시작하기 전에

- 이 강의는 애저톤 프로젝트를 위한 배경지식을 전달드리는 시간입니다.
- 머신러닝과 NLP를 이해할 수 있는 중급 이상 수준의 내용으로 구성되어 있습니다.
- 강의 중에 잘 이해가 되지 않거나 궁금한 내용이 있으시면 강의 후에 Q&A를 통해 문의해주세요.

목차

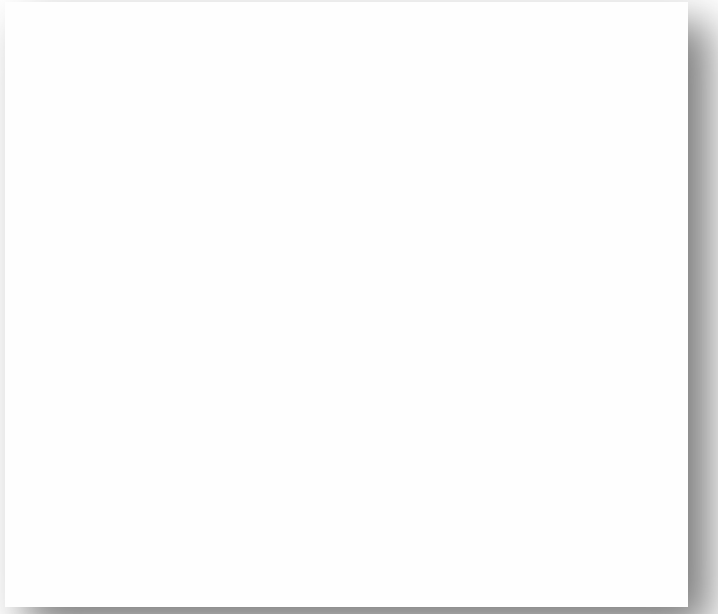
1. Transformer 소개
2. Bert와 GPT에 대해서
3. GPT-3를 이용한 Few-shot learning 실습

실습 링크

- Colab link: https://bit.ly/3JQxjmg
- Github link: https://bit.ly/3yQyxHY

Transformer 소개

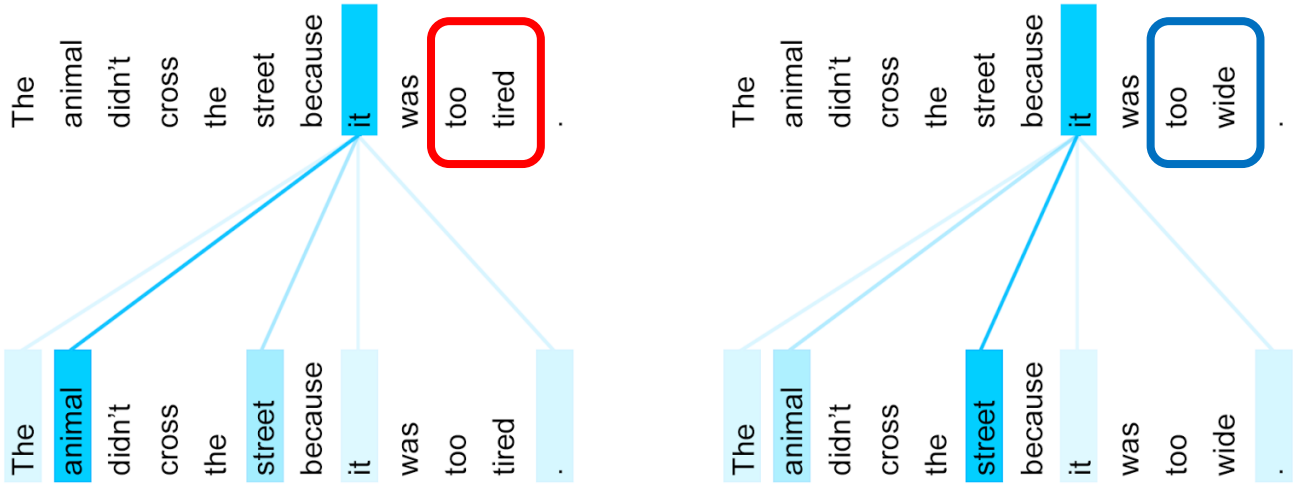
- Transformer 모델이란?
  - 번역 모델 연구를 통해 나온 모델 (Seq2Seq 기반 모델)
  - 문장 속 단어와 같은 순차 데이터 내의 관계를 학습을 통해 파악해 순차 정보의 맥락과 의미를 학습하는 신경망 모델



Tramsformer 모델 동작 방식

Transformer 소개

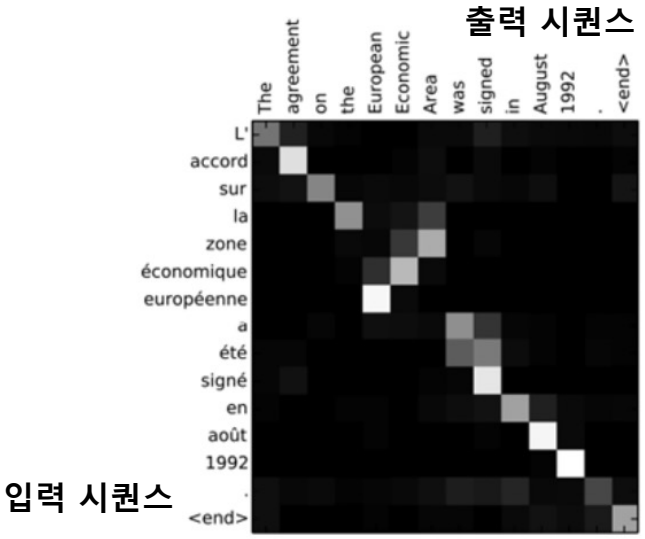
- Transformer 모델이란?
  - 어텐션(Attention) 또는 셀프 어텐션(Self-Attention)이라 불리는 방식을 활용하여 서로 떨어져 있는 데이터 요소들의 의미가 관계에 따라 미묘하게 달라지는 부분을 감지함



