

# 「나의 AI편친」

자연어 처리 모델을 활용한 개인화 AI휴먼

Team 연어유희

팀원: 이소담 이한 장세종 남궁지희



# Content

0 | 팀원 소개

1 | 프로젝트 진행 상황 리뷰

2 | 테크니컬 리뷰

3 | 진행 예정 프로세스 소개

4 | 시연 예시 / Q & A



# 0. 팀원소개



Team 연어유희

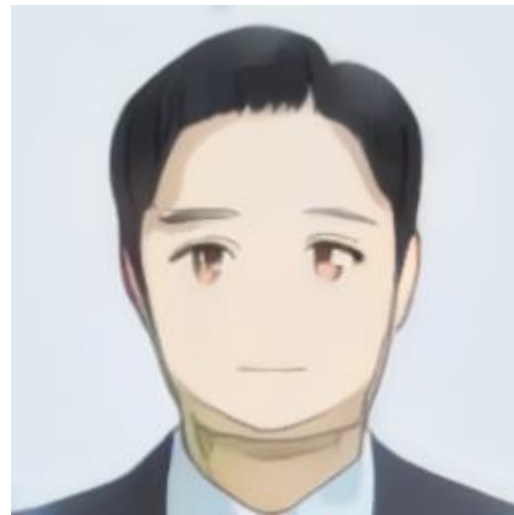
팀원: 이소담 이한 장세종 남궁지희



## 이소담

STT, TTS 음향

★금융데이터 분석



## 이 한

모델구성, 기획, 자료정리

★numerical 데이터 분석





## 장세종

GPT 파인 튜닝 주도

★자료수집



## 남궁 지희

데이터 전처리, 모델구성

★영어논문 분석

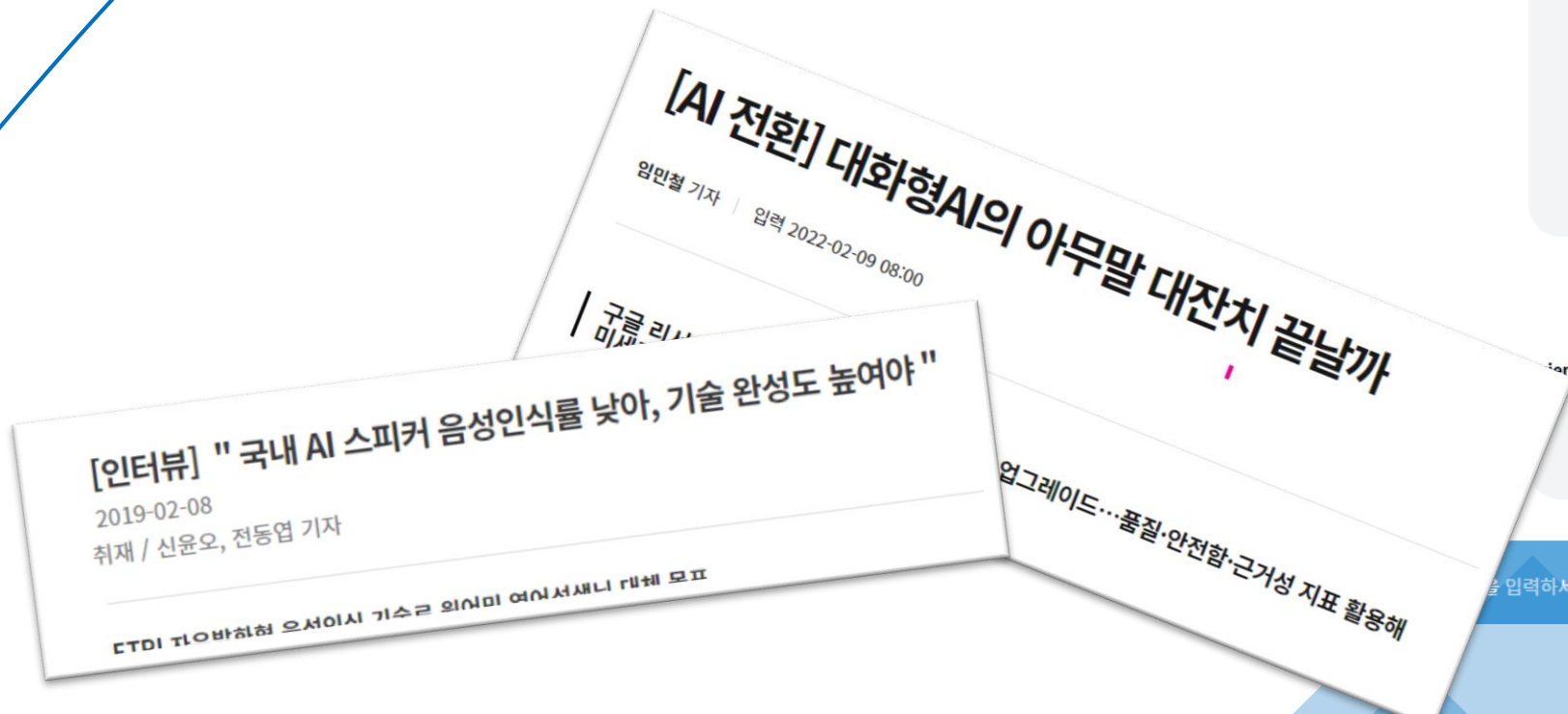
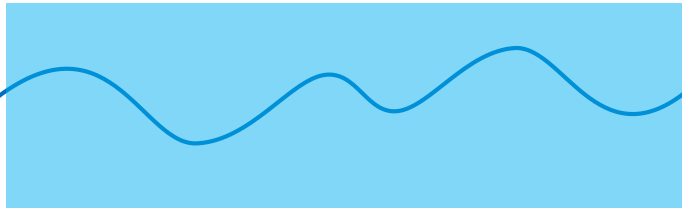
# 1. 프로젝트 진행 상황 리뷰



