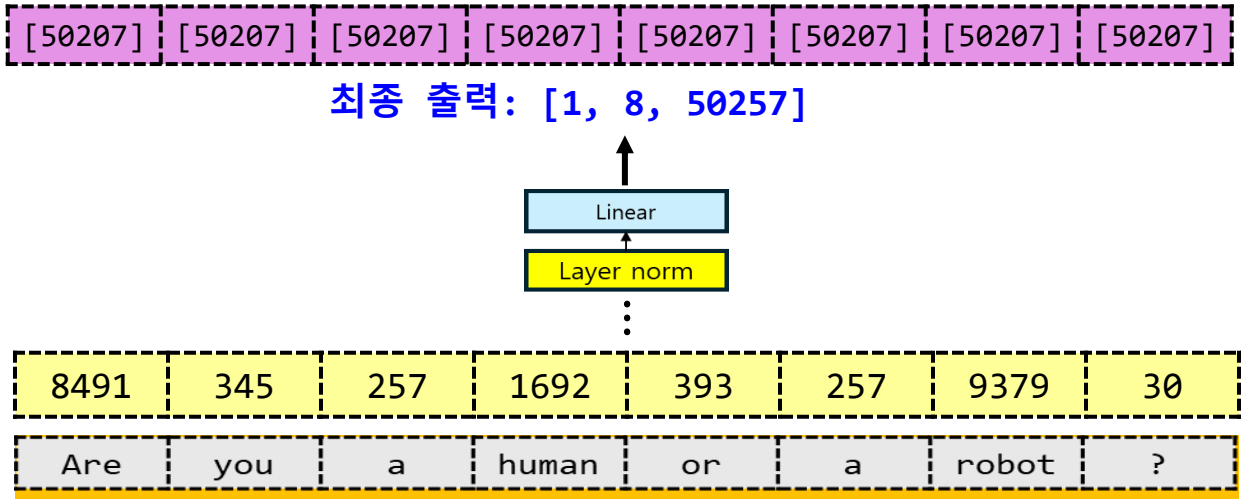
345	257	1692	393	257	9379	30	220

① 8개의 위치에서 50207개의 logits을 출력

② 각 logit은 50207개의 토큰 중 특정 토큰이 나올 로그 확률을 나타냄

③ 예를 들어, 첫번째 위치에서 열 번째 logit은 $\log P(W_1 = 9|W_0 = 8401)$

④ 각 위치에서 타겟 토큰이 나올 확률이 '1'이 되도록 학습하는 것이 목표



Cross-entropy-loss는 나올 수 있는 50207개의 확률 중에서 다음 8개의 확률이 각 타겟 위치에서 '1'이 되도록 학습

1) $\log P(W_1 = 345|W_0 = 8401;\Theta)$
2) $log\, P(W_2 = 257|W_0 = 8401, W_1 = 345;\Theta)$
3) $log\, P(W_3 = 1692|W_0 = 8401, W_1 = 345, W_2 = 257;\Theta)$
4) ⋮
8) $log\, P(W_8 = 220|W_0 = 8401, \ldots, W_6 = 9379, W_7 = 30;\Theta)$

위 확률을 모두 더한 값의 평균에 (-1)을 곱한 값이 cross -entropy loss이다.
주의할 점은 각 확률은 LM에서 정의한 조건부 확률이다.
트랜스포머 모델은 위 조건부 확률을 'causal attention'으로 아름답게 구현함

Training GPT (Generative Pretrained Transformer Network)

GPT-2 (small)

(N) layer = 12
(B) batch = 64
(C) dim = 768
(H) head = 12
(T) max seq length = 1024
(V) vocab size = 50257

