2025.10.01

BERT
GPT-1, 2, 3

팀: 레모네이드

김선준 장우진 장재우 박은수 강희준

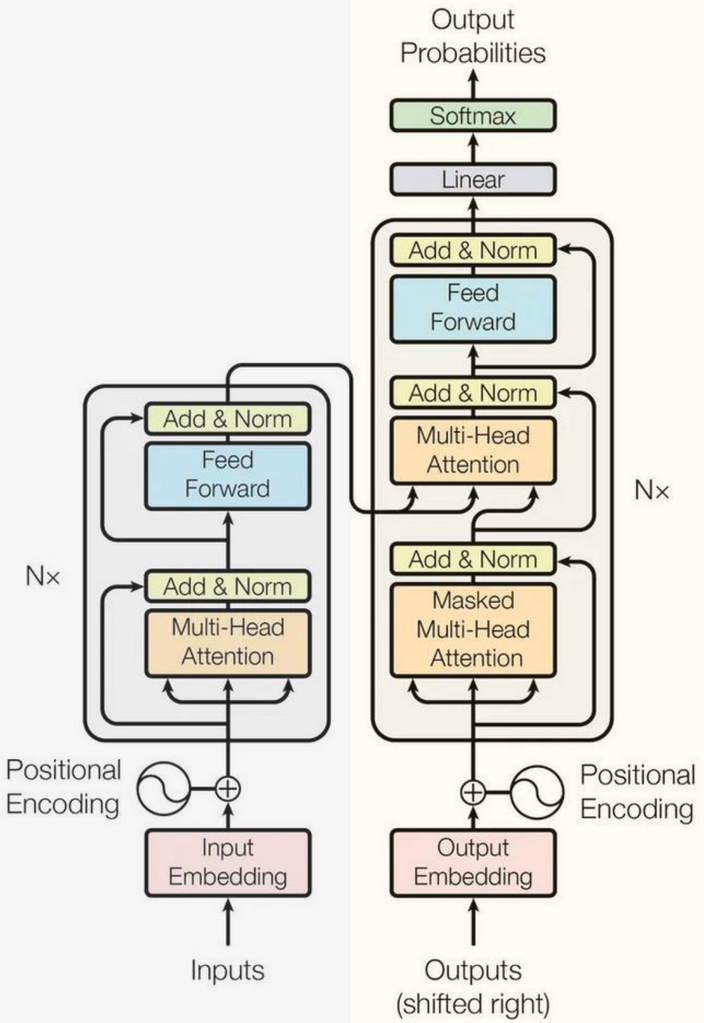
목차

Transformer

Encoder

Decoder

BERT 로 발전

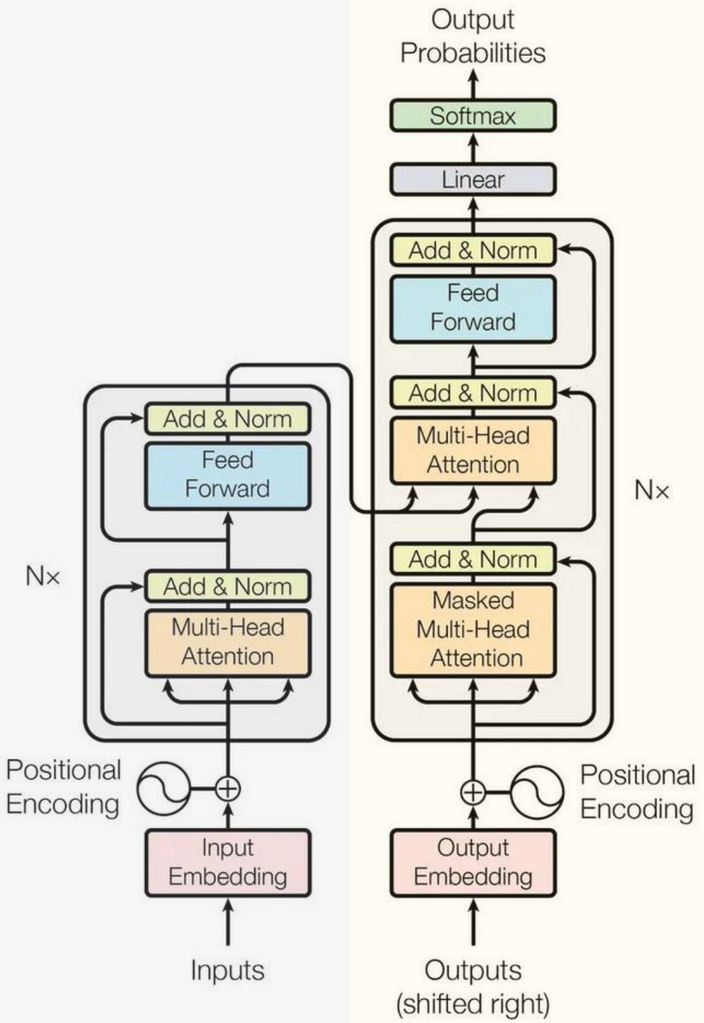


GPT 로 발전

Transformer

Encoder

입력 문맥을 깊이 있게
이해하고
정보를 압축하여
디코더에게 전달



Decoder

인코더에서 받은 정보를
바탕으로 다음 단어를
순차적으로 예측하고
출력하여 최종 결과 문장을 생성

BERT

Bidirectional
Encoder
Representations from
Transformer

BERT

Pre-trained Language Model을 Downstream Task에 적용하는 두 가지 주요 전이 학습

특징 기반 접근법

사전학습 모델을 고정된
특징 추출기(Feature Extractor)로 사용

장점: 빠른 학습
단점: 제한된 성능

→ ELMo, Word2Vec, GloVe

미세조정 접근법

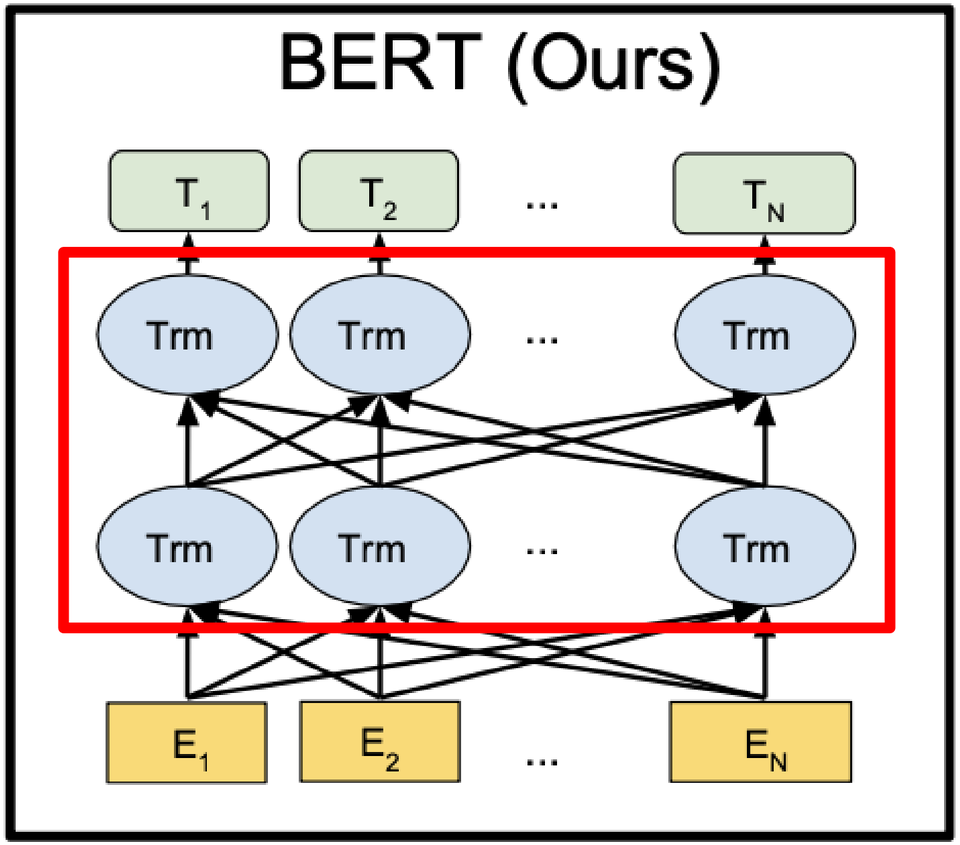
사전학습 모델의 가중치(weights)를
새로운 Task에 맞게 미세조정