Team 연어유희

팀원: 이소담 이한 장세종 남궁지희

무엇을 고민하고  
어떤 한계점을 느꼈는가...



ScienceON Chatbot

오후 05:31

ScienceON Chatbot

네~ 저는 항상 질문을 기다리고있습니다~

오후 05:31

저의 이름은 최강지희 입니다

오후 05:32

ScienceON Chatbot

아래와 같은 말들은 잘 이해해요. 참고부탁드려요.

- 국가과학기술데이터본부 서비스 안내 >
- 어떤 R&D 과제가 있지? >
- 국가슈퍼컴퓨팅본부 대표서비스 >
- 검색 입력 TIP >

오후 05:32

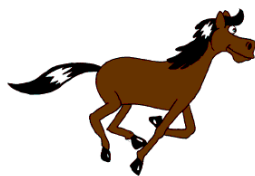
제 이름은 뭐예요?

오후 05:32

ScienceON Bot이라고 합니다.

오후 05:32

을 입력하세요.



2022. 3

2022. 4

2022. 5

## CONTENTS

2월 4주

3월 1주

3월 2주

3월 3주

3월 4주

4월 1주

4월 2주

4월 3주

4월 4주

5월 1주

비고

프로젝트 방향설정

모델선택

데이터수집

논문리뷰

모델구성

퓨전튜닝

테스트 및 개선

최종점검

TTS

STT(API)

Avatar

Kospeech

연동





BERT



GPT



BART

자연어이해(NLU)에  
강점 보유

자연어생성(NLG)에  
강점 보유

자연어이해(NLU)와  
자연어생성(NLG)에서  
모두 좋은 성능 보유

### BERT와 GPT의 강점을 합친 BART

GPT나 BERT는 Transformer\*의 일부분만 활용해서 좋은 성능을 발휘하는 분야가 한정된 반면, BART의 경우에는 Transformer의 모든 부분을 활용하기에 자연어이해(NLU)와 자연어생성(NLG)에서 모두 좋은 성능을 나타냅니다.



BERT



RoBERTa

기존 BERT의 학습을  
강화한 더 우수한 모델

### BERT의 성능을 극대화한 RoBERTa

RoBERTa는 기존 BERT의 학습을 강화하여 성능을 높인 모델로, 현재 가장 널리 쓰이는 PLM 중 하나입니다.