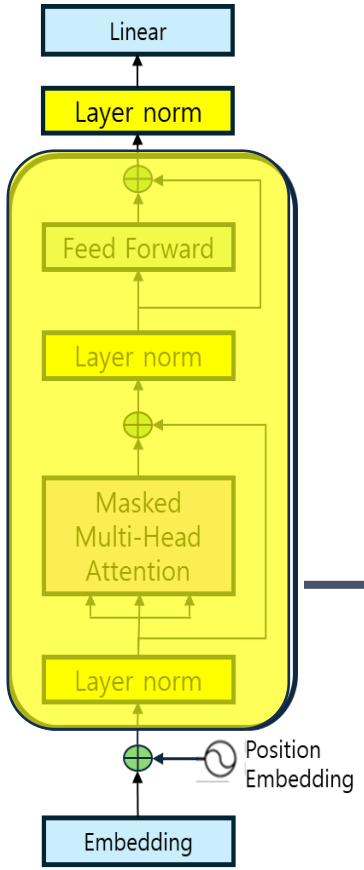
We can also use the existing language functionality in the model to perform sentiment analysis. For the Stanford Sentiment Treebank dataset, which consists of sentences from positive and negative movie reviews, we can use the language model to guess whether a review is positive or negative by inputting the word "very" after the sentence and seeing whether the model predicts the word "positive" or "negative" as more likely. This approach, without adapting the model at all to the task, performs on par with classic baselines ~80% accuracy.

↓ tokenizer ↓

[[82, 399, …, 243], …, [677, 3222, …, 200]]

① input-token-ids: [B, T]
② Embedding output: X = [B, T, C]

Transformer Network



Target and logits

① target dimension? [B, T]
② target 구성
- next token을 predict하도록 구성
- input token이 [1, 2, 3]이라면, targe은 [2, 3, idx]: 여기서 idx는 data에서 next token

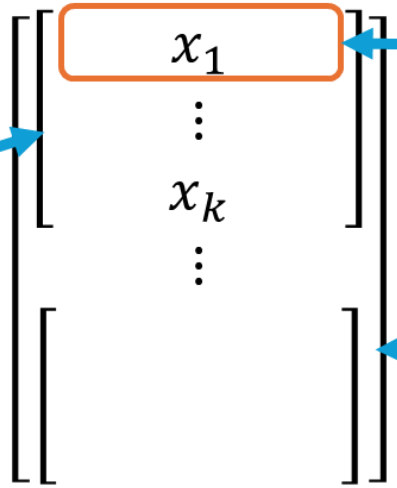
③ 최종 linear 출력인 logits의 shape은 [B, T, V]이고 타겟 토큰을 맞추도록 축-V에 대하여 cross-entropy 최소화

① Layer norm output: [B, T, C]
② Linear output: logits = [B, T, V]
③ Target: [B, T]

Training Input Tensor: batch-seq-dim

We can also use the existing language functionality in the model to perform sentiment analysis. For the Stanford Sentiment Treebank dataset, which consists of sentences from positive and negative movie reviews, we can use the language model to guess whether a review is positive or negative by inputting the word "very" after the sentence and seeing whether the model predicts the word "positive" or "negative" as more likely. This approach, without adapting the model at all to the task, performs on par with classic baselines ~80% accuracy.

context window is a chunk of tokens from training data



embedding vector for each token in a context window

minibatch is a group of context windows

GPT-1: $X = [64 \times 512 \times 768]$

model	number of tokens	embedding dimension	context window (tokens in a batch)	number of batch
GPT-1	40,458	768	512	64
GPT-2	50,257	1,600	1024	512
GPT-3	50,257? (unknown)	12,288	2048	1600? 3.2M in tokens

Cross-Entropy-Loss

1

Probabilities = Softmax on a sequence of logits [B, T, V]

- 길이가 T인 시퀀스의 특정 위치에 V개의 토큰 중 타겟 토큰을 예측하도록 학습
- 트랜스포머 모델의 출력을 'logits'으로 해석 – un-normalized log probabilities
- Logit을 확률로 변환하기 위하여 'softmax'를 각 위치에서 V개의 logits에 대하여 softmax 적용
- Softmax input이 1D vector x일 때, softmax 는

$$softmax(x)_k = \frac{\exp(x_k)}{\sum_i \exp(x_i)}$$

  - 실제로는, 계산의 안정을 위해서 vector x의 max element를 모든 element에서 빼서 계산
  - x_k는 log 확률이므로 exponential를 씌우고 합이 '1' 되도록 normalization하는 것이 softmax operation임

- Softmax를 [B, T, V] tensor에 적용하면, 길이가 T인 시퀀스로 구성된 B개의 샘플의 각 위치에서 나올 수 있는 모든 가능한 토큰 V개의 확률을 계산 하는 것임 – 즉, 입력 샘플에 대한 토큰 확률 분포를 계산

Softmax 예제

```
import torch
import torch.nn.functional as F

# softmax 함수 정의
def softmax(x, dim=-1):
    e_x = torch.exp(x - x.max(dim=dim, keepdim=True).values)
    return e_x / e_x.sum(dim=dim, keepdim=True)

x = torch.randn(32, 1024, 50257)
softmax(x)

# PyTorch에 정의된 softmax 함수 사용
F.softmax(x, dim=-1)
```

Cross-Entropy-Loss

- Cross-Entropy-Loss

$$E[-P_{target} log\ P_{model}]$$

  - Expectation은 학습 샘플의 평균으로 근사화 할 수 있다.