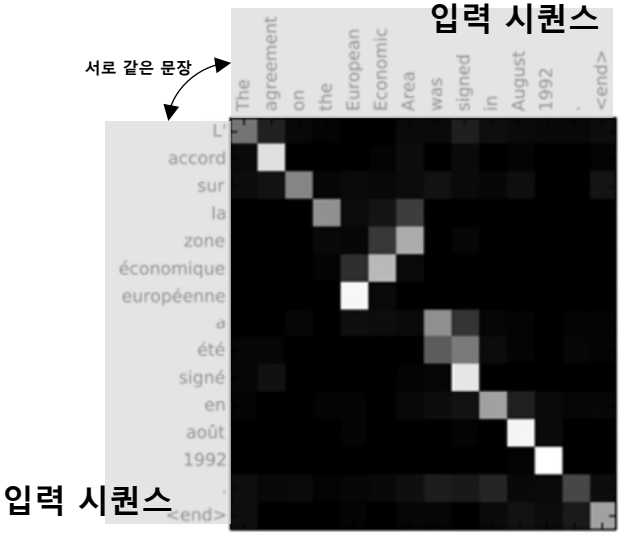
문맥에 따라 한 단어(it)과 가까운 관계 어휘가 달라짐을 확인할 수 있음

Transformer 소개

- Self-Attention 컨셉
  - Seq2Seq 번역 모델에서 Attention의 경우 생성(번역)할 단어와 어떤 입력 시퀀스 단어와의 관계가 얼마나 가까운지를 점수로 표현
  - Self-Attention은 입력 시퀀스 단어들 기준에서 다른 시퀀스 단어들 간의 관계가 얼마나 가까운지를 점수로 표현



Seq2Seq Attention 컨셉



Self-Attention 컨셉

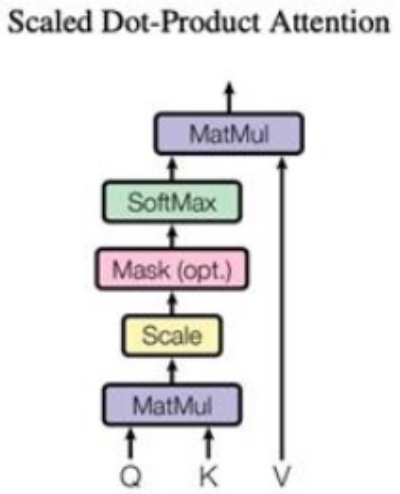
Transformer 소개

- Self-Attention 연산
  - 입력 시퀀스를 Key, Query, Value로 구성하여 시퀀스 내 단어 간의 Attention Score를 구성
  - Key, Query를 이용하여 Attention Score 행렬을 생성하고, Value에 행렬곱 연산을 하는 과정

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

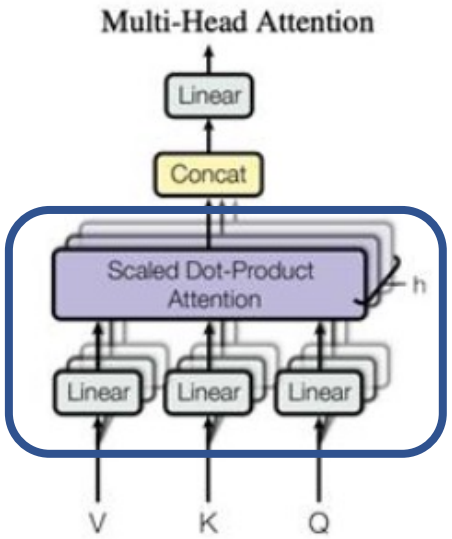
Attention Score 행렬

Self-Attention 연산과정을 거친 단어 임베딩



Transformer 소개

- Multi-head Attention 연산
  - Self-Attention 연산을 각각 N번 하는 과정
  - N번 하는 것을 multi-head 수로 설정
  - 시퀀스 단어 간의 관계를 Attention Score 하나로 보지 않고 여러개로 보고자 하는 의도



Attention 연산 과정

Transformer 소개

- Transformer 모델 구조
  - Encoder-Decoder 구조의 Seq2Seq 모델
    - Decoder는 다음 단어를 생성해야 하기 때문에 생성될 단어들에 대한 Mask를 적용
    - Encoder는 입력 시퀀스 단어의 관계를 봐야하기 때문에 Mask를 적용하지 않음
    - Position Embedding을 이용하여 시퀀스의 순서를 유지할 수 있도록 함
  - Transformer 모델은 다음의 모델로 파생
    - Encoder는 BERT 모델로 표현
    - Decoder는 GPT 모델로 표현