GPT-3



RoBERTa



PET

### GPT-3보다 더 나은 성능을 보여주는 PET

자연어이해(NLU) task에서는 RoBERTa를 통해 Few-shot learning 하는 방법인 PET(Pattern Exploiting Training)를 사용하면 GPT-3의 크기로 인한 비효율적인 측면을 보완할 수

나보다 나를 더 잘 아는

당신을 이해하고

당신에게 귀 기울이는 존재



“안녕👋”

난 너의 AI핀친  
유희지희야”

# 데이터 수집

대화내역 가져오기  
1만 줄 이상  
멀티턴 구현



KETI\_대화데이터\_일상\_오피스.txt

유효기간: ~2022.03.30  
용량: 88.84KB

열기 · 폴더 열기

학원 남궁지희

output\_daily\_1st.json

유효기간: ~2022.03.30  
용량: 1.37MB

저장 · 다른 이름으로 저장

output\_daily\_2nd.json

유효기간: ~2022.03.30  
용량: 974.96KB

열기 · 폴더 열기

output\_daily\_3rd.json

유효기간: ~2022.03.30  
용량: 493.77KB

저장 · 다른 이름으로 저장

output\_task.json

유효기간: ~2022.03.30  
용량: 5.46MB

저장 · 다른 이름으로 저장

readme.txt

유효기간: ~2022.03.30  
용량: 407bytes

저장 · 다른 이름으로 저장

아침 json으로 학습도 되나요?

output\_daily\_1st.json

유효기간: ~2022.03.30  
용량: 1.37MB

저장 · 다른 이름으로 저장

오후 1:12

김민지

넹 json으로 가져오면 되죠~!!

오후 1:15

김민지

Tts도 유튜브 보내주신거 봤는데 엄청 친절하게 설명해주네요?! Tts는 해볼만 한거 같아요!

다만 녹음이 좀 고생스러울거같네요 ㅠㅠ

오후 1:19

오후 1:20

오옷 넹~!

김민지

넹 감사합니다!!

오후 12:53

김민지

카톡데이터로 쓰리던 정도 한번 도전해볼까요?! 각자 카톡데이터 모아보면 만개정도 모을수 있으려나요??

오후 1:09

오후 1:10

오오! 그럴수도 있군요!

“현업에서는 어떤 논문을 보는가?”

## RoBERTa 논문 정리(논문 리뷰) - A Robustly Optimized BERT Pretraining Approach (RoBERTa)

포스팅 개요 이번 포스팅은 자연어처리(NLP) 논문 중 A Robustly Optimized BERT Pretraining Approach라는 논문을 리뷰하는 포스팅입니다. 해당 자연어처리 논문은 흔히 RoBERTa라고 많이 언급되는 논문인데요. 앞서 GPT-1, GPT-2, BERT 논문 리뷰에 이어서 자연어처리 논문 시리즈 네 번째 포스팅입니다. 추가로 해당 포스팅의 내용은 제가 진행하는 사내 자연어 처리 스터디에서 발표한 자료를 블로그로 정리한 자료임을 알려드립니다. 자연어 처리 논문...

deep learning(딥러닝) | 2021. 10. 25. 09:24



## BERT와 GPT의 강점을 합친 BART

GPT나 BERT는 Transformer\*의 일부분만 활용해서 좋은 성능을 발휘하는 분야가 한정된 반면, BART의 경우에는 Transformer의 모든 부분을 활용하기에 자연어이해(NLU)와 자연어생성(NLG)에서 모두 좋은 성능을 나타낸다.



## BERT의 성능을 극대화한 RoBERTa

RoBERTa는 기존 BERT의 학습을 강화하여 성능을 높인 모델로, 현재 가장 널리 쓰이는 PLM 중 하나입니다.



## GPT-3보다 더 나은 성능을 보여주는 PET

자연어이해(NLU) task에서는 RoBERTa를 통해 Few-shot learning 하는 방법인 PET(Pattern Exploiting Training)를 사용하면 GPT-3의 크기로 인한 비효율적인 측면을 보완할 수