장점: 높은 성능
단점: 높은 계산 비용

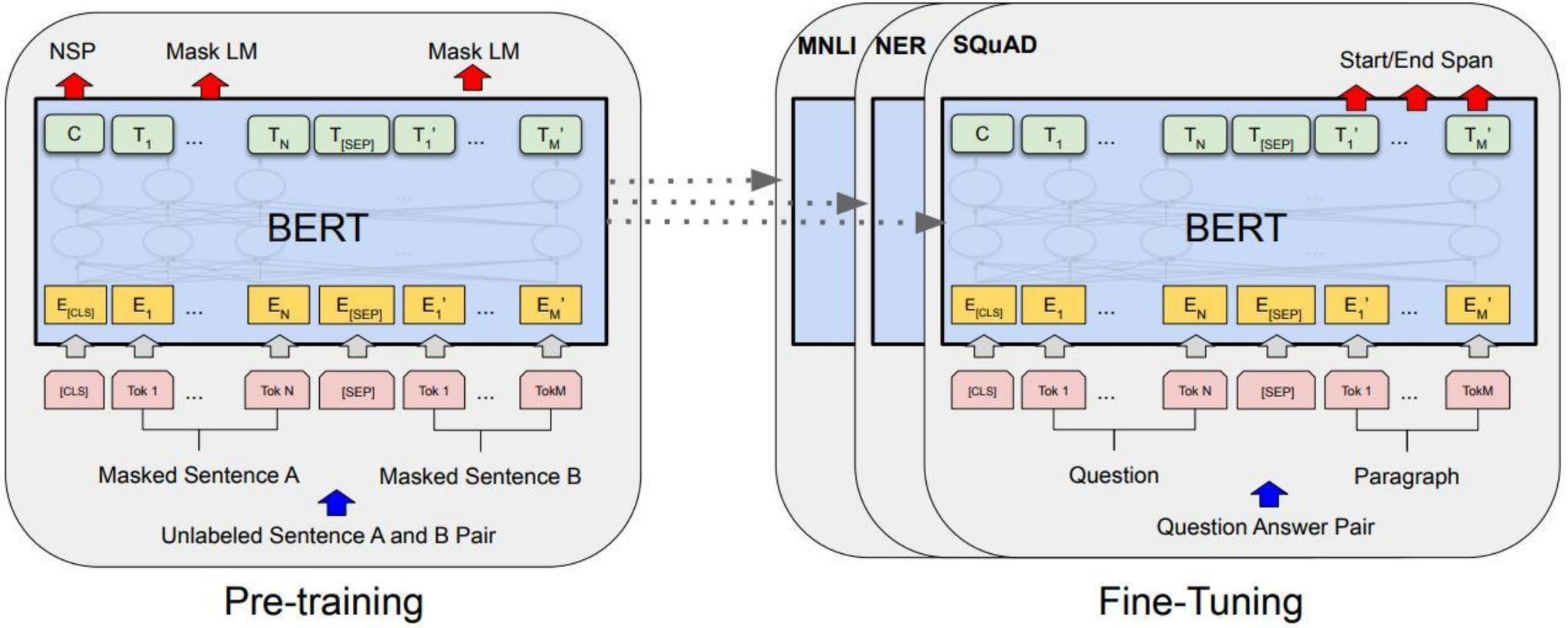
→ GPT-1, BERT가 사용하는 방식

아키텍처

- Transformer의 Encoder를 Multi-layer로 쌓은 구조
- 양방향 문맥(좌우 모두)을 동시에 학습
- 아키텍처 구조와 학습 파라미터 수
  - BERT Base: L=12, H=768, A=12 → 110M params
  - BERT Large: L=24, H=1024, A=16 → 340M params