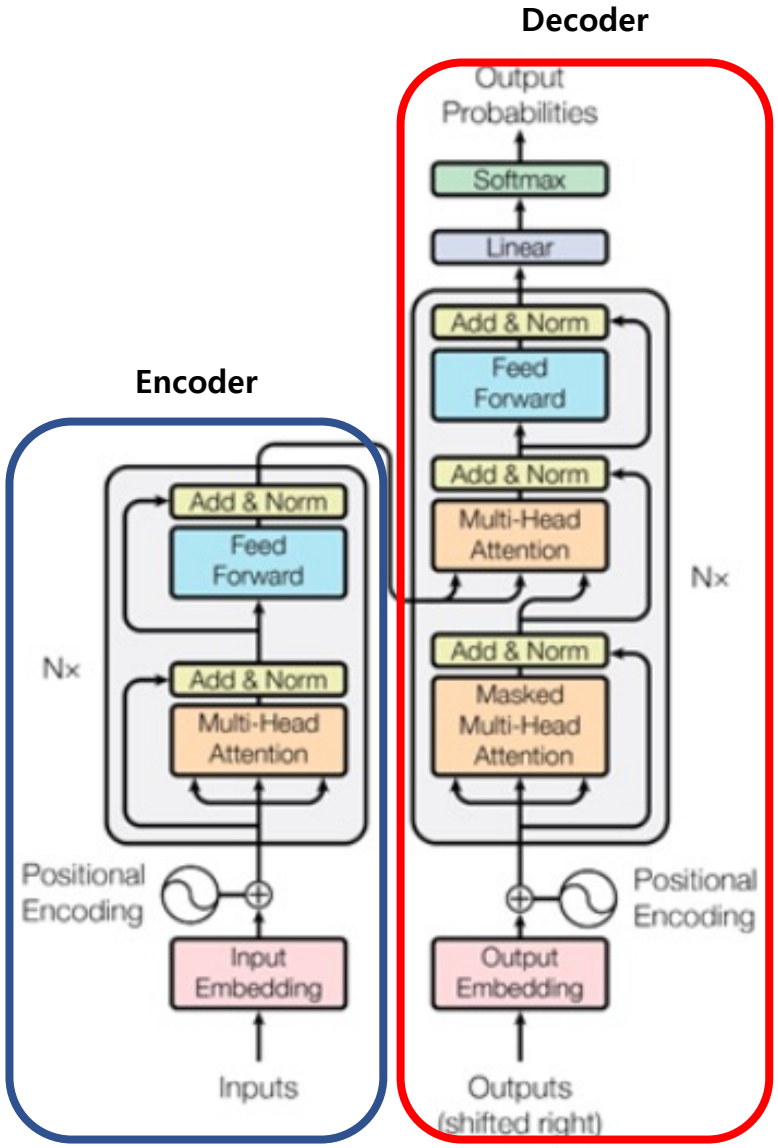


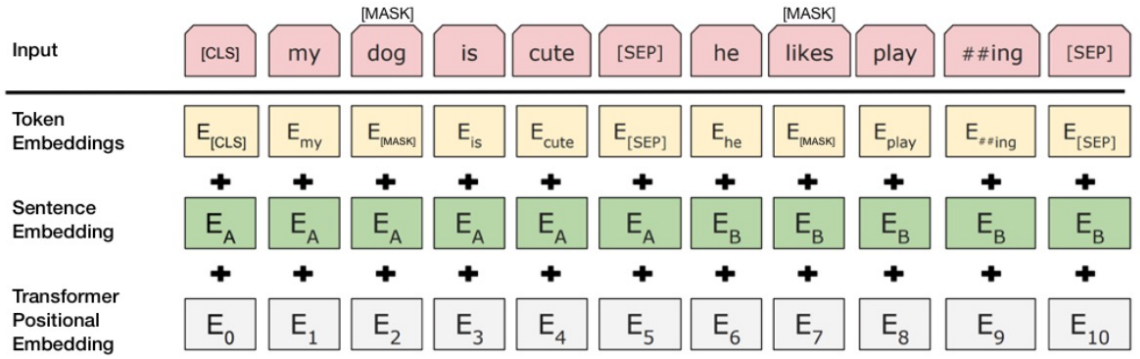
Figure 1: The Transformer - model architecture.

Bert와 GPT에 대해서

- Transformer 의의
  - 문장 내 단어들의 관계를 직관적으로 이해할 수 있는 방법
  - 문장 내 단어들에 대한 관계 정보를 Attention Map으로 연산과정에서 표현 (RNN은 hidden state 정보로만 대략 관계 정보를 알 수 있음)

Bert와 GPT에 대해서

- Bert 모델에 대해서
  - 목표: 문맥을 고려한 언어 표현 벡터를 생성하도록 학습
    - Transformers Encoder 모델 활용한 Pre-trained 모델
    - MLM (Masked Language Model)과 NSP (Next Sentence Prediction) 으로 모델 학습



MLM 학습을 위한 임베딩 입력 구성

Bert와 GPT에 대해서

- Bert 모델에 대해서
  - Pre-trained 모델 Bert를 각 자연어처리 Task에 fine-tuning으로 학습
    - RNN모델에서 학습했던 자연어처리 Task 학습 방식 그대로 Bert 모델에 학습
  - Pre-training 과정에서 학습된 언어정보가 성능향상에 도움을 줌