## 2. 테크니컬 리뷰



Team 연어유희

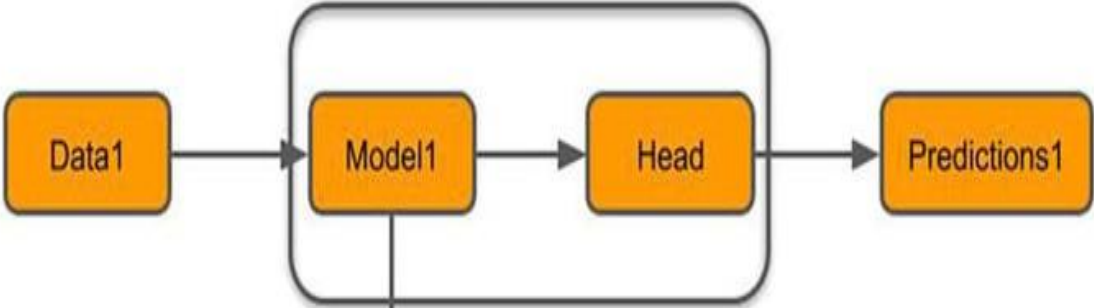
팀원: 이소담 이한 장세종 남궁지희



# 자연어 처리 모델 학습 방법

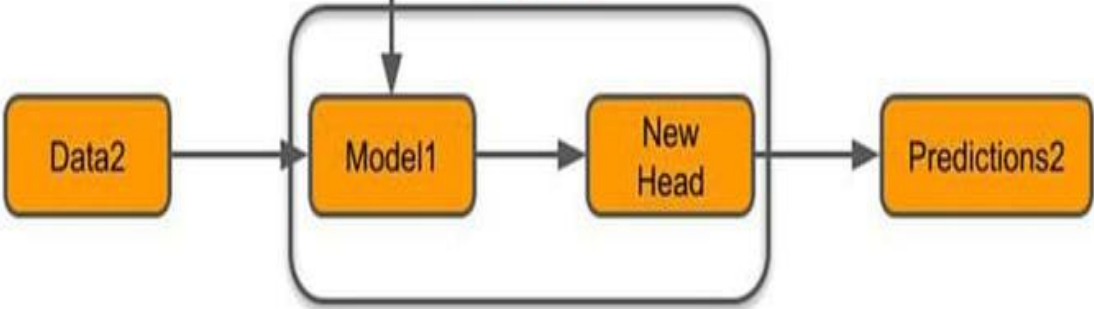
## Transfer Learning

Task 1 업스트림 태스크



Knowledge transfer

Task 2 다운스트림 태스크



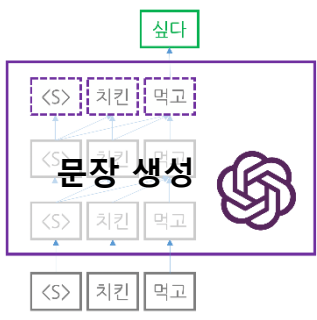
자기 지도 학습(self-supervised learning)  
파인튜닝(fine-tuning)  
프리트레이닝 마친 모델을 다운스트림 태스크 업데이트

