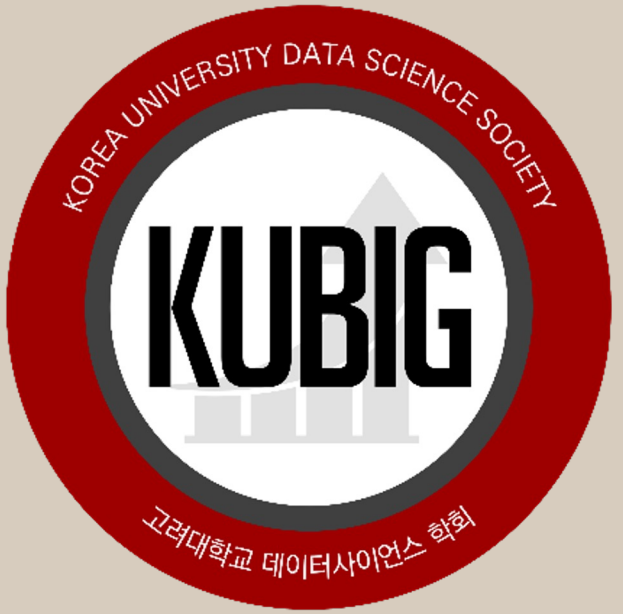
KUBIG 24-W
겨울방학 BASIC STUDY SESSION

NLP SESSION
WEEK5



Today's Session Leader: 17기 홍여빈

CONTENTS

KUBIG
DATA SCIENCE & AI

01 Announcement, 우수 복습과제 Review

1-1. Announcement

KUBIG CONTEST 2월 29일 예정.

1, 3, 4팀 한솔데코 대회 출전
2팀 별도 주제
팀별 중간발표 2월 15일 session 종료 직후 팀당 3~5분 분량으로 간단히 발표

1-2. 우수 복습과제 코드 Review

KUBIG DATA SCIENCE & AI

정준님

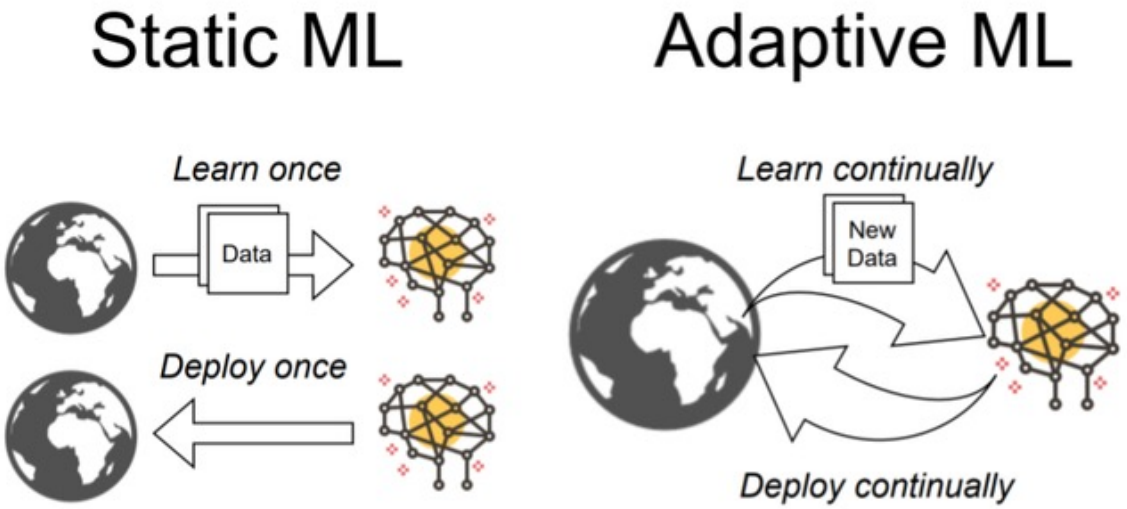
4주차
복습과제1

Transformer 챗봇 구현

화면공유 하셔서 3분 내외로 가볍게 리뷰해주시면 됩니다!

02 BERT

Fine-tuning paradigm의 시작

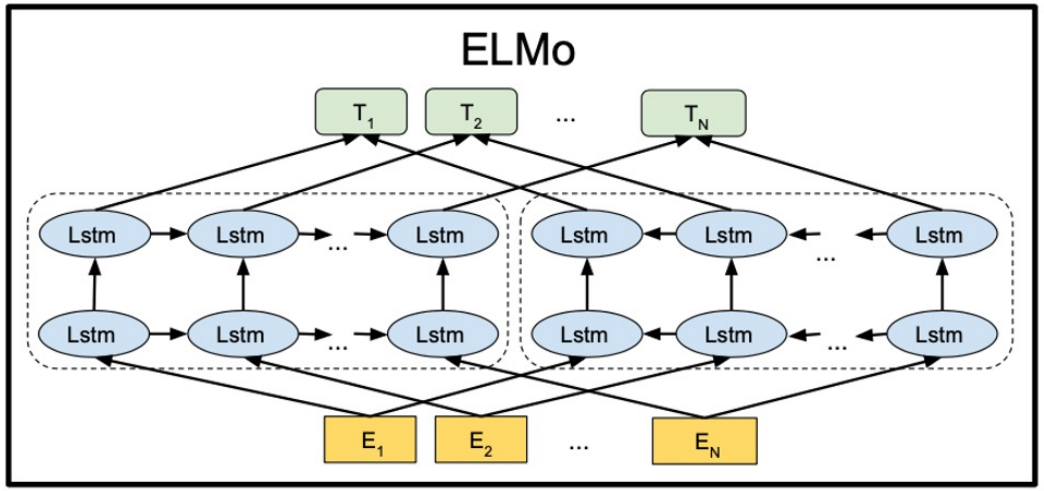


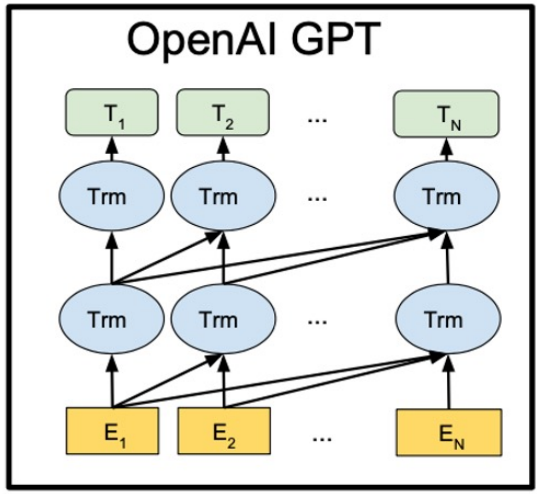
1. Adapter-based approach ex) LoRA
2. Replay-based approach
3. Feature-based approach
4. Representation-based approach ex) fine-tuning
5. Regularization-based approach ex) SPG

...