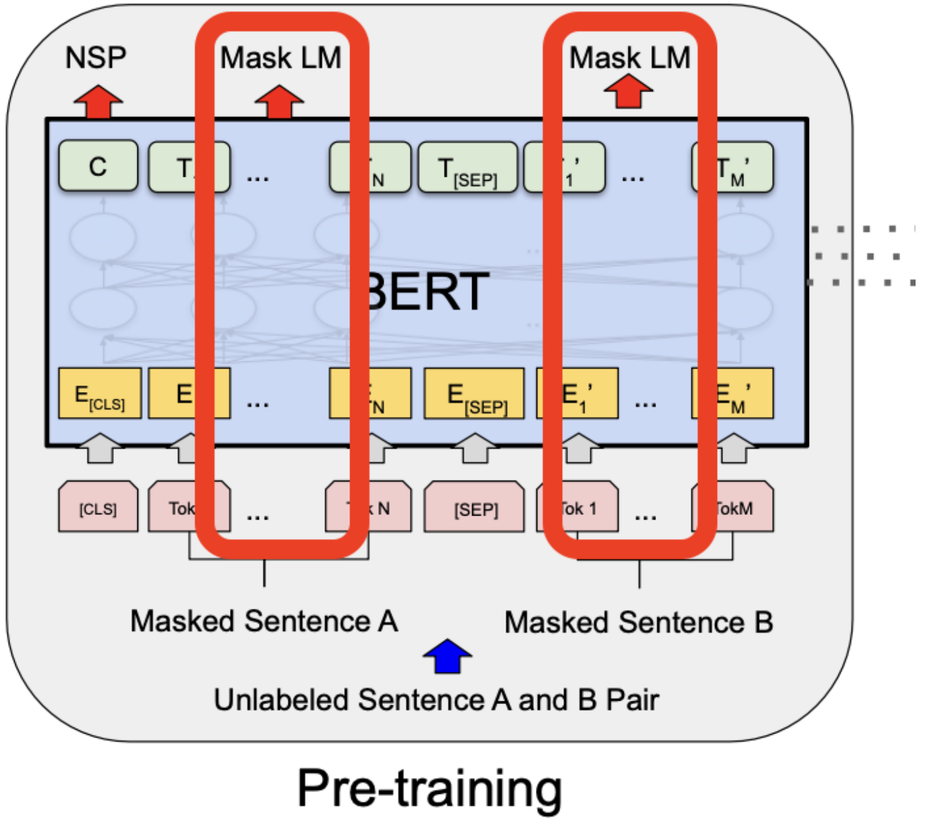


학습 방법



BERT 모델의 학습은 Pre-training 후 Fine-Tuning으로 Task에 맞춰 미세 조정하는 방식을 사용

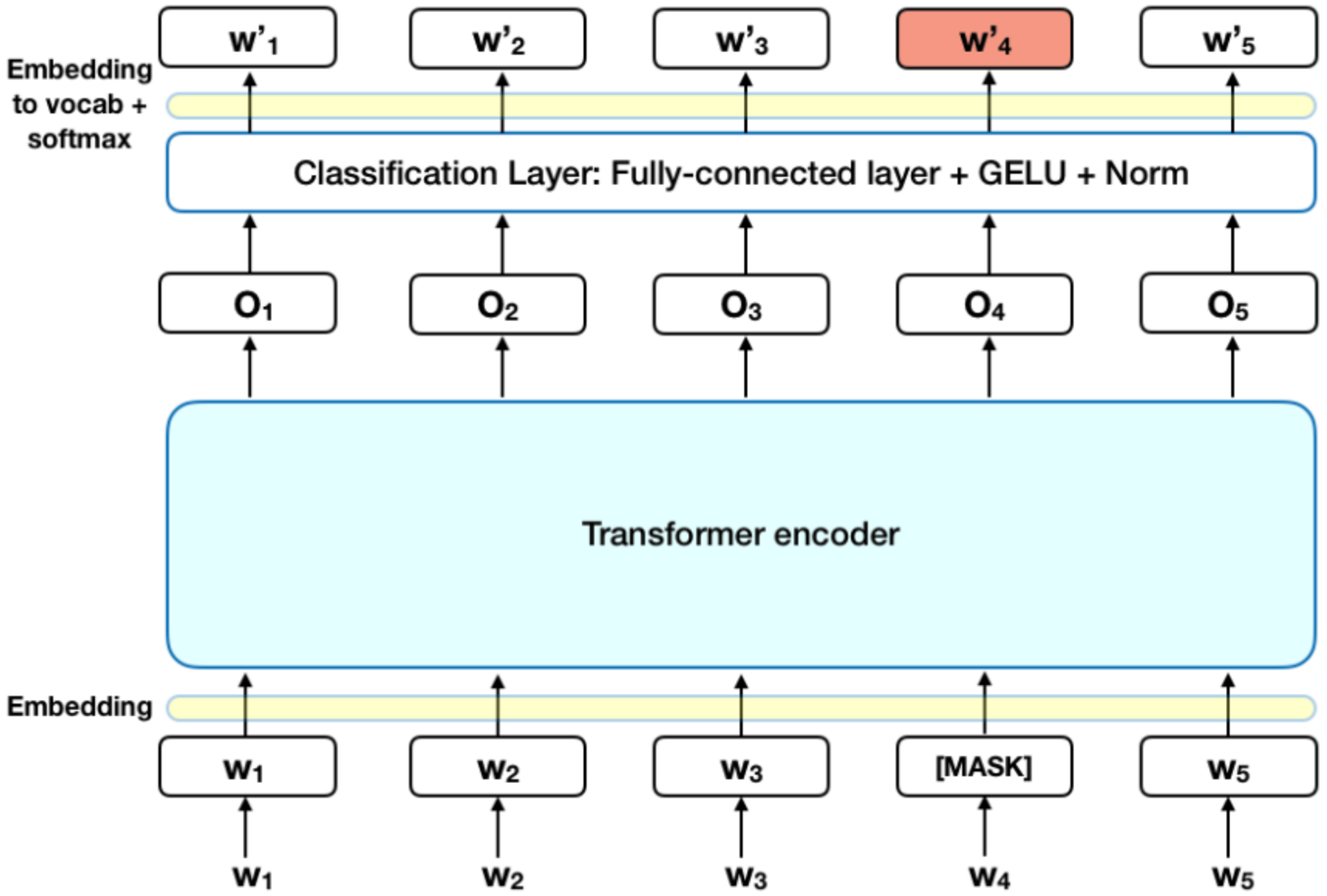
MLM(Masked Language Model)



Pre-training

양방향으로 학습하면서 여러 층인 모델의 구조적인 특징으로 인해 학습 과정에서 정보 유출이 발생
"다음 단어 예측“(조건부 확률)을 포기하는 대신, " 가려진 단어 맞추기" (Masked LM) 방식을 도입

Masked 토큰으로 인한 사전학습과 파인튜닝 간의 불일치(Mismatch)



BERT의 해결책: 80-10-10 규칙
입력 토큰의 15%를 선정
선정된 토큰 중

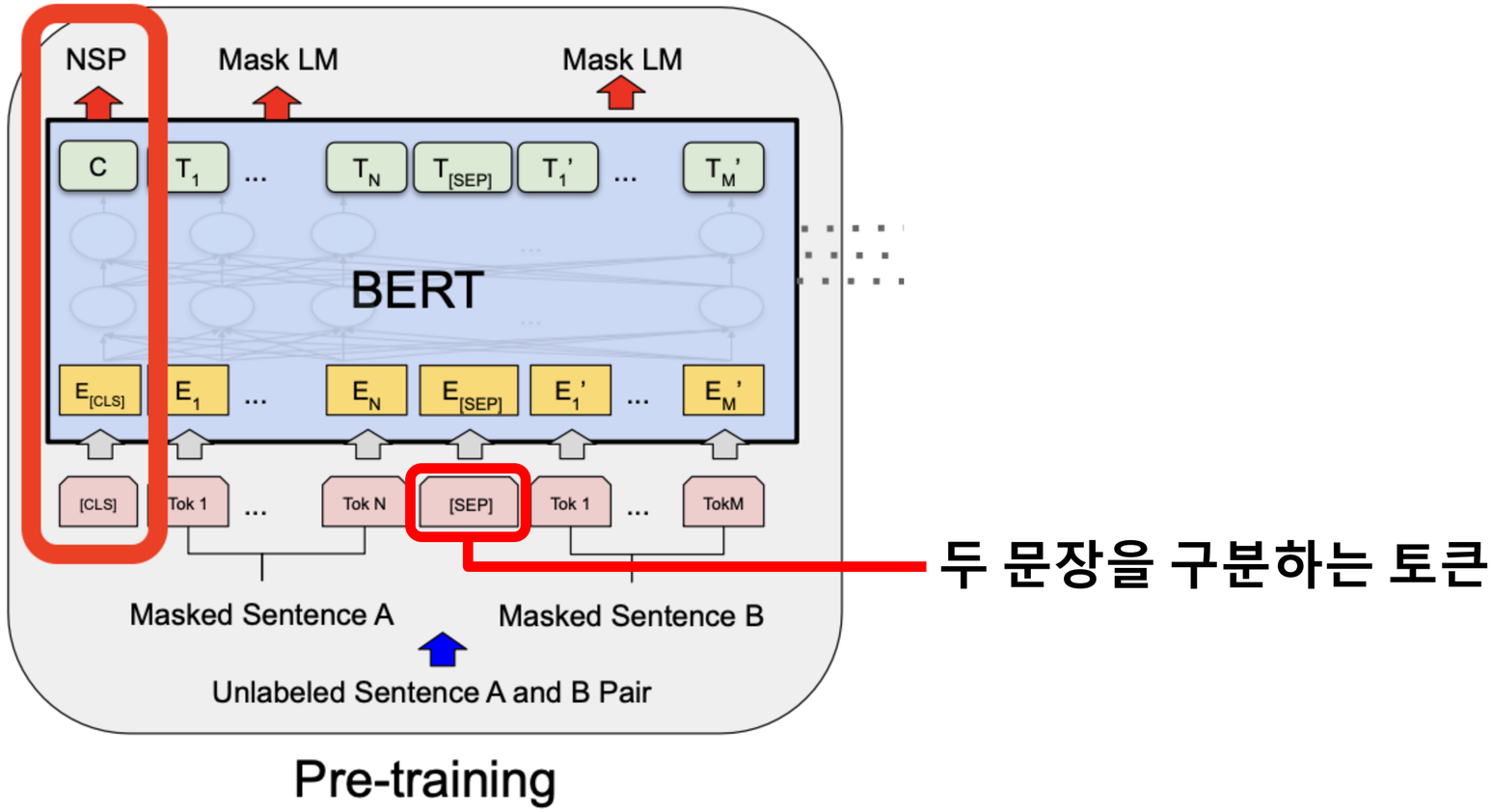
- 80%는 Masked 토큰으로 교체
- 10%는 무작위 토큰으로 교체
- 10%는 원본 그대로 유지

Masking Rates			Dev Set Results		
MASK	SAME	RND	MNLI	NER	
			Fine-tune	Fine-tune	Feature-based
80%	10%	10%	84.2	95.4	94.9
100%	0%	0%	84.3	94.9	94.0
80%	0%	20%	84.1	95.2	94.6
80%	20%	0%	84.4	95.2	94.7
0%	20%	80%	83.7	94.8	94.6
0%	0%	100%	83.6	94.9	94.6

Table 8: Ablation over different masking strategies.