BERT: Pre-training then connect with a downstream MLP for fine tuning

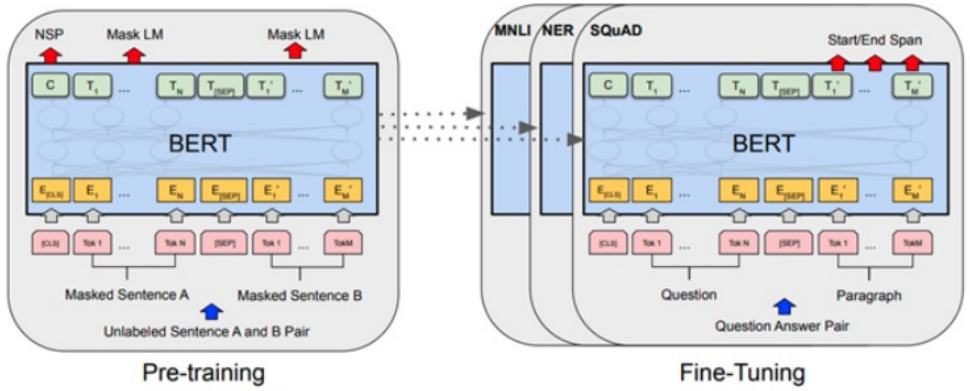
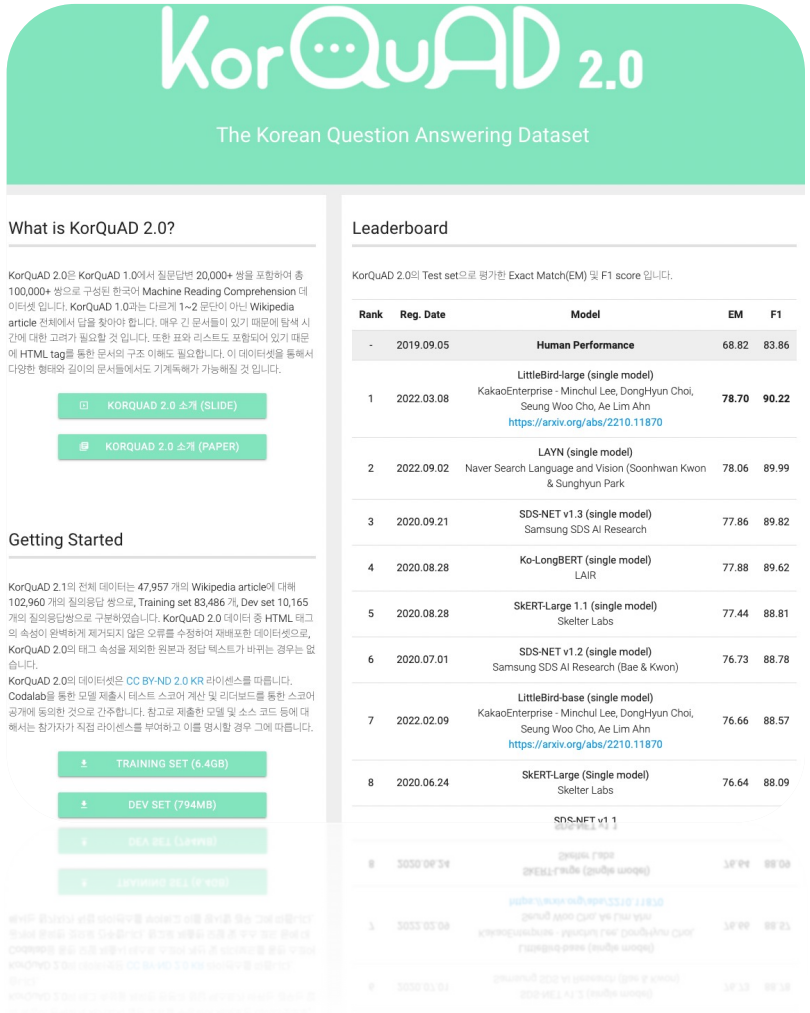


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

Bert와 GPT에 대해서

- Bert 모델 활용 예시
  - KorQuad (Reading Comprehension Task)
    - 문서와 질문이 주어졌을 때 정답 단어를 문서 텍스트 안에서 답을 찾는 Task
    - 대회에 참가한 대부분의 모델들이 Bert 기반 모델을 활용
    - 현재는 Human Performance 넘어선 상황
    - 지식 또는 컨텐츠 검색 서비스에서 활용하는 기술

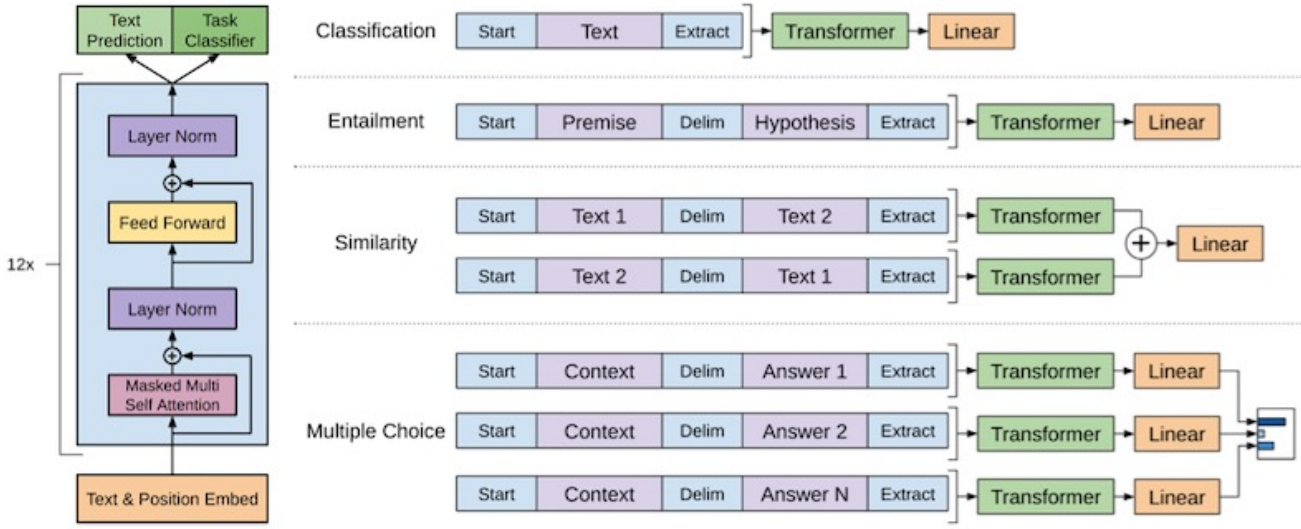


Bert와 GPT에 대해서

- GPT 모델에 대해서
  - 목표: 좋은(?) 퀄리티의 텍스트를 생성하도록 학습
    - Transformers Decoder 모델 활용한 Pre-trained 모델
    - Next Token Prediction 방식으로 Language Generation 학습을 함
  - 현재 GPT3의 경우 모델 파라메터 크기가 Bert 보다 더 큰 모델임
    - Bert-Base: 110M
    - Bert-Large: 340M
    - GPT-2: 1.5B
    - GPT-3: 175B