훈련 방법

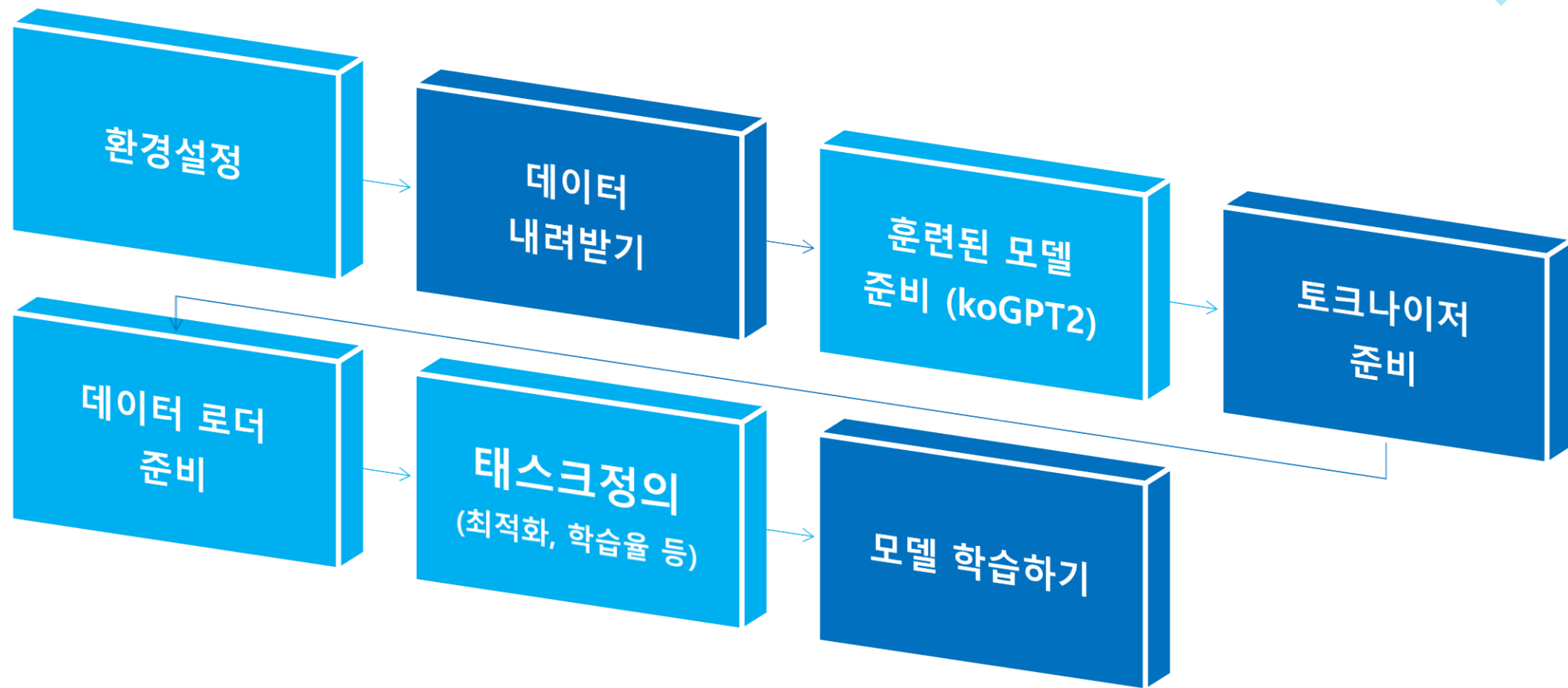
I	AM	a	?	GPT
I	AM	?	boy	BERT

다음 단어 맞추기  
(언어모델)  
빈칸 채우기  
(마스크 언어모델)

### 분류(classification)



# 학습 파이프라인



```
import torch
from transformers import GPT2LMHeadModel
from transformers import PreTrainedTokenizerFast

import numpy as np
import pandas as pd
import torch
from pytorch_lightning import Trainer
from pytorch_lightning.callbacks import ModelCheckpoint
from pytorch_lightning.core.lightning import LightningModule
from torch.utils.data import DataLoader, Dataset
from transformers.optimization import AdamW, get_cosine_schedule_with_warmup
from transformers import PreTrainedTokenizerFast, GPT2LMHeadModel
import re
```

환경설정하기

```

Chatbot_Data = pd.read_csv(path + "ChatBotData_지희.csv")
class ChatbotDataset(Dataset): # 자동으로 데이터를 불러오는 클래스
    def __init__(self, chats, max_len=40): # 데이터셋의 전처리를 해주는 부분
        self._data = chats # 챗봇 데이터
        self.max_len = max_len # 최대 길이를 저장한다.
        self.q_token = Q_TKN # 질문
        self.a_token = A_TKN # 대답
        self.sent_token = SENT # 감정
        self.eos = EOS # 문장의 끝을 나타내는 token
        self.mask = MASK # 마스크를 나타내는 token
        self.tokenizer = koGPT2_TOKENIZER # 데이터를 불러오는 부분

    def __len__(self): # chatbotdata 의 길이를 리턴한다.
        return len(self._data) # 데이터의 길이를 리턴한다.

```

## 데이터 내려받기

```

model = GPT2LMHeadModel.from_pretrained('skt/kogpt2-base-v2')
koGPT2_TOKENIZER = PreTrainedTokenizerFast.from_pretrained("skt/kogpt2-base-v2",
bos_token=BOS, eos_token=EOS, unk_token="<unk>", pad_token=PAD, mask_token=MASK,)