NSP(Next Sequence Prediction)

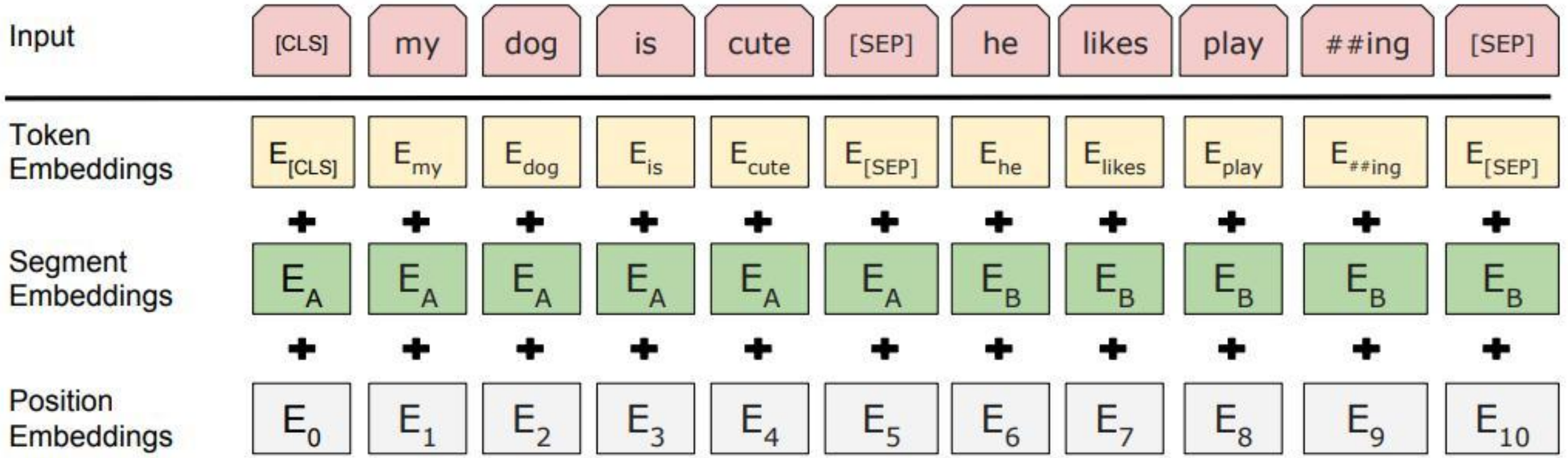
IsNext
or
NotNext



Pre-training

문장 사이의 관계를 학습할 수 있도록 두 개의 문장을 연결하여 입력으로 사용
주어진 두 개의 문장이 이어지는 문장인지 아닌지를 판단하는 Binary Classification 작업을 추가
이 과정에서 첫번째 [CLS] 토큰을 활용하여 Output Layer에서 예측하도록 학습

입력 표현

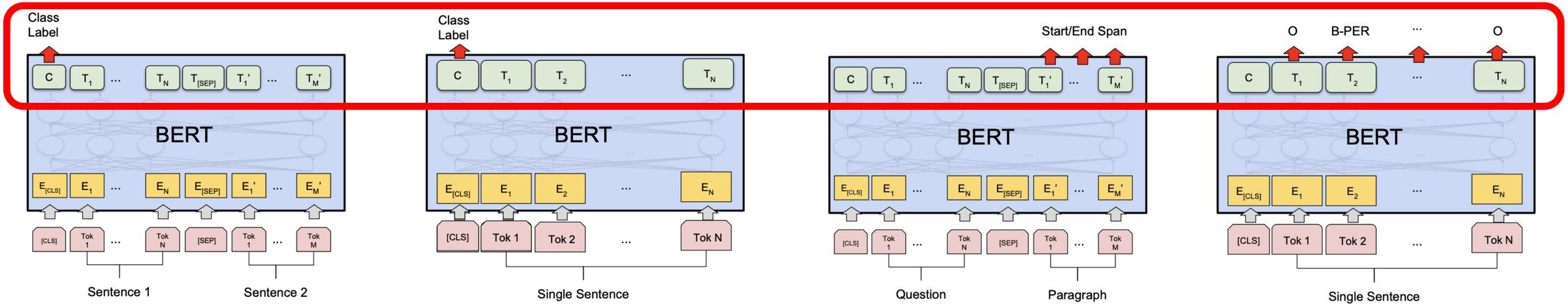


Token Embedding: WordPiece 규칙으로 문장을 토큰 단위([CLS], [MASK], [SEP] 포함)로 나누고 벡터화
Positional Embedding: 위치를 나타내는 벡터를 sin/cos함수로 계산
Segment Embedding: 토큰이 어느 문장에 속하는지 구분하기 위한 벡터(E_A, E_B)

→ 세 가지의 임베딩 모두 Pre-training 과정에서 학습되는 파라미터에 포함

BERT

파인튜닝



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

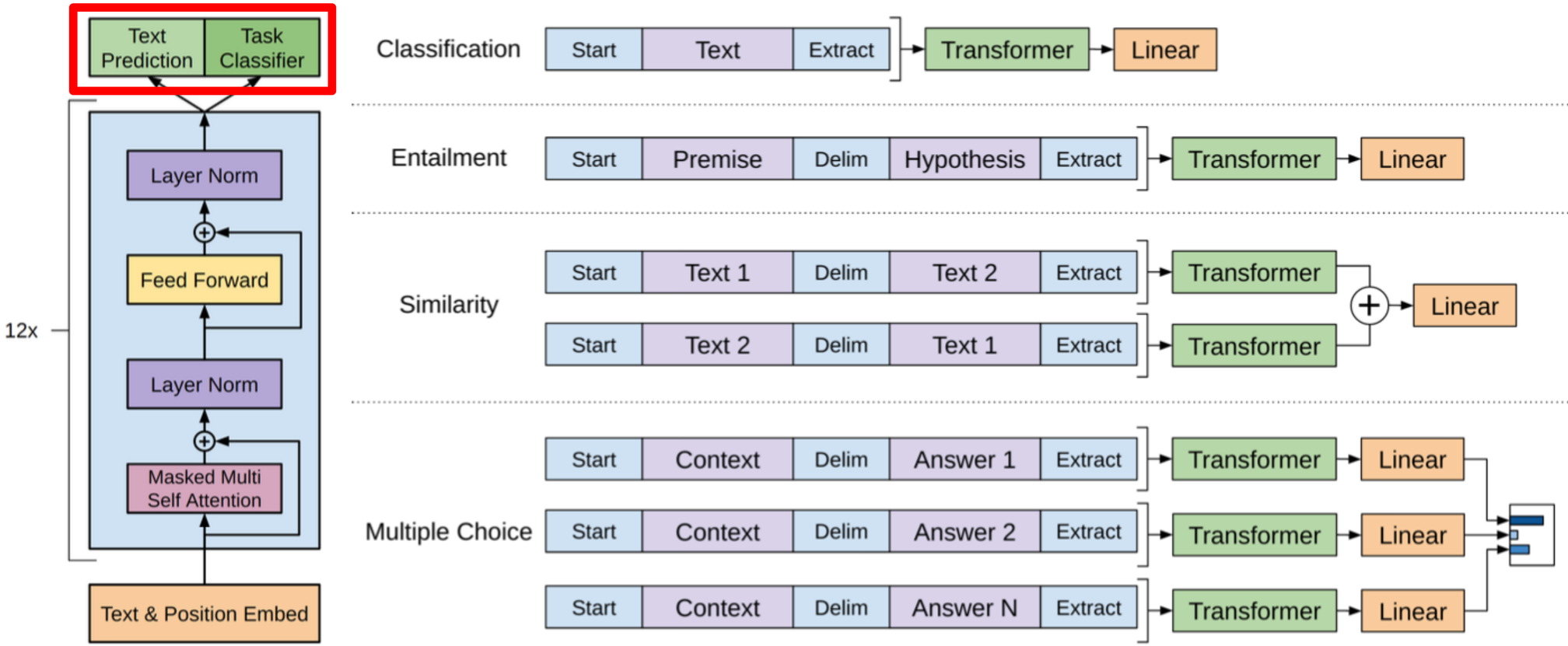
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Downstream task에 맞춰서 Output layer만 추가
사전 학습된 파라미터는 Frozen하여 진행 가능