Continual Learning

이전 데이터에 대한 일반화 기능을 계속 유지하면서 새로운 데이터에 대해서도 잘 일반화할 수 있도록 학습





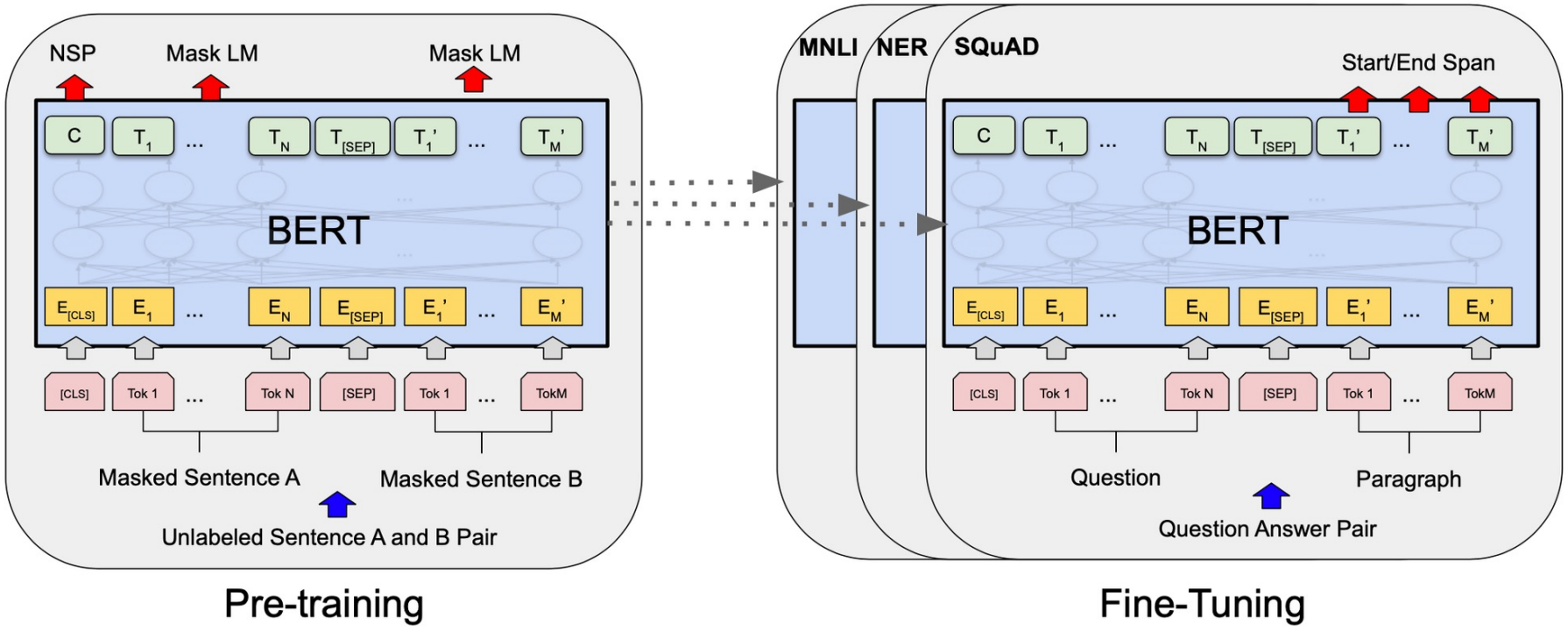
Feature-based (ELMo)

context-sensitive feature를 양방향 LSTM에서 각각 추출, hidden layers를 concat하는 방식.
특정 단어에 쌓인 embedding들을 선형 결합하여 가중치를 조절.

Fine-tuning

unlabeled text에서 pre-train되고 supervised down-stream task에서 fine-tuning.
모든 parameter를 update.

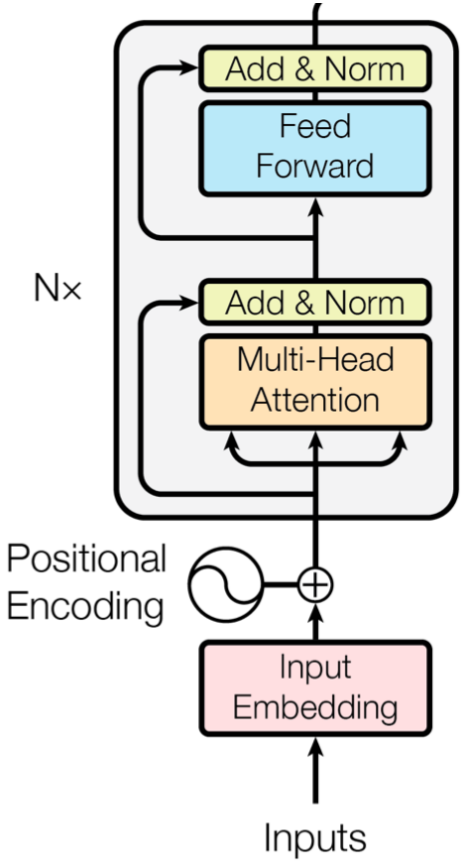
2-2. Fine-tuning paradigm



레이블이 지정되지 않은 텍스트에서 심층적인 양방향 contextual representation을 사전 학습 (unsupervised learning)

다양한 downstream task를 미세 조정

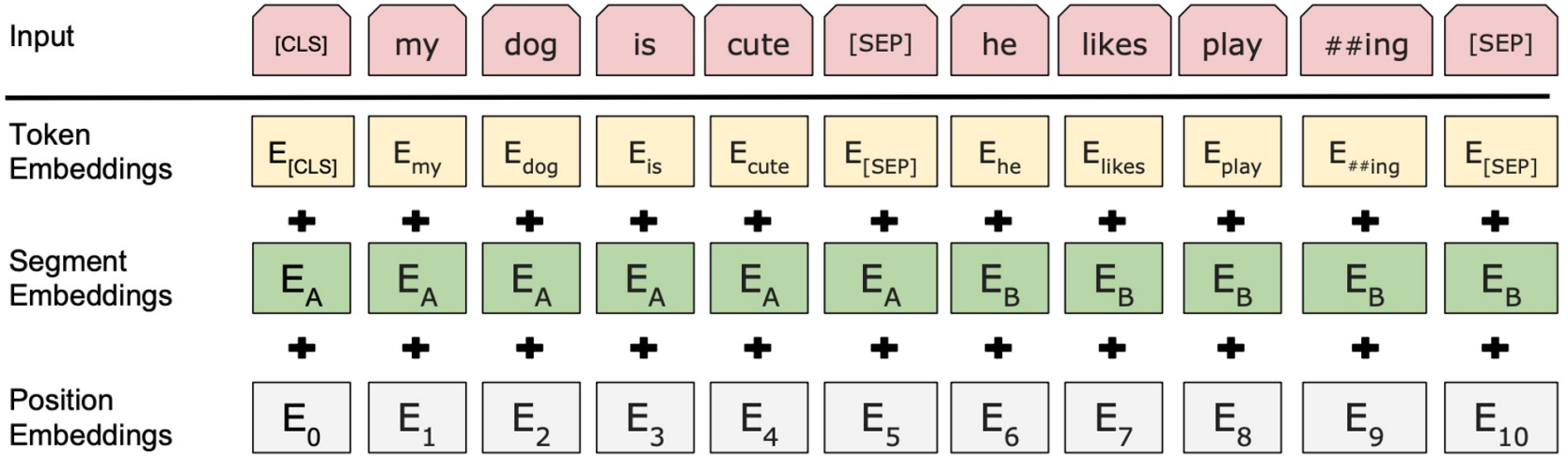
BERT (Bidirectional Encoder Representations from Transformers)