- Cross-Entropy-Loss를 트랜스포머 모델에 적용하면,
  - ① P_{model} 은 입력 샘플에 대하여 트랜스포머 모델이 계산한 토큰 확률 분포
  - ② P_{target} 은 입력 샘플에 대하여 트랜스포머 모델이 예측해야 할 정답 토큰 확률 분포
  - ③ P_{target} 과 같이 타겟 토큰 확률이 '1', 다른 확률은 '0'인 확률 분포를 1-hot vector라고 부름

- P_{target} 이 1-hot vector임을 적용하면,

$$E[-log\ P_{model}(W)]$$

  - 여기서, W는 특정 샘플을 의미하며, $P_{model}(W)$은 특정 샘플의 특정 위치에서 target token이 나올 모델의 예측 확률을 의미함

$P_{model}(W)$의미

- $P_{model}(W)$는 특정 샘플의 특정 위치에서 나와야 할 target token의 확률을 모델이 예측한 값이다.
- 주목해야 할 점은 트랜스포머 모델이 Masked Causal Attention을 사용하기 때문에, 시퀀스의 특정 위치에서의 확률 예측은 그 이전 위치의 정보를 반영한 조건부 확률이라는 점이다.
- 특정 샘플 시퀀스에 $(w_1, \dots, w_T)$ 대하여 모델이 예측한 로그 확률 값은 다음과 같이 된다.

$$log\ P_{model}(w_1, \dots, w_T) = log\ P_{model}(w_1) + log\ P_{model}(w_2|w_1) + \cdots + log\ P_{model}(w_T|w_1, \dots, w_{T-1})$$

  - $P_{model}(w_1)$: 1-번째 위치에 토큰 w_1 나올 확률 예측
  - $log\ P_{model}(w_2|w_1)$: 1-번째 위치에 토큰 토큰 w_1 나오고, 2-번째 위치에 토큰 w_2 가 나올 확률

- 트랜스포머 모델의 (a) Masked Causal Attention과 Cross-Entropy-Loss는 'Language Model'을 문자 그대로 구현한 것이며, (b) 대용량 데이터를 효율적으로 학습할 수 있다 점에서 'Large Language Model'이라고 부른다.

Cross-Entropy-Loss 구현

- 수식 정의 대로 구현: Logits으로부터 Softmax를 구하고, 주어진 타겟을 이용하여 해당 logits을 수집(gather)하고 (-1)을 곱하고, 평균을 구한다. → 계산 안정성이 떨어짐

- 실제 구현

① Logits으로부터 log-softmax를 계산하여 log-probabilities를 구한다.

② 타겟을 이용하여 해당 클래스의 log-probs만 뽑는다.

③ 선택한 log-probs의 구하고 (-1)을 곱하고 평균을 구한다.

```
log_probs = F.log_softmax(logits, dim=-1)
chosen_log_probs = log_probs[torch.arange(len(targets)), targets] # indexing이용 selection
nll = -chosen_log_probs
loss = nll.mean()
```

cross-entropy loss 구현

```
def log_softmax(x, dim=-1):
    e_x = torch.exp(x - x.max(dim=dim, keepdim=True).values)
    sum_e_x = e_x.sum(dim=dim, keepdim=True)
    return x - x.max(dim=dim, keepdim=True).values - torch.log(sum_e_x)


def cross_entropy(pred, target):
    log_probs = log_softmax(pred, dim=-1)
    nll_loss = -log_probs[torch.arange(len(target)), target]
    return nll_loss.mean()
```