GPT-1

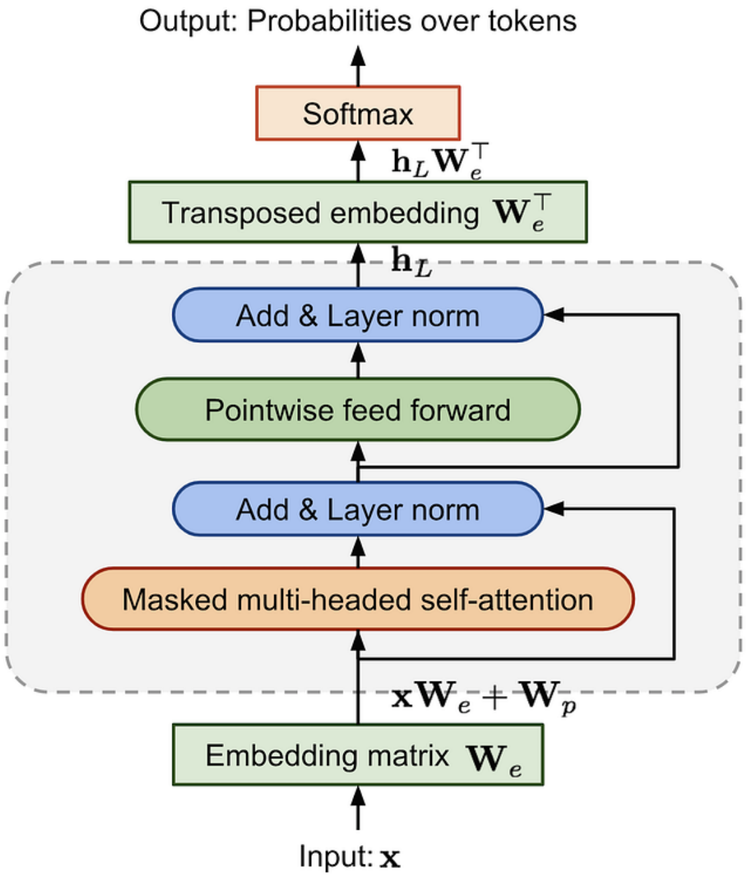
Generative
Pre-trained
Transformer

아키텍처



Transformer의 Decoder를 쌓은 구조(X 12)
BERT와 동일하게 Downstream task에 맞춰 Output layer를 추가하고 Fine-tuning

Pre-training



$L_1(u)$를 최대화하는 방향으로 사전 학습
라벨이 없는 Sequence로 비지도 학습

$$h_0 = UW_e + W_p$$

$$h_l = \texttt{transformer\_block}(h_{l-1}) \forall i \in [1, n]$$

$$P(u) = \texttt{softmax}(h_n W_e^T)$$

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta)$$

Fine-tuning

$$P(y|x^1,\ldots,x^m) = \texttt{softmax}(h_l^m W_y)$$

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1,\ldots,x^m)$$

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C})$$

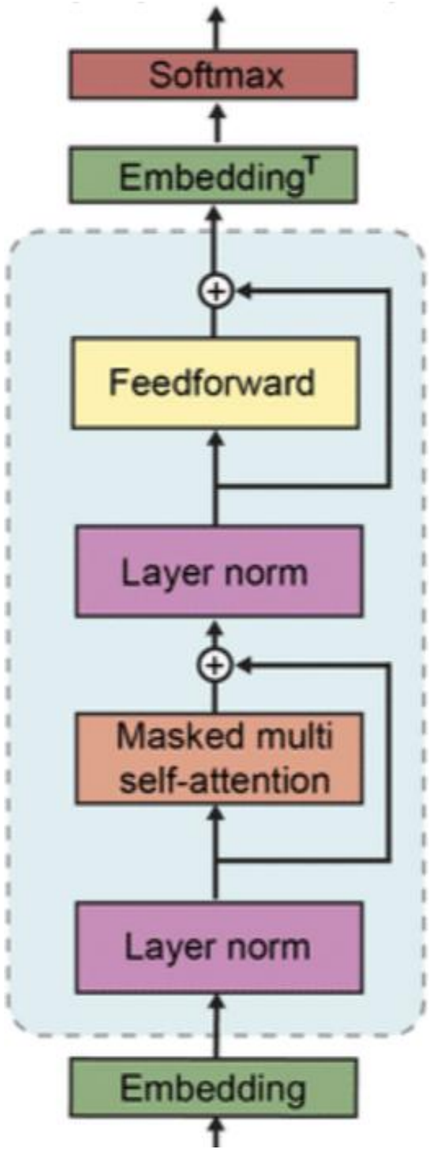
$\boldsymbol{L_2(C)}$는 Fine-tuning을 위한 목적 함수(Objective Function)
$\boldsymbol{L_3(C)}$는 사전 학습(다음 단어 예측)에 사용된 목적 함수로, 미세 조정 시 보조 훈련 목표로 사용

- 모델이 특정 Task에 과적합되는 것을 막아줌
- 모델이 새로운 과제를 더 빨리 학습

GPT-2

Generative
Pre-trained
Transformer

아키텍처



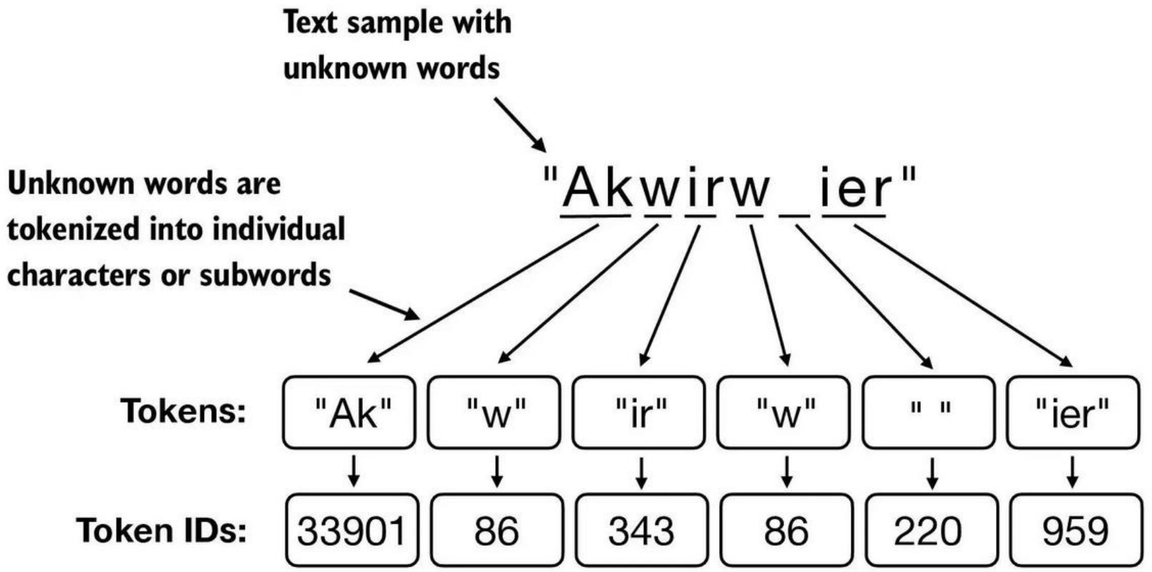
Layer normalization의 위치 변화

Post-LN → Pre-LN

훈련 안정성(Training Stability)을 크게 향상

Fine-tuning 과정 없이 비지도 학습만 진행하여 다양한 Downstream task를 수행하는 모델을 만드는 것이 목표

Byte Pair Encoding



자주 같이 나오는 글자들을 합쳐서 하나의 ‘토큰’으로 만드는 단어 분리 압축 알고리즘

처음 보는 단어(신조어, 오타 등)에 대처하기 위한 핵심 기술

General model 학습을 위한 데이터셋

”I’m not the cleverest man in the world, but like they say in French: **Je ne suis pas un imbecile [I’m not a fool].**
In a now-deleted post from Aug. 16, Soheil Eid, Tory candidate in the riding of Joliette, wrote in French: ”**Mentez mentez, il en restera toujours quelque chose**,” which translates as, ”**Lie lie and something will always remain.**”
“I hate the word ‘**perfume**,”’ Burr says. ‘It’s somewhat better in French: ‘**parfum**.’
If listened carefully at 29:55, a conversation can be heard between two guys in French: “**-Comment on fait pour aller de l’autre coté? -Quel autre coté?**”, which means “**- How do you get to the other side? - What side?**”.
If this sounds like a bit of a stretch, consider this question in French: **As-tu aller au cinéma?**, or **Did you go to the movies?**, which literally translates as Have-you to go to movies/theater?
“**Brevet Sans Garantie Du Gouvernement**”, translated to English: “**Patented without government warranty**”.