- 초기 트랜스포머 모델 : L = 6, D = 512, A = 8
- BERT-Base : L=12, D=768, A=12
- BERT-Large : L=24, D=1024, A=16

Input of BERT
sentence / sequence

- "sentence": 실제 언어 문장이 아닌 임의의 연속 텍스트 span
- "sequence": 단일 문장이거나 함께 묶인 두 문장



- Token Embeddings: WordPiece embedding을 사용. 첫 번째 토큰은 [CLS]이고 문장 쌍을 사용할 때는 분리 토큰 [SEP]을 포함.
- Segment Embeddings: 토큰이 문장 A 또는 B에 속하는지
- Position Embeddings: Transformer와 비슷한 positional encoding

2-4. Embedding of BERT

KUBIG DATA SCIENCE & AI

```python
class BertEmbeddings(nn.Module):
    """Construct the embeddings from word, position and token_type embeddings."""

    def __init__(self, config):
        super().__init__()
        self.word_embeddings = nn.Embedding(config.vocab_size, config.hidden_size, padding_idx=config.pad_token_id)
        self.position_embeddings = nn.Embedding(config.max_position_embeddings, config.hidden_size)
        self.token_type_embeddings = nn.Embedding(config.type_vocab_size, config.hidden_size)

        # self.LayerNorm is not snake-cased to stick with TensorFlow model variable name and be able to load
        # any TensorFlow checkpoint file
        self.LayerNorm = nn.LayerNorm(config.hidden_size, eps=config.layer_norm_eps)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        # position_ids (1, len position emb) is contiguous in memory and exported when serialized
        self.position_embedding_type = getattr(config, "position_embedding_type", "absolute")

    def forward(
        self,
        input_ids: Optional[torch.LongTensor] = None,
        token_type_ids: Optional[torch.LongTensor] = None,
        position_ids: Optional[torch.LongTensor] = None,
        inputs_embeds: Optional[torch.FloatTensor] = None,
        past_key_values_length: int = 0,
    ) -> torch.Tensor:

        embeddings = inputs_embeds + token_type_embeddings
        if self.position_embedding_type == "absolute":
            position_embeddings = self.position_embeddings(position_ids)
            embeddings += position_embeddings
        embeddings = self.LayerNorm(embeddings)
        embeddings = self.dropout(embeddings)
        return embeddings
```

2-4. Embedding of BERT

```
# dictionary
l o w : 5,  l o w e r : 2,  n e w e s t : 6,  w i d e s t : 3
```

Dictionary: 어떤 훈련 데이터로부터 각 단어들의 빈도수를 카운트하고 각 단어와 각 단어의 빈도수가 기록되어져 있는 결과를 기록한 것

```
# vocabulary
l, o, w, e, r, n, s, t, i, d
```

테스트 과정에서 'lowest'란 단어가 등장한다면 기계는 이 단어를 학습한 적이 없으므로 해당 단어에 대해서 제대로 대응하지 못하는 OOV 문제가 발생

Update 1

```
# dictionary update!
l o w : 5,
l o w e r : 2,
n e w es t : 6,
w i d es t : 3
```

```
# vocabulary update!
l, o, w, e, r, n, s, t, i, d, es
```

Update 2

```
# dictionary update!
l o w : 5,
l o w e r : 2,
n e w est : 6,
w i d est : 3
```

```
# vocabulary update!
l, o, w, e, r, n, s, t, i, d, es, est
```

Update 3

```
# dictionary update!
lo w : 5,
lo w e r : 2,
n e w est : 6,
w i d est : 3
```

Copy

```
# vocabulary update!
l, o, w, e, r, n, s, t, i, d, es, est, lo
```

2-4. Embedding of BERT

Update 10

```
# vocabulary update!
l, o, w, e, r, n, s, t, i, d, es, est, lo, low, ne, new, newest, wi, wid, widest
```

- 테스트 과정에서 'lowest'란 단어가 등장한다면, 기존에는 OOV에 해당되는 단어가 되었겠지만 BPE 알고리즘을 사용한 위의 단어 집합에서는 더 이상 'lowest'는 OOV가 아님.
- 'lowest'를 'low'와 'est' 두 단어로 인코딩 -> 이 두 단어는 둘 다 단어 집합에 있는 단어이므로 OOV가 아님.

2-4. Embedding of BERT

KUBIG DATA SCIENCE & AI

BERT tokenizer

```python
tokens = tokenizer("I love NLP!")

print(f"input ids : {tokens['input_ids']}")
print(f"token type ids : {tokens['token_type_ids']}")
print(f"attention mask : {tokens['attention_mask']}")
```

[실행 결과]

```
input ids : [101, 1045, 2293, 17953, 2361, 999, 102]
token type ids : [0, 0, 0, 0, 0, 0, 0]
attention mask : [1, 1, 1, 1, 1, 1, 1]
```

•input_ids : token들의 id 리스트(sequence of token id)
•token_type_ids : 각 token이 어떤 문장에 속하는지를 나타내는 리스트. sentence A에 속하는 token에는 0을, sentence B에 속하는 token에는 1을 부여.
•attention_mask : attention 연산이 수행되어야 할 token과 무시해야 할 token을 구별하는 정보가 담긴 리스트. attention 연산이 수행되어야 할, 일반적인 token에는 1을 부여하고, padding과 같이 attention 연산이 수행될 필요가 없는 token들에는 0을 부여.