사용 예

```
logits = torch.tensor([[2.0, 0.5, 0.1, -1.0],
                       [0.1, 0.2, 3.0, 0.5],
                       [1.0, 2.0, 0.1, 0.0]])
targets = torch.tensor([0, 2, 1])  # class indices

loss_custom = cross_entropy(logits, targets)

import torch.nn.Functional as F
F.cross_entropy(logits, targets)  # combines log_softmax and nll_loss
```

torch.nn as nn 사용 예

```
# 가상 데이터 (batch_size=2, seq_len=4, vocab_size=10)
batch_size, seq_len, vocab_size = 2, 4, 10
logits = torch.randn(batch_size, seq_len, vocab_size)  # 모델 출력 (logits)
targets = torch.randint(0, vocab_size, (batch_size, seq_len))  # 정답 레이블

print("logits shape:", logits.shape)   # (2, 4, 10)
print("targets shape:", targets.shape) # (2, 4)

# CrossEntropyLoss는 입력을 (N, C) 형태로 기대하므로 변환 필요
loss_fn = nn.CrossEntropyLoss()

# (batch_size * seq_len, vocab_size) 로 flatten
logits = logits.view(-1, vocab_size)   # (8, 10)
targets = targets.view(-1)             # (8,)
loss = loss_fn(logits, targets)

print("Loss:", loss.item())
```

2

Text Generation from GPT-2

GPT 모델에서 Text 생성하기

- GPT 모델을 학습할 때, 입력은 [B, T, D] 이다. Test할 때는?
- B (Batch size): 학습할 때 보다 크거나 작을 수 있다.
- T (Sequence length): 학습할 때 보다 작을 수 있으며, 기준 값보다 커지면 앞에 부분을 잘라서 입력
- D (embedded dimension): 학습할 때와 반드시 일치하여야 함

- 입력이 [1, 8, 768] 일 때, 트랜스포머 모델의 logits size는?
- logits 직전의 layer norm output size는 [1, 8, 768]
- logits을 출력하는 linear의 weights [V, 768]와 bias [V], (V는 vocab size)
- logits의 크기는? [1, 8, V]

Next Token 구하기

- Next token은 어떻게 구하는가? Autoregression 방식
- Logits의 마지막 값 [1, -1, V]에서 V에 대하여 softmax를 하여, 다음 토큰이 나올 확률분포를 구한다.
- 구해진 확률 분포로부터 'sampling'하여 다음 토큰을 구함
- 생성된 토큰을 이전 토큰 시퀀스와 함께 다시 입력하여 같은 방법으로 다음 토큰을 구한다.
- 점차적으로 시퀀스 길이가 길어진다.

- 확률 분포에서 샘플링 하는 방법

```
logits = logits[:, -1, :] / temperature
if top_k is not None:
    v, _ = torch.topk(logits, top_k)
    logits[logits < v[:, [-1]]] = -float('Inf')
# apply softmax to convert logits to (normalized) probabilities
probs = F.softmax(logits, dim=-1)
# sample from the distribution
idx_next = torch.multinomial(probs, num_samples=1)
```

GPT2를 이용한 Text Generation

1. Random Text Generation from non-trained GPT2 model.

- Download GPT2.py (Simpler Version of nanoGPT)
- Use `generate` function to produce random text from initialized GPT2.

2. Text Generation from Huggingface pretrained GPT2 model

① Use `generate` function to produce texts

② Use `logits` to produce texts

GPT-2 에서 text generate 하기 – GPT2.py

```
@torch.no_grad()
def generate(self, idx, max_new_tokens, temperature=1.0, top_k=None):
        for _ in range(max_new_tokens):
        # if the sequence context is growing too long we must crop it at seq_len
        idx_cond = idx if idx.size(1) <= self.config.seq_len else idx[:, -self.config.seq_len:]
        # forward the model to get the logits for the index in the sequence
        logits, _ = self(idx_cond)
        # pluck the logits at the final step and scale by desired temperature
        logits = logits[:, -1, :] / temperature
        # optionally crop the logits to only the top k options
        if top_k is not None:
            v, _ = torch.topk(logits, min(top_k, logits.size(-1)))
            logits[logits < v[:, [-1]]] = -float('Inf')
        # apply softmax to convert logits to (normalized) probabilities
        probs = F.softmax(logits, dim=-1)
        # sample from the distribution
        idx_next = torch.multinomial(probs, num_samples=1)
        # append sampled index to the running sequence and continue
        idx = torch.cat((idx, idx_next), dim=1)
    return idx
```