Table 1. Examples of naturally occurring demonstrations of English to French and French to English translation found throughout the WebText training set.

하나의 도메인에 치우치지 않고, 최대한 많은 데이터 확보

WebText는 사람이 직접 필터링을 진행

Reddit으로부터 최소 3개의 평가를 받은 외부 링크만을 사용

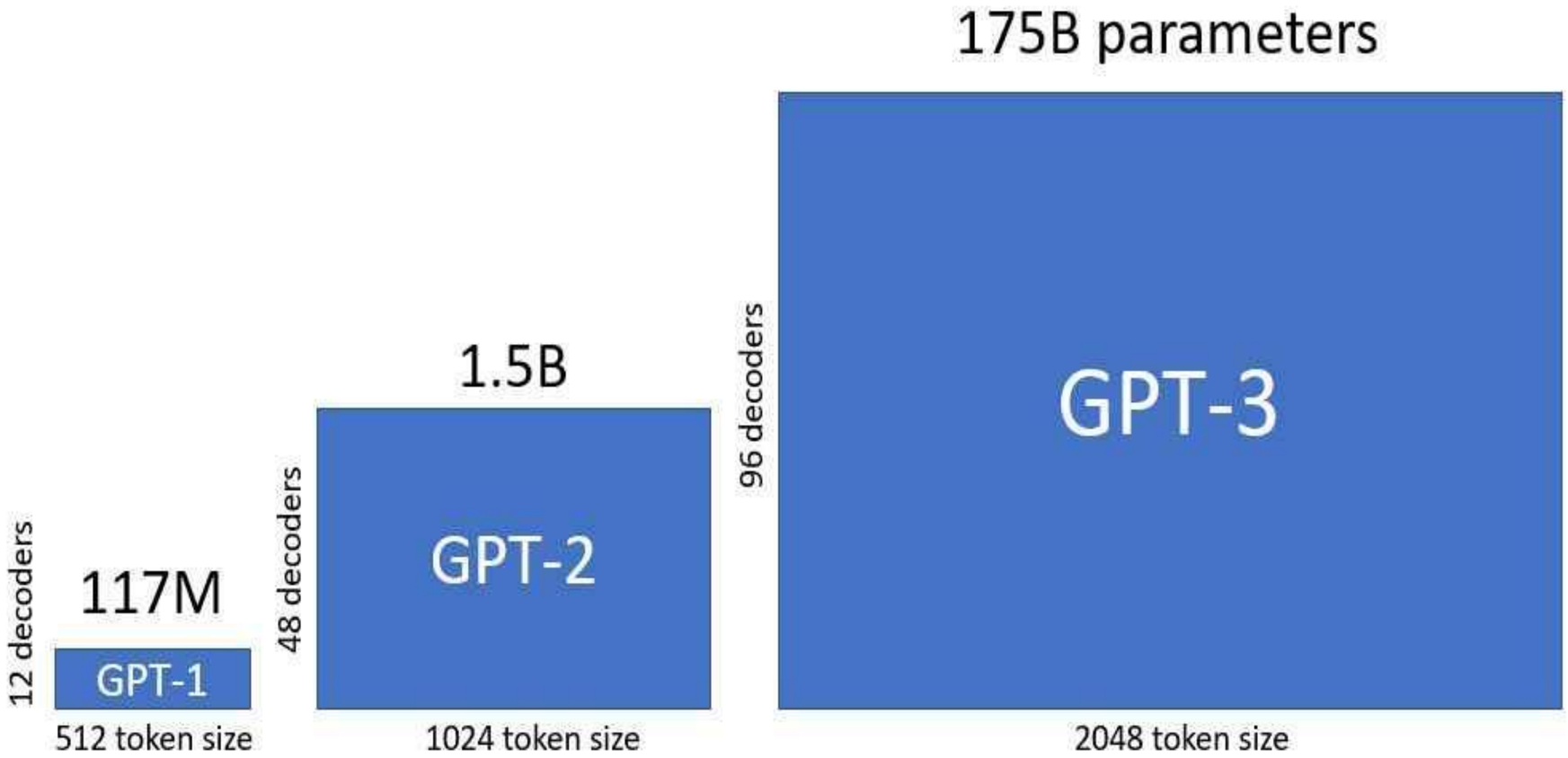
위키피디아와 같은 대중적인 문서를 제외

중복 제거 및 정제 후 40GB의 텍스트 확보

GPT-3

Generative
Pre-trained
Transformer

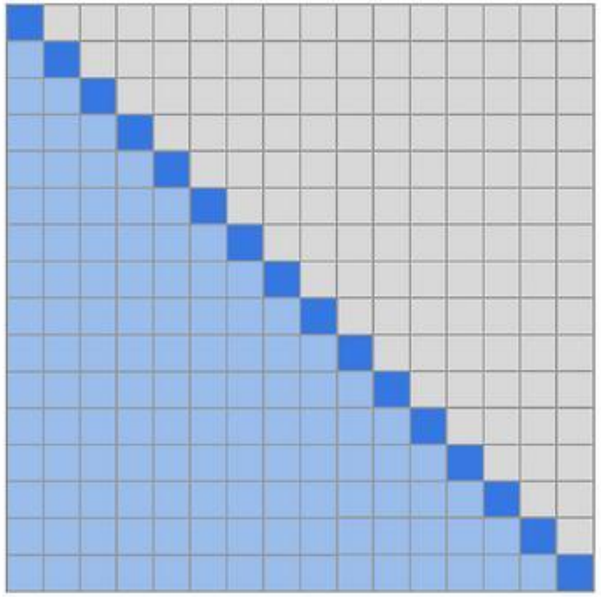
이전 모델보다 10배 이상 많은 1750억(175B) 개의 파라미터



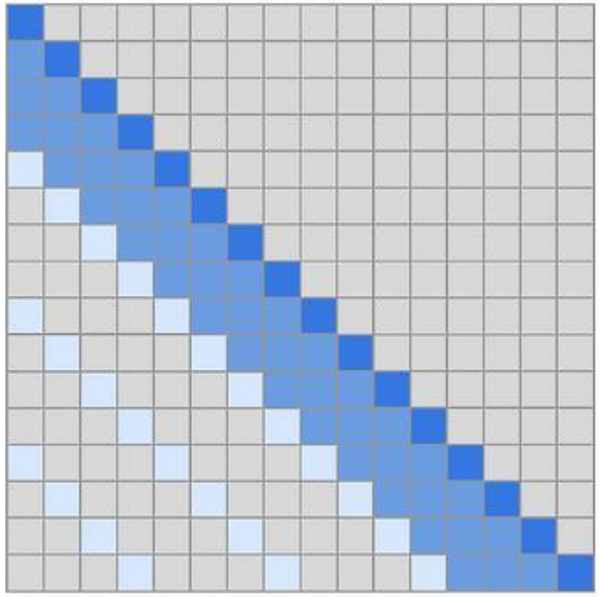
구조적 변화는 없이 모델 크기가 증가

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or “GPT-3”	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