2-4. Embedding of BERT

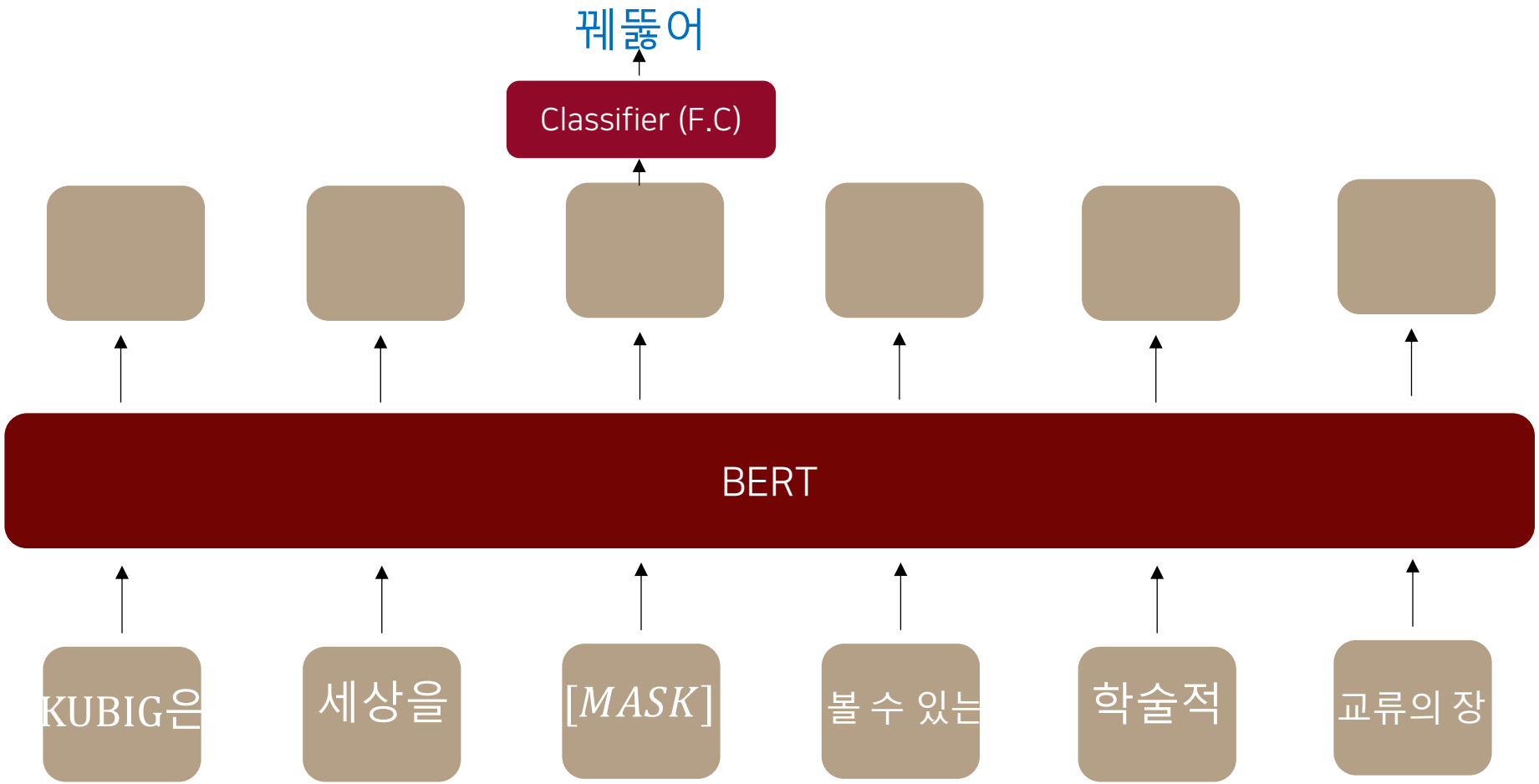
```python
print(tokenizer.convert_ids_to_tokens(1045))  # 하나만 바꿀 수도 있고
print(tokenizer.convert_ids_to_tokens([101, 1045, 2293, 17953, 2361, 999, 102]))  # 여러 개를 바꿀 수도
```

[실행 결과]

```
i
['[CLS]', 'i', 'love', 'nl', '##p', '!', '[SEP]']
```

- [PAD] - 0
- [UNK] - 100
- [CLS] – 101 -> BERT가 분류 문제를 풀기위한 특별 토큰
- [SEP] – 102 -> 두 문장을 분리
- [MASK] - 103

Bidirectional conditioning을 통해 multi-layered context에서 maksed token을 예측



2-5. Pre-training BERT 1

BERT는 입력 토큰의 15%를 무작위로 마스킹한 다음 마스킹된 토큰만 예측.
Fine-tuning 시에는 [MASK] 토큰이 나타나지 않기 때문에 pre-train과 fine-tuning 사이에 mismatch가 발생하는 문제가 있어서 다음과 같은 방법을 사용.

- 80% : 단어를 [MASK] 토큰으로 교체

쿠빅은 세상을 꿰뚫어 볼 수 있는 학술적 교류의 장 → 쿠빅은 세상을 [MASK] 볼 수 있는 학술적 교류의 장

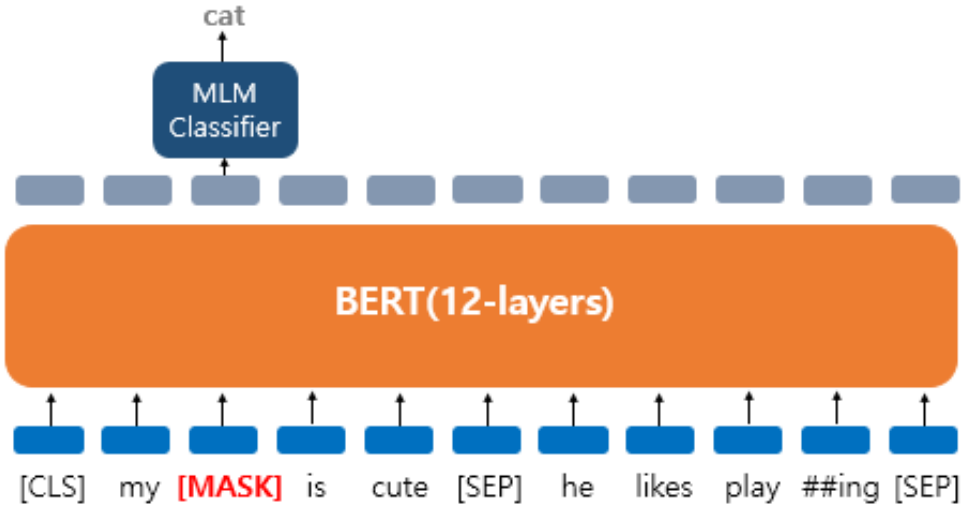
- 10% : 단어를 random word로 교체

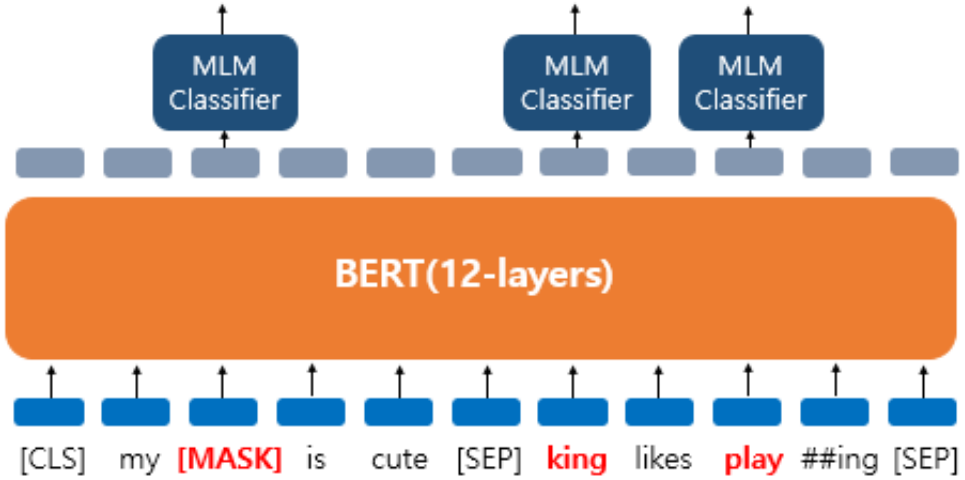
쿠빅은 세상을 꿰뚫어 볼 수 있는 학술적 교류의 장 → 쿠빅은 세상을 뚫어뻥 볼 수 있는 학술적 교류의 장