Bert와 GPT에 대해서

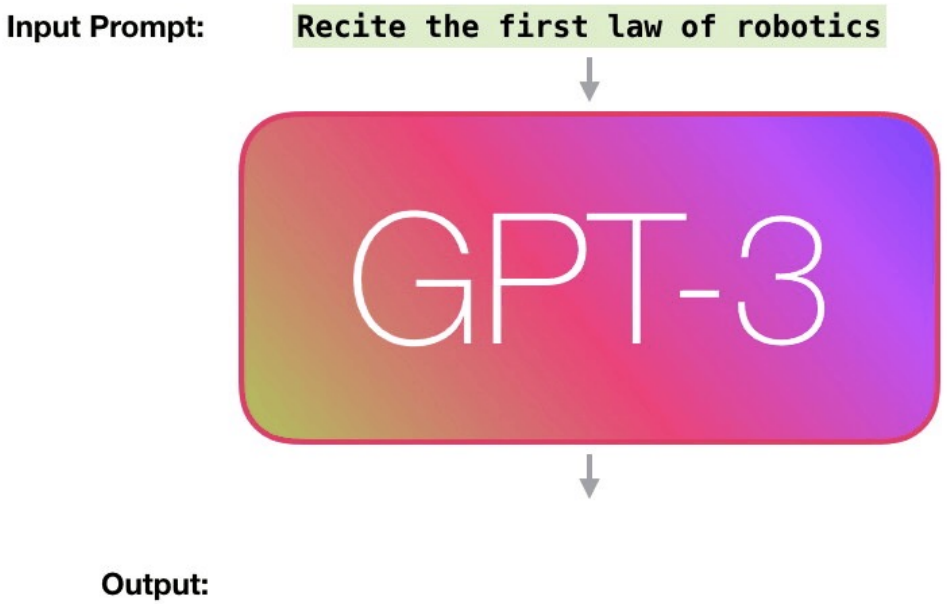
- GPT-1 모델에 대해서
  - 초기 GPT-1은 Bert와 같이 Pre-training이후 finetune 방식으로 소개



Bert와 같이 Pre-training을 하고 자연어 Task에 맞게 finetune을 함

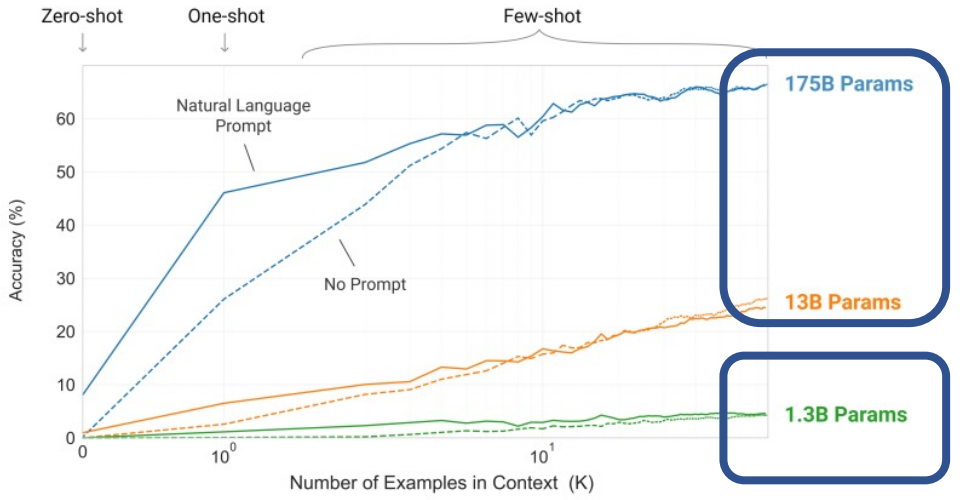
Bert와 GPT에 대해서

- GPT-2,3 모델에 대해서



Bert와 GPT에 대해서

- GPT-2,3 모델에 대해서
  - GPT-2
    - **"Language Models are Unsupervised Multi-task Learners"** 논문으로 소개
    - 언어모델만으로 자연어처리 Task를 해결할 수 있음을 보이고자 함
  - GPT-3
    - **"Language Models are Few-shot Learners"** 라는 논문으로 소개
    - GPT-2 모델에서 파라메터 크기를 100배 이상 늘려 finetuning 없이 few-shot learning으로 자연어처리 Task를 해결할 수 있음을 보임