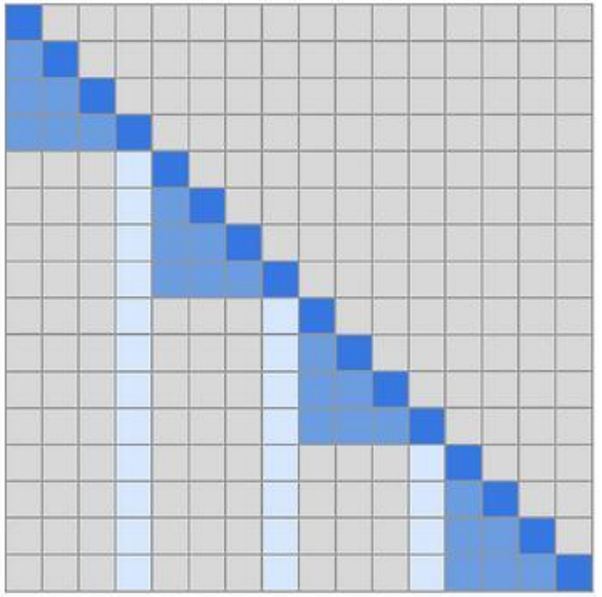
Sparse Attention



(a) Transformer



(b) Sparse Transformer (strided)

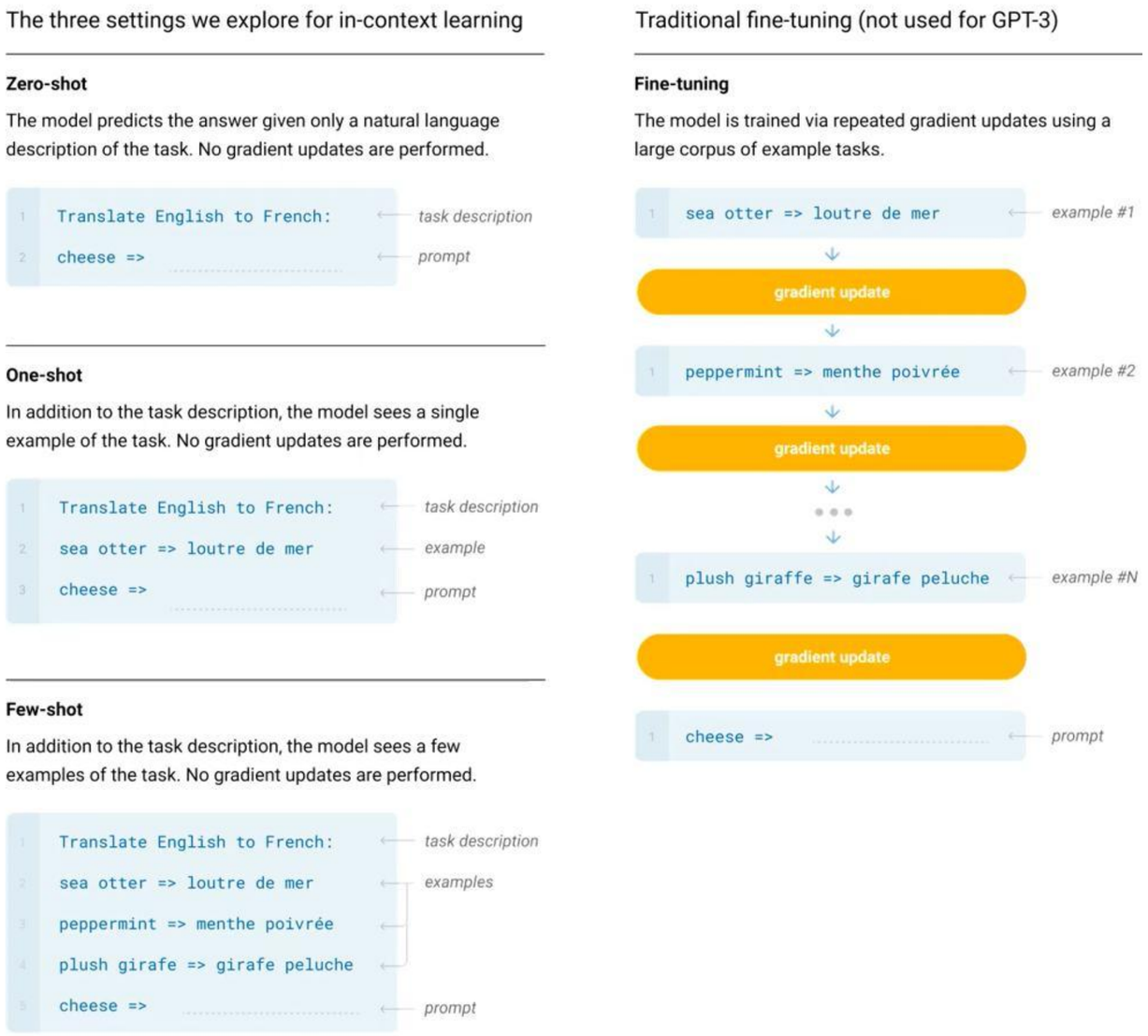


(c) Sparse Transformer (fixed)

Sparse Attention: 정해진 패턴에 따라 일부 관계만 계산

→ 연산량이 크게 감소

Zero-shot, One-shot, Few-shot



In-Context Learning
프롬프트만으로 새로운 작업을 수행

Zero-shot, One-shot, Few-shot 방식

- 별도의 학습 불필요
- 프롬프트를 통한 빠른 작업 수행
- 높은 데이터 효율성
- 범용성 및 유연성

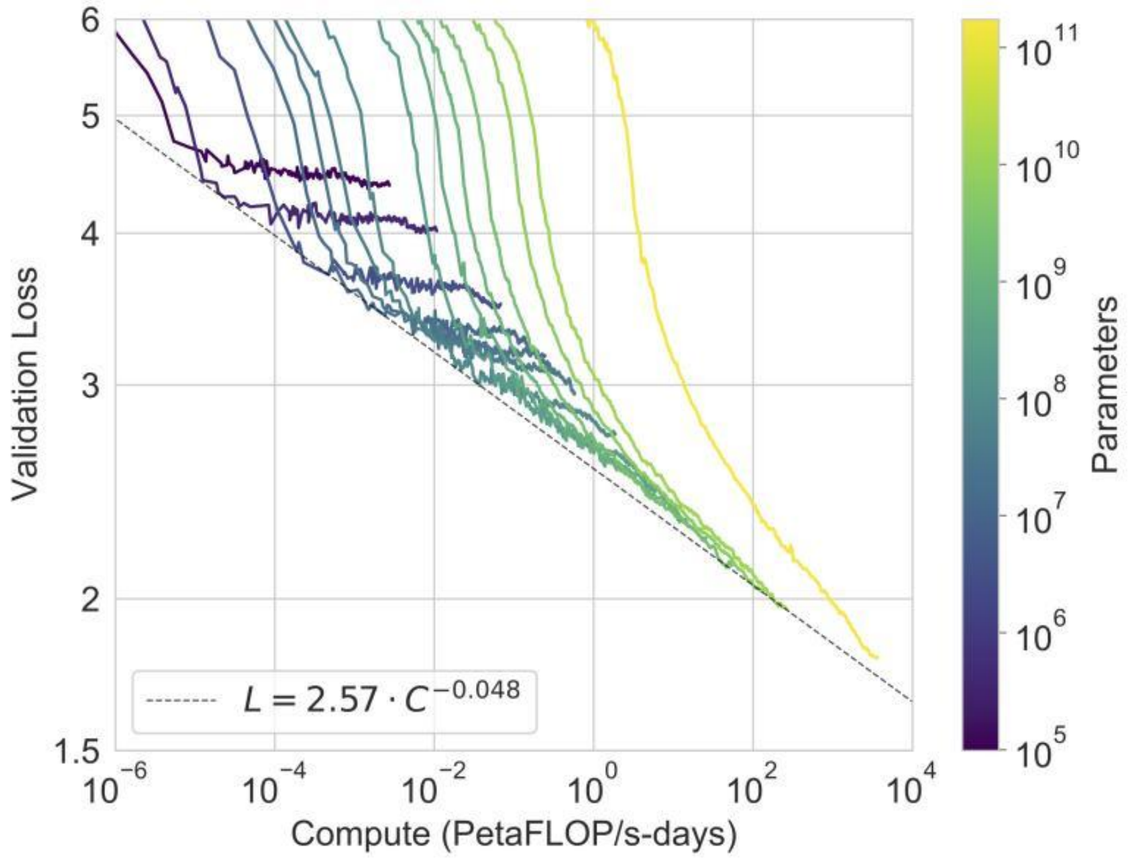
Training

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

훈련 데이터에서 연구에 사용된 모든 벤치마크의 학습 데이터셋 및 테스트 데이터셋과 겹치는 부분을 찾아 제거하려 시도
하지만 데이터를 필터링하는 과정에 버그가 있어 일부 중복 데이터를 놓침