- 10% : 단어를 교체하지 않음

쿠빅은 세상을 꿰뚫어 볼 수 있는 학술적 교류의 장 → 쿠빅은 세상을 꿰뚫어 볼 수 있는 학술적 교류의 장

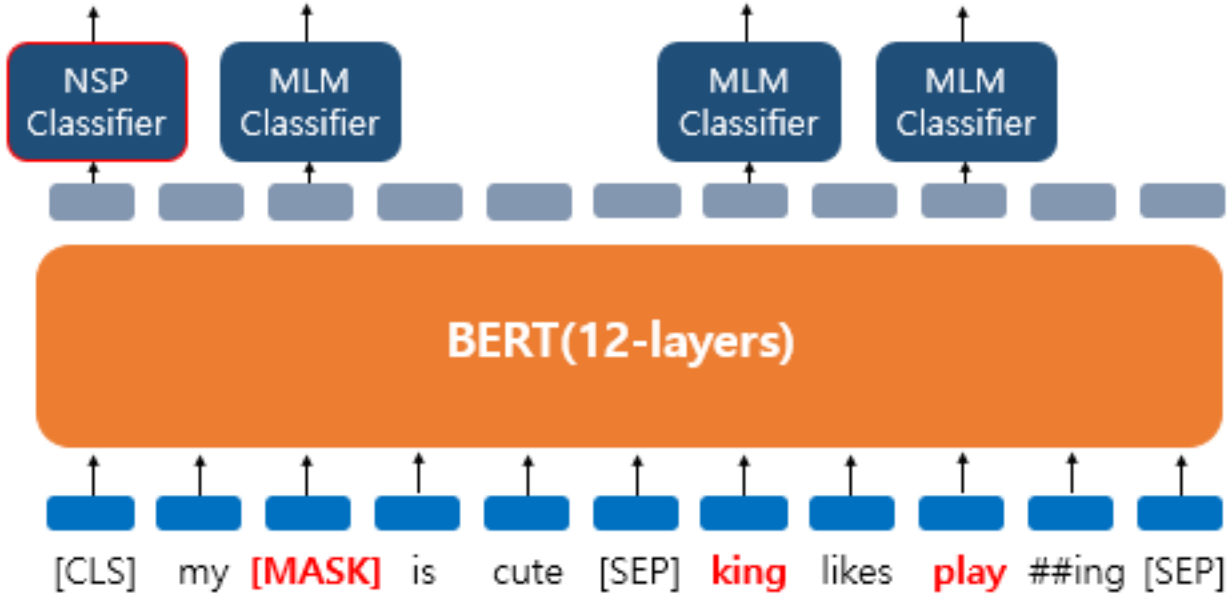
2-5. Pre-training BERT 1



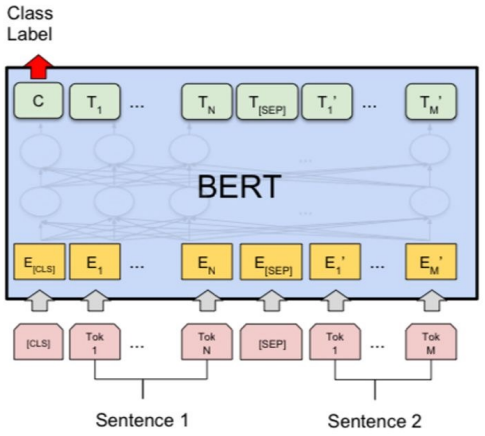


- 이어지는 문장의 경우
  Sentence A : The man went to the store.
  Sentence B : He bought a gallon of milk.
  Label = **IsNextSentence**

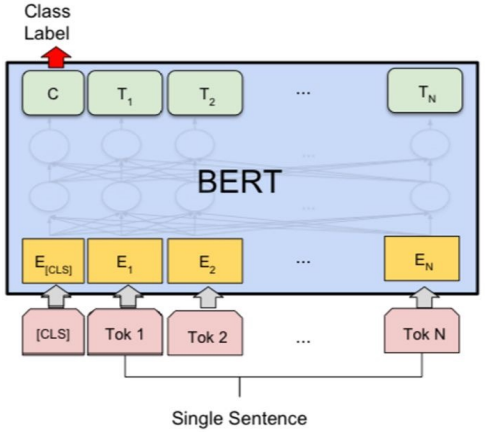
- 이어지는 문장이 아닌 경우
  Sentence A : The man went to the store.
  Sentence B : dogs are so cute.
  Label = **NotNextSentence**



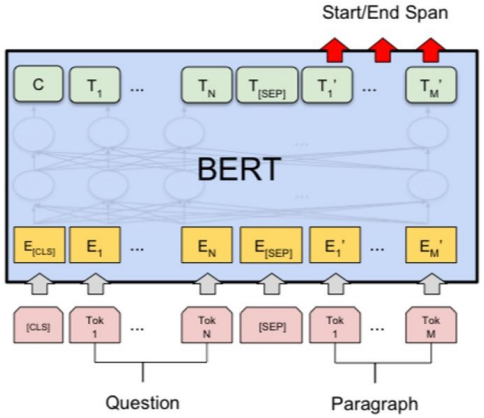
2-7. Fine-tuning BERT



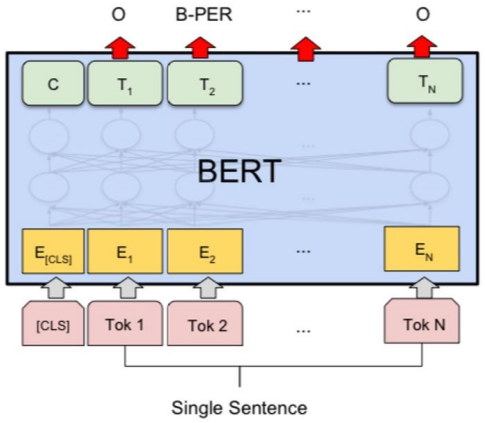
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