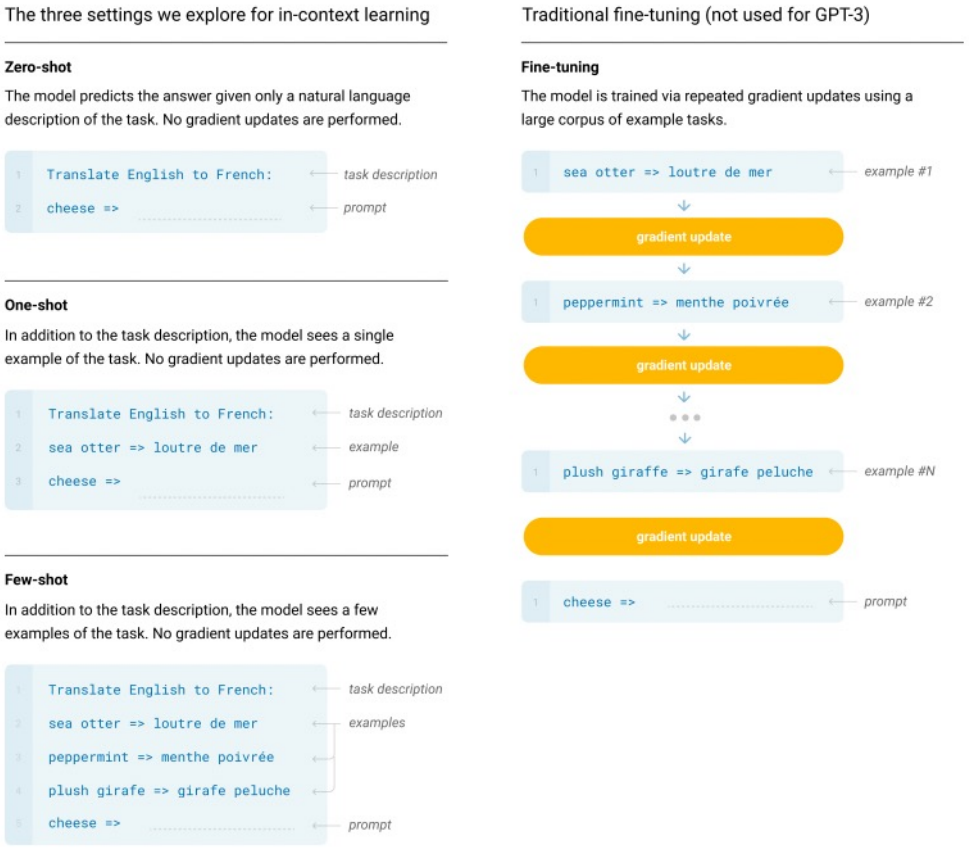
모델 크기와 Few-shot 수에 따른 자연어처리 Task 성능 변화

Bert와 GPT에 대해서

- Few-shot Learning에 대해서
  - Zero-shot (예시 없이 모델에 질문을 하는 경우)
    - 다음에 대해 답을 해주세요, 3+1=?
  - One-shot (1개의 예시를 제시하고 질문을 하는 경우)
    - 1+6=7, 다음에 대해 답을 해주세요, 3+1=?
  - Few-shot (K=3)
    - 1+2=3, 3+6=9, 13+2=15, 다음에 대해 답을 해주세요, 3+1=?

Bert와 GPT에 대해서

- Few-shot Learning vs Fine-tuning
  - Fine-tuning
    - 데이터셋을 모델에 Gradient Update를 통해 학습하는 방식
    - 많은 학습 데이터를 (적어도 1~10k건) 필요
  - Few-shot Learning
    - GPT-3와 같은 거대 언어모델을 이용하여 In-Context Learning 을 이용하는 방식
    - In-Context Learning 예시를 통해 모델이 추론 단계에서 Task를 유추하는 현상
    - 적은 데이터 샘플만으로 가능



Bert와 GPT에 대해서

- Few-shot Learning vs Fine-tuning
  - Fine-tuning
    - 용도가 명확하고 정확성을 요구하는 Task에서 활용
    - 모델의 목적이 명확해야 효과가 있음 (e.g. 스팸 필터링, 개체명 탐지)
  - Few-shot Learning
    - 용도에 대한 정의가 정해지지 않고 Prototype을 하는 상황에서 활용
    - 모델을 통해 얻을 수 있는 용도를 빠르게 탐색하는데 효과가 있음

Bert와 GPT에 대해서

- GPT가 왜 사람들에게 쉽게 다가온걸까?
  - 생성 모델과 Few-shot learning
    - 입력 내용을 토대로 다음 단어가 생성이 되는 것을 목표
    - Fine-tuning없이도 few-shot 성능이 좋은 결과를 보임
    - 모델 스케일을 키우면 성능이 좋아진다는 경향이 있음
    - 명령어 만으로 사용자가 원하는 것을 쉽게 볼 수 있는 점

Bert와 GPT에 대해서

- GPT가 왜 사람들에게 쉽게 다가온걸까?
  - 왜 Bert는 GPT 처럼 안될까요?
    - Bert는 텍스트 생성 아닌 입력 문장에 대한 언어정보를 표현하는 것이 목표
    - 텍스트가 가진 문맥 정보를 표현하는 것이 중요하고, 이를 Finetuning으로 특정 Task에 최적화 하는 접근
    - GPT는 이와 상관없이 학습없이 텍스트 인터페이스로 사용자 명령으로 Task 수행을 하는 목적을 지향

Bert와 GPT에 대해서

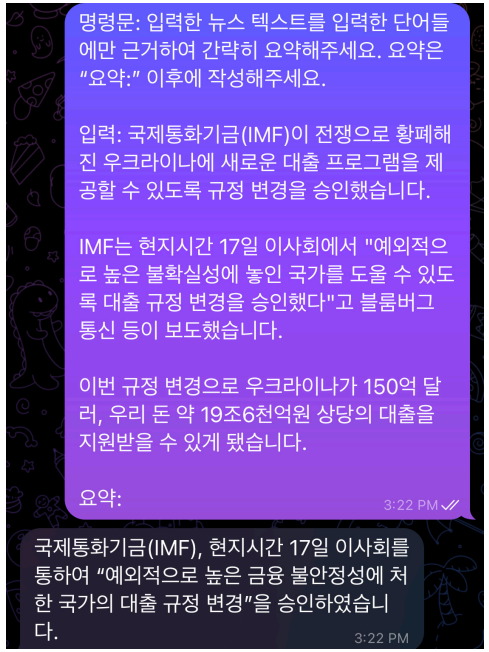
- GPT가 왜 사람들에게 쉽게 다가온걸까?
  - 사용자 입장에서 접근하기 쉬운 인터페이스
    - 사용자가 입력하면 모델 출력만으로 바로 확인할 수 있는 구조
    - 목적이 정해지지 않아도 질의를 해볼 수 있는 구조
    - 질문만 잘하면 원하는 답을 얻을 수 있다는 사용자 경험들이 축적될 수 있음

지금까지 이야기한 것들

1. Trasformer
   - Self-Attention
   - Encoder-Decoder
2. Bert
   - Pre-training
   - Finetuning
3. GPT
   - Text Generation
   - Few-shot Learning

GPT-3를 이용한 Few-shot learning 실습

- 한국어 오픈소스 GPT-3을 활용한 Few-shot Learning 실습
  - KoAlpaca 모델을 가지고 NSMC task를 실습
    - KoAlpaca 모델은 polyglot-ko 5.8b 모델(GPT-3)을 Instruction 오픈 데이터를 이용하여 finetuning한 모델
    - ChatGPT 만큼은 아니지만 사용자 명령에 적절한 응답을 줄 수 있는 모델
    - KoAlpaca project
      - Github: https://github.com/Beomi/KoAlpaca
      - Demo: https://t.me/KoAlpacaBot