이미 모델 훈련이 완료된 상태였고, 모델을 다시 훈련시키는 데는 엄청난 비용이 들기 때문에 재훈련은 진행 못함

Result



파라미터의 증가에 따라 정확도가 향상

구현 코드

구현 코드

K-V 캐시가 적용된 Masked Self-Attention 클래스

```
class Head(nn.Module):
    def __init__(self, head_size):
        super().__init__()
        self.ke
        self.qu
        self.va
        self.re
        self.dr

    def forward
        B, T, C
        k = sel

        if past
            pas
            k =
            v =

        k = k[:

        wei = q

        T_k = k.shape[-2]
        if T > 1:
            wei = wei.masked_fill(self.tril[:T, :T_k] == 0, float('-inf'))

        wei = F.softmax(wei, dim=-1)
        wei = self.dropout(wei)
        out = wei @ v
        return out, (k, v)
```

```
class MultiHeadAttention(nn.Module):
    def __init__(self, num_heads, head_size):
        super().__init__()
        self.heads = nn.ModuleList([Head(head_size) for _ in range(num_heads)])
        self.proj = nn.Linear(n_embd, n_embd)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, past_kv_list=None):
        head_outputs, new_kv_list = [], []
        for i, h in enumerate(self.heads):
            past_kv = past_kv_list[i] if past_kv_list is not None else None
            out, new_kv = h(x, past_kv=past_kv)
            head_outputs.append(out); new_kv_list.append(new_kv)

        out = torch.cat(head_outputs, dim=-1)
        out = self.dropout(self.proj(out))
        return out, new_kv_list
```

```
class FeedForward(nn.Module):
    def __init__(self, n_embd):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(n_embd, 4 * n_embd), nn.GELU(),
            nn.Linear(4 * n_embd, n_embd), nn.Dropout(dropout),
        )
    def forward(self, x):
        return self.net(x)
```

```
, block_size)))
```

구현 코드

Multi-Head Attention & Feedforward 클래스

```python
class MultiHeadAttention(nn.Module):
    def __init__(self, num_heads, head_size):
        super().__init__()
        self.heads = nn.ModuleList([Head(head_size) for _ in range(num_heads)])
        self.proj = nn.Linear(n_embd, n_embd)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, past_kv_list=None):
        head_outputs, new_kv_list = [], []
        for i, h in enumerate(self.heads):
            past_kv = past_kv_list[i] if past_kv_list is not None else None
            out, new_kv = h(x, past_kv=past_kv)
            head_outputs.append(out); new_kv_list.append(new_kv)

        out = torch.cat(head_outputs, dim=-1)
        out = self.dropout(self.proj(out))
        return out, new_kv_list

class FeedForward(nn.Module):
    def __init__(self, n_embd):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(n_embd, 4 * n_embd), nn.GELU(),
            nn.Linear(4 * n_embd, n_embd), nn.Dropout(dropout),
        )
    def forward(self, x):
        return self.net(x)
```

구현 코드

Multi-Head Attention, Feedforward, Decoder Block 클래스

```python
class MultiHeadAttention(nn.Module):
    def __init__(self, num_heads, head_size):
        super().__init__()
        self.heads = nn.ModuleList([Head(head_size) for _ in range(num_heads)])
        self.proj = nn.Linear(n_embd, n_embd)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, past_kv_list=None):
        head_outputs, new_kv_list = [], []
        for i, h in enumerate(self.heads):
            past_kv = past_kv_list[i] if past_kv_list is not None else None
            out, new_kv = h(x, past_kv=past_kv)
            head_outputs.append(out); new_kv_list.append(new_kv)

        out = torch.cat(head_outputs, dim=-1)
        out = self.dropout(self.proj(out))
        return out, new_kv_list

class FeedForward(nn.Module):
    def __init__(self, n_embd):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(n_embd, 4 * n_embd), nn.GELU(),
            nn.Linear(4 * n_embd, n_embd), nn.Dropout(dropout),
        )
    def forward(self, x):
        return self.net(x)
```

```python
class Block(nn.Module):
    def __init__(self, n_embd, n_head):
        super().__init__()
        head_size = n_embd // n_head
        self.sa = MultiHeadAttention(n_head, head_size)
        self.ffwd = FeedForward(n_embd)
        self.ln1 = nn.LayerNorm(n_embd)
        self.ln2 = nn.LayerNorm(n_embd)

    def forward(self, x, past_kv_list=None):
        sa_out, new_sa_kv_list = self.sa(self.ln1(x), past_kv_list=past_kv_list)
        x = x + sa_out
        x = x + self.ffwd(self.ln2(x))
        return x, new_sa_kv_list
```

구현 코드

Mini-GPT 클래스

```python
class MiniGPTModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.token_embedding_table = nn.Embedding(vocab_size, n_embd)
        self.position_embedding_table = nn.Embedding(block_size, n_embd)
        self.blocks = nn.ModuleList([Block(n_embd, n_head=n_head) for _ in range(n_layer)])
        self.ln_f = nn.LayerNorm(n_embd)
        self.lm_head = nn.Linear(n_embd, vocab_size)

    def forward(self, idx, targets=None, past_key_values=None):
        B, T = idx.shape
        past_length = 0
        if past_key_values is not None:
            past_length = past_key_values[0][0][0].shape[-2]

        if T == 1 and past_key_values is not None:
            pos = torch.tensor([past_length % block_size], dtype=torch.long, device=device)
        else:
            pos = torch.arange(0, T, dtype=torch.long, device=device)

        tok_emb = self.token_embedding_table(idx)
        pos_emb = self.position_embedding_table(pos)
        x = tok_emb + pos_emb

        new_past_key_values = []
        for i, block in enumerate(self.blocks):
            past_kv_list_for_block = past_key_values[i] if past_key_values is not None else None
            x, new_kv_list = block(x, past_kv_list=past_kv_list_for_block)
            new_past_key_values.append(new_kv_list)

        x = self.ln_f(x)
        logits = self.lm_head(x)
```

참고 자료

Improving Language Understanding by Generative Pre-Training (GPT-1), 2018년 Publication: OpenAI Technical Report Authors: Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever

Language Models are Unsupervised Multitask Learners (GPT-2), 2019년 Publication: OpenAI Technical Report Authors: Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever

Language Models are Few-Shot Learners (GPT-3), 2020년 Publication: arXiv Authors: Tom B. Brown et al.

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (BERT), 2019년 Publication: arXiv Authors: Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova

Addressing "Documentation Debt" in Machine Learning Research: A Retrospective Datasheet for BookCorpus, 2021년 Publication: Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS) Authors: Jack Bandy, Nicholas Vincent

SWAG: A Large-Scale Adversarial Dataset for Grounded Commonsense Inference, 2018년 Publication: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP) Authors: Rowan Zellers, Yonatan Bisk, Roy Schwartz, Yejin Choi

SQuAD: 100,000+ Questions for Machine Comprehension of Text, 2016년 Publication: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP) Authors: Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, Percy Liang

Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition, 2003년 Publication: Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 Authors: Erik F. Tjong Kim Sang, Fien De Meulder

Generating Long Sequences with Sparse Transformers, 2019년 Publication: arXiv Authors: Rewon Child, Scott Gray, Alec Radford, Ilya Sutskever

https://jalammar.github.io/illustrated-gpt2/

Q&A