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

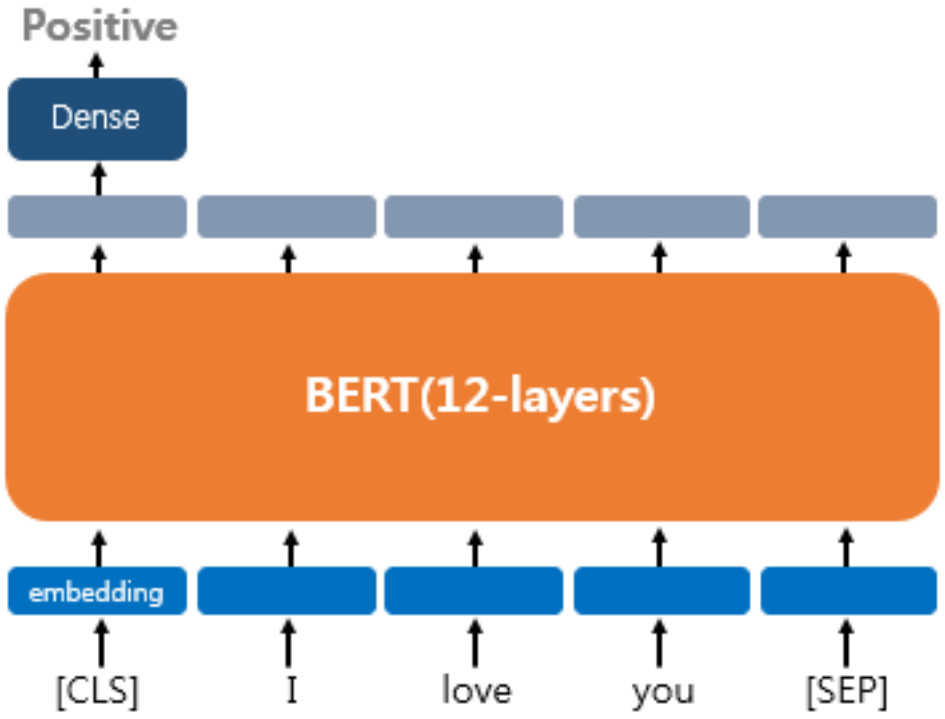


(c) Question Answering Tasks: SQuAD v1.1



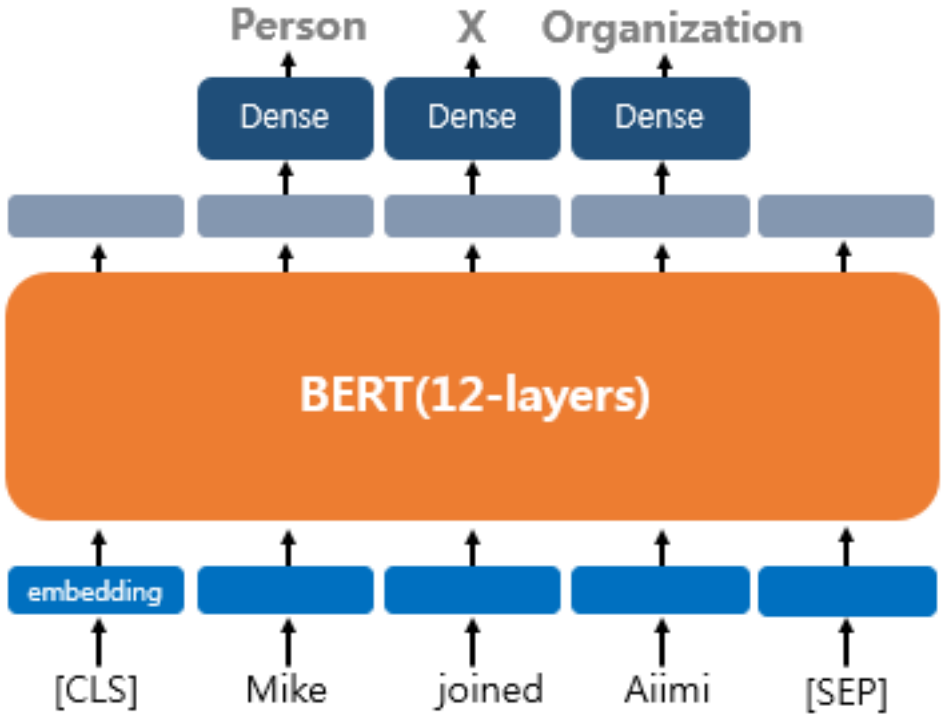
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

- 각 downstream task에 대해 작업별 입력 및 출력을 BERT에 연결하고 레이블이 지정된 데이터를 사용하여 모든 매개변수를 처음부터 끝까지 fine-tuning.
- 각 task는 동일한 사전 훈련된 매개변수로 초기화되더라도 별도의 미세 조정 모델이 존재하게 됨.



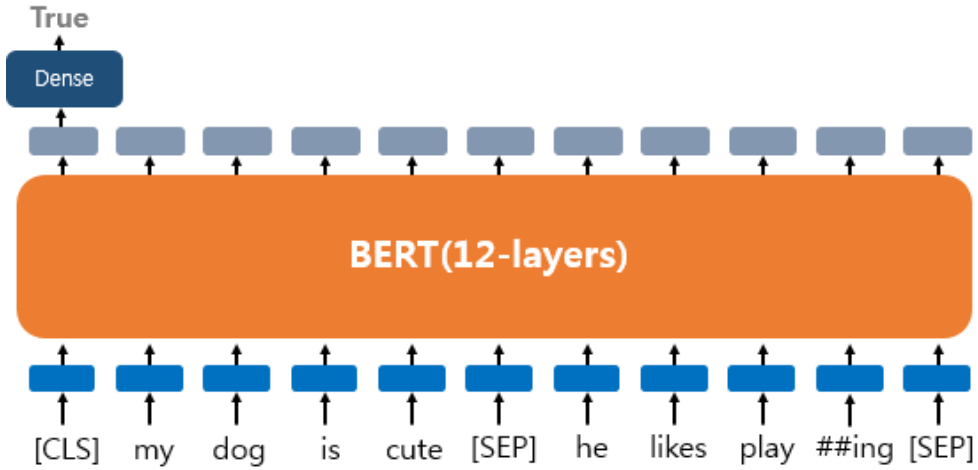
Single Sentence Classification

- 하나의 문서에 대한 텍스트 분류 유형
- 문서의 시작에 [CLS] 라는 토큰을 입력
- 사전학습과 마찬가지로 텍스트 분류 문제를 풀기 위해서 [CLS] 토큰의 위치의 출력층에서 fully-connected layer를 추가하여 분류에 대한 예측을 진행



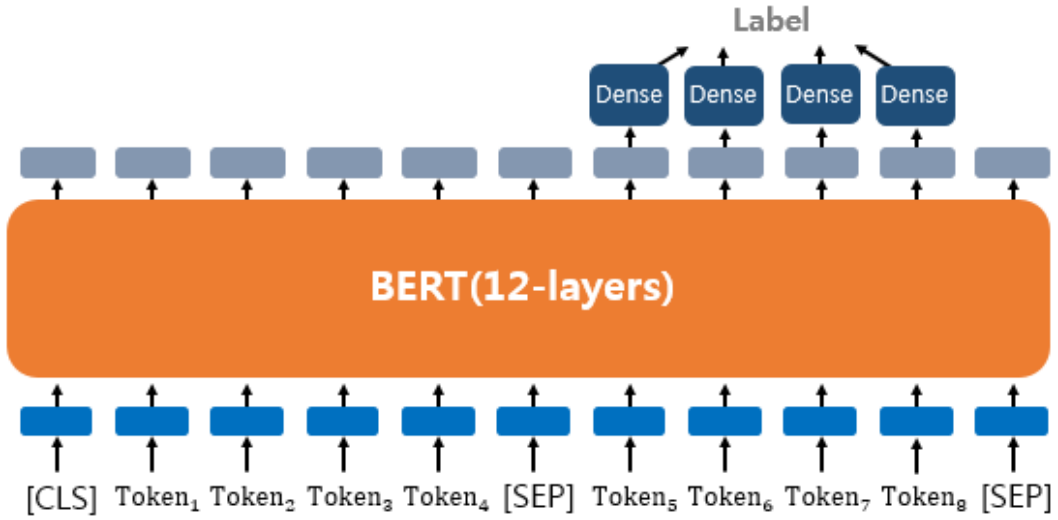
Single Sentence Tagging

- 대표적으로 문장의 각 단어에 품사를 태깅하는 품사 태깅 작업과 개체를 태깅하는 개체명 인식 작업
- 출력층에서는 입력 텍스트의 각 토큰의 위치에 FC layer를 사용하여 분류에 대한 예측을 함