KoAlpaca 데모 예시

GPT-3를 이용한 Few-shot learning 실습

- Polyglot 프로젝트
  - 오픈소스 그룹 EleutherAI에서 진행하는 프로젝트
    - EleutherAI는 GPT와 같은 모델에 대한 연구개발을 비영리로 진행하는 집단
  - polyglot-ko를 시작으로 다양한 언어에 대한 모델을 제공할 예정
  - Polyglot project
    - Github: https://github.com/EleutherAI/polyglot
    - Demo: https://master-polyglot-deploy-jason9693.endpoint.ainize.ai

환경 구성

- 라이브러리 설치
  - 실습에서 활용할 라이브러리 설치
    - Transformers: gpt-3 모델을 다운받고 실행할 수 있도록 하는 라이브러리

```
!pip install transformers
!pip install accelerate
```

실습 준비

- 학습 데이터 준비
  - NSMC 데이터를 다운로드 (Github에 저장된 데이터)

```
!mkdir -p data_in/KOR/naver_movie
!wget https://raw.githubusercontent.com/NLP-kr/tensorflow-ml-nlp-tf2/master/7.PRETRAIN_METHOD/data_in/KOR/naver_movie/ratings_train.txt \
                -O data_in/KOR/naver_movie/ratings_train.txt
!wget https://raw.githubusercontent.com/NLP-kr/tensorflow-ml-nlp-tf2/master/7.PRETRAIN_METHOD/data_in/KOR/naver_movie/ratings_test.txt \
                -O data_in/KOR/naver_movie/ratings_test.txt
```

실습 준비

- 주요 호출 라이브러리
  - transformers: 모델 및 토크나이저 호출
  - pandas: 데이터 불러오기
  - re: 데이터 전처리

```
import pandas as pd
import re

from transformers import AutoModelForCausalLM, AutoTokenizer
```

실습 준비

- 모델을 사용하기 위한 위한 클래스
  - transformers.AutoModelForCausalLM
    - GPT-3 모델을 생성하기 위한 클래스
    - AutoModelFOrCausalLM을 이용하여 텍스트 생성을 함
  - transformers.AutoTokenizer
    - GPT-3 모델에 입력할 텍스트를 토큰화 하기 위한 클래스
    - 텍스트를 AutoTokenizer를 통해 인덱스 데이터로 변환

실습 준비

- 랜덤 초기값 설정
  - Few-shot learning에서 활용할 예제들을 샘플링으로 추출
  - 샘플링을 재현하기 위해 랜덤 초기값을 지정

```
SEED_NUM = 1234
random.seed(SEED_NUM)
```

모델 불러오기

- 모델과 토크나이저 불러오기
  - Few-shot learning에서 활용할 예제들을 샘플링으로 추출
  - 샘플링을 재현하기 위해 랜덤 초기값을 지정

```
tokenizer = AutoTokenizer.from_pretrained("EleutherAI/polyglot-ko-5.8b")
cls_model = AutoModelForCausalLM.from_pretrained("beomi/KoAlpaca-Polyglot",
                                                 torch_dtype=torch.float16,
                                                 device_map='sequential’,
                                                 low_cpu_mem_usage=True).cuda()
```

데이터 불러오기

- 학습 데이터 불러오기
  - NSMC 텍스트 파일을 pandas 라이브러리로 불러오기

```
DATA_IN_PATH = './data_in/KOR'
DATA_OUT_PATH = './data_out/KOR'

DATA_TRAIN_PATH = os.path.join(DATA_IN_PATH, 'naver_movie', 'ratings_train.txt')
DATA_TEST_PATH = os.path.join(DATA_IN_PATH, 'naver_movie', 'ratings_test.txt')

train_data = pd.read_csv(DATA_TRAIN_PATH, header = 0, delimiter = '\t', quoting = 3)
train_data = train_data.dropna()
```

Few-shot Learning을 위한 준비

- Few-shot learning을 위해선
  - GPT-3 모델에 입력할 Prompt (명령) 텍스트 구성이 필요
  - Few-shot prompt을 위한 템플릿 생성을 자동화

```
print('데이터 positive 라벨: ', '긍정')
print('데이터 negative 라벨: ', '부정’)
```

OUTPUT: 데이터 positive 라벨: 긍정
데이터 negative 라벨: 부정

```
# 학습 예시 케이스 구조
Print('문장: 오늘 기분이 좋아\n감정: 긍정\n’)
```

OUTPUT: 문장: 오늘 기분이 좋아
감정: 긍정

Few-shot Learning을 위한 준비

- 텍스트 토큰 길이에 대한 데이터 분석
  - GPT-3 모델에 입력할 수 있는 최대 토큰 길이
  - 입력할 Prompt 텍스트의 최대 토큰 길이

```
print('gpt3 최대 토큰 길이: ', cls_model.config.max_position_embeddings)
```

OUTPUT: gpt2 최대 토큰 길이: 2048

Few-shot Learning을 위한 준비

- 텍스트 토큰 길이에 대한 데이터 분석
  - NSMC 문장 토큰 길이 분포를 통해 긴 텍스트를 제거할 수 있도록 함

```
sent_lens = [len(tokenizer(s).input_ids) for s in tqdm(train_data['document'])]

print('Few shot 케이스 토큰 평균 길이: ', np.mean(sent_lens))
print('Few shot 케이스 토큰 최대 길이: ', np.max(sent_lens))
print('Few shot 케이스 토큰 길이 표준편차: ',np.std(sent_lens))
print('Few shot 케이스 토큰 길이 80 퍼센타일: ',np.percentile(sent_lens, 80))
```

OUTPUT:

Few shot 케이스 토큰 평균 길이: 20.22
Few shot 케이스 토큰 최대 길이: 280
Few shot 케이스 토큰 길이 표준편차: 16.488
Few shot 케이스 토큰 길이 80 퍼센타일: 27.0

Few-shot Learning을 위한 준비

- Few-shot 예제 데이터 구성
  - 토큰길이 25 이하인 문장들에 대해서만 few-shot 예제로 구성
  - GPT-3 모델에 최대 70여개의 예제를 입력할 수 있음

```
train_fewshot_data = []

for train_sent, train_label in tqdm(train_data[['document', 'label']].values):
    tokens = tokenizer(train_sent).input_ids

    if len(tokens) <= 25:
        train_fewshot_data.append((train_sent, train_label))
```