```

## Pre-Trained 모델 준비하기

```

Q_TKN = "<usr>" # 질문
A_TKN = "<sys>" # 대답
BOS = "</s>" # 문장의 시작을 나타내는 token
EOS = "</s>" # 문장의 끝을 나타내는 token
PAD = "<pad>"
SENT = '<unused1>' # 감정
MASK = "<unused0>"
...

bos_token : 문장의 시작을 나타내는 token
eos_token : 문장의 끝을 나타내는 token
unk_token : 모르는 단어를 나타내는 token
pad_token : 동일한 batch 내에서 입력의 크기를 동일하게 하기 위해서 사용하는 token
PreTrainedTokenizer 에서 제공되는 함수는
tokenize() : tokenizer를 이용해서 string을 token id의 리스트로 변환한다.
get_added_vocab() : token to index에 해당하는 dict를 리턴한다.
batch_decode() : token id로 구성된 입력을 하나의 연결된 string으로 출력한다.
convert_ids_to_tokens() : token id 의 리스트를 token으로 변환한다. skip_special_tokens=True로 하면 decoding할 때 special token들을 제거한다.
convert_tokens_to_ids() : token string의 리스트를 token id 또는 Token id의 리스트로 변환한다.
decode() : tokenizer 와 vocabulary를 이용해서 token id를 string으로 변환한다. skip_special_token=True로 지정하면 speical token들을 제외한다.
encode() : token string을 token id 의 리스트로 변환한다. add_special_tokens=False로 지정하면 token id로 변환할 때 special token들을 제외한다.
padding을 통해서 padding token을 어떻게 추가할지도 지정할 수 있다.
...

koGPT2_TOKENIZER = PreTrainedTokenizerFast.from_pretrained("skt/kogpt2-base-v2",
                                                         bos_token=BOS, eos_token=EOS, unk_token="<unk>", pad_token=PAD, mask_token=MASK,)
model = GPT2LMHeadModel.from_pretrained('skt/kogpt2-base-v2')
path = 'C:\\\\Users\\bitcamp\\Desktop\\TP\\'

```

## 토크나이저 준비하기



```

class ChatbotDataset(Dataset): # 자동으로 데이터를 불러오는 클래스
    def __getitem__(self, idx): # 로드한 챗봇 데이터를 차례차례 DataLoader로 넘겨주는 메서드
        turn = self._data.iloc[idx] # 챗봇 데이터의 인덱스를 가져온다.
        q = turn["Q"] # 질문을 가져온다.
        q = re.sub(r"([?.!,])", r" ", q) # 구두점들을 제거한다.
        a = turn["A"] # 답변을 가져온다.
        a = re.sub(r"([?.!,])", r" ", a) # 구두점들을 제거한다.
        q_toked = self.tokenizer.tokenize(self.q_token + q + self.sent_token)
        # 질문 + 감정 # 질문을 토큰나이징한다.
        q_len = len(q_toked) # 질문의 길이를 구한다.
        a_toked = self.tokenizer.tokenize(self.a_token + a + self.eos)
        # 대답 + 끝 # 대답을 토큰나이징한다.
        a_len = len(a_toked) # 대답의 길이를 구한다.

        if q_len > self.max_len: # 질문의 길이가 최대길이보다 크면
            a_len = self.max_len - q_len #답변의 길이를 최대길이 - 질문길이
            if a_len <= 0: #질문의 길이가 너무 길어 질문만으로 최대 길이를 초과 한다면
                q_toked = q_toked[-(int(self.max_len / 2)) :] #질문길이를 최대길이의 반으로
                q_len = len(q_toked) #질문의 길이를 구한다.
                a_len = self.max_len - q_len #답변의 길이를 최대길이 - 질문길이
            a_toked = a_toked[:a_len] #답변의 길이만큼 대답을 가져온다.
            a_len = len(a_toked) #답변의 길이를 구한다.