Sentence Pair Classification

- 자연어 추론(Natural language inference) : 두 문장이 주어졌을 때, 하나의 문장이 다른 문장과 논리적으로 어떤 관계에 있는지를 분류
- Sentence 0 임베딩과 Sentence 1 임베딩이라는 두 종류의 세그먼트 임베딩을 사용



Question Answering Task

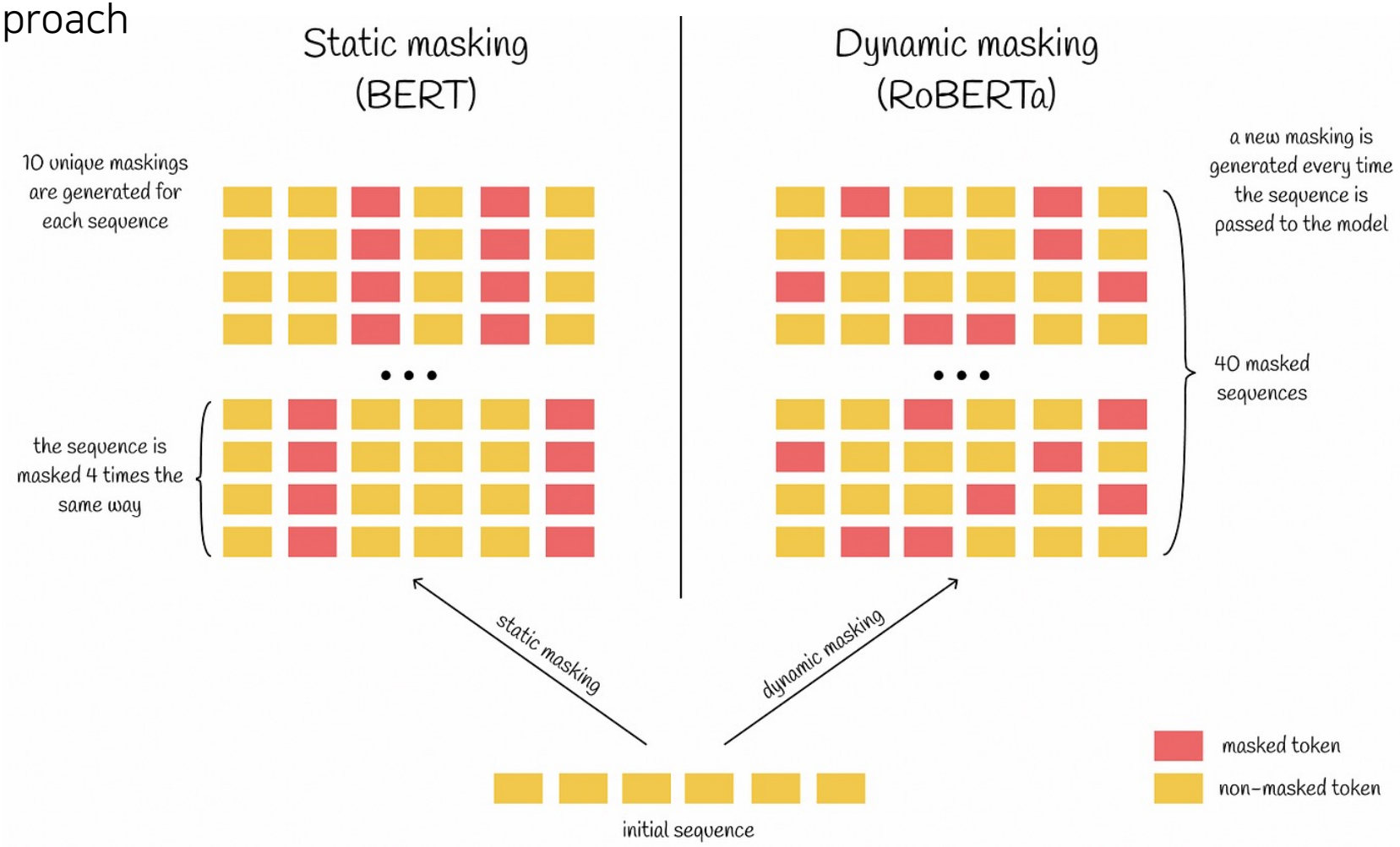
- 질문과 본문이라는 두 개의 텍스트의 쌍을 입력
- SQuAD(Stanford Question Answering Dataset) v1.1 : 질문과 본문을 입력받으면, 본문의 일부분을 추출해서 질문에 답변

Ex)
질문 : "강우가 떨어지도록 영향을 주는 것은?"
본문: "기상학에서 강우는 대기 수증기가 응결되어 중력의 영향을 받고 떨어지는 것을 의미합니다. 강우의 주요 형태는 이슬비, 비, 진눈깨비, 눈, 싸락눈 및 우박이 있습니다."
정답 : "중력"

03 Post-BERT Pre-training Advancements

Robustly optimized BERT approach

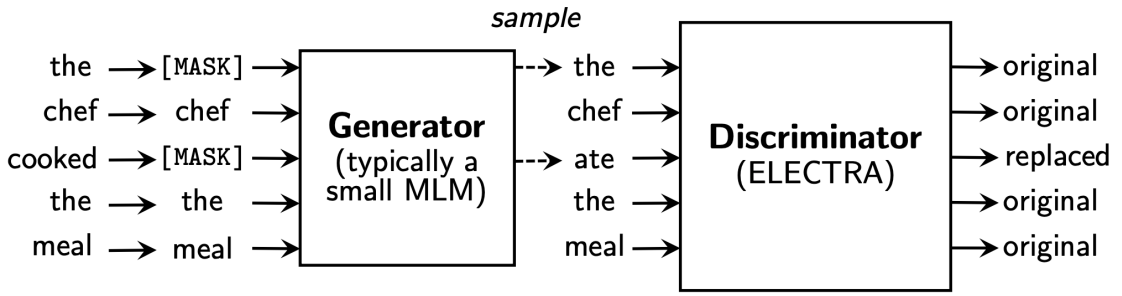
1. 더 큰 배치로 모델을 더 오랫동안 훈련
2. NSP 목적함수 제거
3. 더 긴 시퀀스에 대한 훈련
4. Dynamic masking: 토큰은 각 에포크마다 다르게 마스킹됨
5. 더 큰 데이터 세트 수집



Efficiently Learning an Encoder that Classifies Token Replacements Accurately

Limitation of BERT

1.전체 토큰 중 15%에 대해서만 loss가 발생 (= 하나의 example에 대해서 고작 15%만 학습함)
2. 많은 학습 비용
3.학습 때는 [MASK] 토큰을 모델이 참고하여 예측하지만 실제(inference)로는 [MASK]토큰이 존재하지 않음.