Few-shot Learning 및 평가

- 평가 데이터 불러오기
  - 학습 데이터와 마찬가지로 pandas 라이브러리를 이용하여 불러오기

```
test_data = pd.read_csv(DATA_TEST_PATH, header=0, delimiter='\t', quoting=3)
test_data = test_data.dropna()
test_data.head()
```

	id	document	label
0	6270596	굳 ㅋ	1
1	9274899	GDNTOPCLASSINTHECLUB	0
2	8544678	뭐야 이 평점들은.... 나쁘진 않지만 10점 짜리는 더더욱 아니잖아	0
3	6825595	지루하지는 않은데 완전 막장임... 돈주고 보기에는....	0
4	6723715	3D만 아니었어도 별 다섯 개 줬을텐데.. 왜 3D로 나와서 제 심기를 불편하게 하죠??	0

Few-shot Learning 및 평가

- 평가 데이터와 Few-shot 예제 데이터 구성
  - 평가할 데이터 중 일부만을 가지고 Few-shot 결과를 확인
  - Few-shot 예제의 경우 미리 10개씩 예제를 묶어둠

```
# 평가 데이터 수
sample_size = 500

# 평가에서 활용할 few-shot 예제를 묶음으로 저장
train_fewshot_samples = []
for _ in range(sample_size):
fewshot_examples = sample(train_fewshot_data, 10)
    train_fewshot_samples.append(fewshot_examples)

if sample_size < len(test_data['id']):
    test_data = test_data.sample(sample_size, random_state=SEED_NUM)
```

Few-shot Learning 및 평가

- Few-shot 템플릿 구성을 위한 함수
  - 댓글 감정 분석을 위한 prompt 구성
  - 댓글 텍스트 전처리 작업 (한글 char로만 구성)

```
# Prompt 구성
def build_prompt_text(sent):
    return "문장: " + sent + '\n감정:’

# 텍스트 전처리
def clean_text(sent):
    sent_clean = re.sub("[^가-힣ㄱ-ㅎㅏ-ㅣ\\s]", "", sent)
    return sent_clean
```

Few-shot Learning 및 평가

- Few-shot 템플릿 구성을 위한 함수
  - Few-shot 예시 템플릿 구성

```
# Prompt 구성
def generate_fewshot_example(data):
    example_text, example_label = data
    # 텍스트 전처리
    cleaned_example_text = clean_text(example_text)
    # Prompt 형식 구성
    fewshot_example_text = build_prompt_text(cleaned_example_text)
    # Label 추가
    fewshot_example_text += ' 긍정' if example_label == 1 else ' 부정' + '\n'

    return fewshot_example_text
```

Few-shot Learning 및 평가

- GPT-3를 활용한 감정 분석
  - Prompt 텍스트를 GPT-3 모델에 입력하여 감정 결과 출력

```
def predict_by_generation(prompt_text):
    # 토큰화 및 인덱싱
    tokens = tokenizer(prompt_text, return_tensors="pt")
    token_ids, attn_mask = tokens.input_ids.cuda(), tokens.attention_mask.cuda()
    # 텍스트 생성
    gen_tokens = cls_model.generate(input_ids=token_ids, attention_mask=attn_mask,
                                    max_new_tokens=1, pad_token_id=0)
    # 인덱스 복호화
    pred = tokenizer.batch_decode(gen_tokens[:, -1])[0].strip()

    return pred
```

Few-shot Learning 및 평가

- Few-shot Learning 실행

```
pred_tokens = []
real_labels = []

total_len = len(test_data[['document','label']].values)

for i, test_data in tqdm(enumerate(test_data[['document','label']].values), total=total_len):
    prompt_text = ''

    for ex in train_fewshot_samples[i]:
        prompt_text += generate_fewshot_example(ex)

    prompt_text += generate_fewshot_example(test_data)

    pred = predict_by_generation(prompt_text)

    pred_tokens.append(pred)
    real_labels.append('긍정' if test_label == 1 else '부정')
```

Few-shot Learning 및 평가

- Few-shot Learning 실행
  - Few-shot 예제를 Prompt 텍스트로 구성하여 예측할 test 입력 데이터에 대한 감정을 예측

```
prompt_text = ''

# K개의 Few-shot 예제 구성
for ex in train_fewshot_samples[i]:
    prompt_text += generate_fewshot_example(ex)

# Test 입력 데이터 추가
prompt_text += generate_fewshot_example(test_data)

# 감정 예측
pred = predict_by_generation(prompt_text)
```

Few-shot Learning 및 평가

- NSMC Few-shot Learning 결과 평가
  - 예측한 label 토큰과 정답 label 토큰과 같은지 accuracy로 평가

```
accuracy_match = [p == t for p, t in zip(pred_tokens, real_labels)]
accuracy = len([m for m in accuracy_match if m]) / len(real_labels)

print(accuracy)
```

OUTPUT: 0.694……

Few-shot Learning 및 평가

- 다른 Prompt 템플릿으로 구성한다면?
  - Prompt 템플릿 양식에 따라 성능이 바뀔 수도 있음

```
def build_prompt_text(sent):
    return '다음 문장은 긍정일까요 부정일까요?\n' + sent + '\n정답:’
……
(생략)
……

accuracy_match = [p == t for p, t in zip(pred_tokens, real_labels)]
accuracy = len([m for m in accuracy_match if m]) / len(real_labels)

print(accuracy)
```

OUTPUT: 0.768……

실습 내용 정리

1. Few-shot 예제 구성
2. Few-shot Prompt 템플릿 텍스트 구성
3. 텍스트 토큰화 및 생성 과정
4. 다양한 템플릿 구성을 통한 성능 개선 가능

질의응답

<End of Document>
감사합니다.