Generator G : BERT의 MLM
Discriminator D : 입력 토큰 시퀀스에 대해서 각 토큰이 original인지 replaced인지 binary classification

$$\mathcal{L}_{\text{MLM}}(\mathbf{x}, \theta_G) = \mathbb{E}\left(\sum_{i \in \mathbf{m}} -\log p_G(x_i | \mathbf{x}^{masked})\right)$$

$$\mathcal{L}_{Disc}(\mathbf{x}, \theta_D)$$

$$= \mathbb{E}\left(\sum_{t=1}^{n} -1(x_t^{corrupt} = x_t)\log D(\mathbf{x}^{corrupt}, t) - 1(x_t^{corrupt} \neq x_t)\log (1 - D(\mathbf{x}^{corrupt}, t))\right)$$

$$\min_{\theta_G, \theta_D} \sum_{\mathbf{x} \in \mathcal{X}} \mathcal{L}_{\text{MLM}}(\mathbf{x}, \theta_G) + \lambda \mathcal{L}_{Disc}(\mathbf{x}, \theta_D)$$

- generator loss와 discriminator loss의 합을 최소화하도록 학습
- 샘플링 과정이 있기 때문에 discriminator loss는 generator로 역전파 되지 않음
- 위의 구조로 pre-training을 마친 뒤에 discriminator만 취해서 downstream task으로 fine-tuning을 진행

GAN과의 차이점

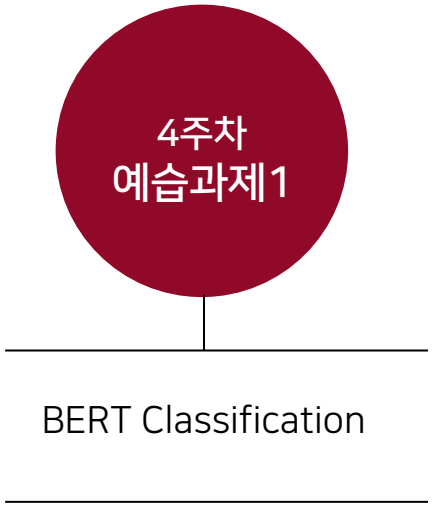
1. Generator가 원래 토큰과 동일한 토큰을 생성했을 때, GAN은 negetive sample (fake)로 간주하지만 ELECTRA는 positive sample로 간주
2. Generator가 discriminator를 속이기 위해 adversarial하게 학습하는 게 아니고 maximum likelihood로 학습
   - Generator에서 샘플링하는 과정 때문에 역전파가 불가능하고, 따라서 adversarial하게 generator를 학습하는 게 어려움
   - 그래서 강화 학습으로 이를 구현해보았지만 maximum likelihood로 학습시키는 것보다 성능이 좋지 않았음
3. Generator의 입력으로 노이즈 벡터를 넣어주지 않음

06 Announcement

Week4 예습과제 Review, week5 예복습 과제 안내, week6진도 안내

6-1. 우수 예습과제 Review

송성님



화면공유 하셔서 3분 내외로 가볍게 리뷰해주시면 됩니다!

6-1. 우수 예습과제 Review

수민님

In [14]:

```
class BertClassifier(nn.Module):

    def __init__(self, dropout=0.5):

        # nn.Modul 클래스의 초기화 메서드 실행
        super(BertClassifier, self).__init__()

        # 사전훈련된 Bert 모델 불러옴
        self.bert = BertModel.from_pretrained('bert-base-cased')
        # nn.Dropout 사용, 드롭아웃 레이어 생성
        self.dropout = nn.Dropout(dropout)
        # 768 차원의 BERT 출력을 5차원으로 선형변환하는 레이어 생성
        self.linear = nn.Linear(768, 5)
        # ReLU 활성화 함수 생성
        self.relu = nn.ReLU()

    def forward(self, input_id, mask): # input_id는 BERT의 입력토큰 ID, mask는 어텐션 마스크

        # BERT 모델이 입력 전달, pooled_output 반환 ( 불필요한 출력은 _로 무시 )
        _, pooled_output = self.bert(input_ids= input_id, attention_mask=mask,return_dict=
        # dropout 레이어를 통과시켜 dropout 적용한 출력을 얻음
        dropout_output = self.dropout(pooled_output)
        # 출력을 감정 클래스 수에 해당하는 차원으로 변환
        linear_output = self.linear(dropout_output)
        # ReLU 활성화 함수를 적용하여 최종출력을 얻음
        final_layer = self.relu(linear_output)

        return final_layer
```

6-1. 우수 예습과제 Review

수민님

```
In [9]: # bert-large-cased 토크나이저 로드(추가)
        tokenizer2 = BertTokenizer.from_pretrained('bert-large-cased')

tokenizer_config.json:   0%|          | 0.00/29.0 [00:00<?, ?B/s]
vocab.txt:   0%|          | 0.00/213k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/436k [00:00<?, ?B/s]
config.json:   0%|          | 0.00/762 [00:00<?, ?B/s]

In [10]: # bert-base-uncased 로드(추가)
         tokenizer3 = BertTokenizer.from_pretrained('bert-base-uncased')

tokenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]
vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]
```

6-1. 우수 예습과제 Review

유민님

In [42]:

```
def train(model, train_data, val_data, learning_rate, epochs):

    train, val = Dataset(train_data), Dataset(val_data)

    train_dataloader = torch.utils.data.DataLoader(train, batch_size=2, shuffle=True) # batch size 2
    val_dataloader = torch.utils.data.DataLoader(val, batch_size=2)

    use_cuda = torch.cuda.is_available()
    device = torch.device("cuda" if use_cuda else "cpu")

    criterion = nn.CrossEntropyLoss() # CrossEntropyLoss 사용
    optimizer = Adam(model.parameters(), lr= learning_rate) # Adam optimizer 사용

    if use_cuda:
            # model, loss func를 gpu로
            model = model.cuda()
            criterion = criterion.cuda()

    for epoch_num in range(epochs):
            # train accuracy, loss를 기록하기 위해 init.
            total_acc_train = 0
            total_loss_train = 0

            # for문 작업 현황을 보기 위해tqdm사용
            for train_input, train_label in tqdm(train_dataloader):

                train_label = train_label.to(device)
                mask = train_input['attention_mask'].to(device)
                input_id = train_input['input_ids'].squeeze(1).to(device) # input_ids에 있는 불필요한 차원 제거 후 dev

                output = model(input_id, mask)

                batch_loss = criterion(output, train_label.long()) #train_label에 long을 해줘야 계산 가능.
                total_loss_train += batch_loss.item()

                acc = (output.argmax(dim=1) == train_label).sum().item() # output중 값이 가장 큰 값을 label과 비교 ->
                total_acc_train += acc
```

6-2. Week5 예,복습과제 안내, Week6 진도 안내

코드과제의 파일형식은 ipynb로, KUBIG 24-1 **Github** repo에 업로드 될 예정입니다!
Colab 환경에서 제작된 과제들이므로 google colab에서 실행하시는 것을 권장드립니다.

WEEK5 복습과제1

BERT for sequence classification

WEEK5 예습과제1

koGPT-2 문장 생성

WEEK6 진도

- GPT

WEEK6 진도 해당 범위(읽어오시길 권장 드립니다!)

- https://medium.com/walmartglobaltech/the-journey-of-open-ai-gpt-models-32d95b7b7fb2

E.O.D
수고하셨습니다!