```

## 데이터로더 준비하기

데이터 로더는 학습 때 데이터를 배치(batch) 단위로 모델에 할당하는 역할로 전체 데이터 가운데 일부 인스턴스를 뽑아(sample) 배치를 구성. 데이터셋(dataset)은 데이터 로더의 구성 요소 중 하나

```
class ChatbotDataset(Dataset): # 자동으로 데이터를 불러오는 클래스
    def __getitem__(self, idx): # 로드한 챗봇 데이터를 차례차례 DataLoader로 넘겨주는
        메서드
```

```
        if q_len + a_len > self.max_len: #질문길이 + 답변길이가 최대길이보다 크면
            a_len = self.max_len - q_len #답변의 길이를 최대길이 - 질문길이
            if a_len <= 0: #질문의 길이가 너무 길어 질문만으로 최대 길이를 초과 한다면
                q_toked = q_toked[-(int(self.max_len / 2)) :] #질문길이를 최대길이의 반으로
                q_len = len(q_toked) # 질문의 길이를 구한다.
                a_len = self.max_len - q_len #답변의
            a_toked = a_toked[:a_len] #답변의 길이만큼
            a_len = len(a_toked) # 답변의 길이를
        # 답변 labels = [mask, mask, ....., mask, ..., <bos>,...답변.
        # 답변의 길이만큼 마스크를 추가한다.
        labels = [self.mask,] * q_len + a_toked[1:] # 마스크를 질문

        # mask = 질문길이 0 + 답변길이 1 + 나머지 0
        mask = [0] * q_len + [1] * a_len + [0] * (self.max_len - q_len - a_len)
        # 답변의 길이만큼 1을 넣어준다.
        # 답변 labels을 index 로 만든다.
        labels_ids = self.tokenizer.convert_tokens_to_ids(labels)
```

```
class ChatbotDataset(Dataset): # 자동으로 데이터를 불러오는 클래스
    def __getitem__(self, idx): # 로드한 챗봇 데이터를 차례차례
        DataLoader로 넘겨주는 메서드
```

```
        # 최대길이만큼 PADDING
        while len(labels_ids) < self.max_len:
            labels_ids += [self.tokenizer.pad_token_id]

        # 질문 + 답변을 index 로 만든다.
        token_ids = self.tokenizer.convert_tokens_to_ids(q_toked + a_toked)
        # 질문과 답변을 index로 변환한다.

        # 최대길이만큼 PADDING
        while len(token_ids) < self.max_len: # 질문길이 + 답변길이만큼 넣어준다.
            token_ids += [self.tokenizer.pad_token_id] # PADDING

        #질문+답변, 마스크, 답변
        return (token_ids, np.array(mask), labels_ids)
        # 답변을 index로 변환한다.
```

## 데이터로더 준비하기

자연어 처리 모델은 계산 가능한 형태, 즉 숫자 입력을 받으므로 각 토큰을 그에 해당하는 정수로 변환하는 인덱싱(indexing)을 수행, 인덱싱은 보통 토큰나이저가 토큰화와 함께 작업

```
path = 'C:\\Users\\bitcamp\\Desktop\\TP\\'
Chatbot_Data = pd.read_csv(path + "ChatBotData_지희.csv")
train_set = ChatbotDataset(Chatbot_Data, max_len=40)

train_dataloader = DataLoader(train_set, batch_size=32, num_workers=0, shuffle=True,
collate_fn=collate_batch,)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model.to(device)
model.train()

learning_rate = 3e-5
criterion = torch.nn.CrossEntropyLoss(reduction="none") # 기준
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

epoch = 10
Sneg = -1e18
print ("start")
```

태스크 정의

```

print ("start")
for epoch in range(epoch):
    for batch_idx, samples in enumerate(train_dataloader): # 학습 데이터를 불러온다.
        optimizer.zero_grad() # 그라디언트 초기화
        token_ids, mask, label = samples # 학습 데이터를 불러온다.(질문+답변, 마스크, 라벨)
        out = model(token_ids) # out : (batch_size, max_len, vocab_size)
        out = out.logits #Returns a new tensor with the logit of the elements of input
        mask_3d = mask.unsqueeze(dim=2).repeat_interleave(repeats=out.shape[2], dim=2)
        # 그래서 어느 차원에 1인 차원을 생성할 지 꼭 지정해주어야한다.
        # unsqueeze : 차원을 추가한다, repeat_interleave: 차원을 추가한다
        mask_out = torch.where(mask_3d == 1, out, Sneg * torch.ones_like(out))
        #torch.ones_like(out): 같은 shape의 tensor생성, mask_out은 mask_3d가 1인 경우 out(출력), mask가 0인 경우 Sneg
        loss = criterion(mask_out.transpose(2, 1), label)
        # criterion : 손실함수, transpose : 차원을 바꾼다.
        avg_loss = loss.sum() / mask.sum()
        avg_loss.backward() # backward : 역전파
        # 학습 끝
        optimizer.step() # step : 학습
print ("end")

```

모델 학습하기

```

sent = '0' # 감성어 추출을 위한 변수
with torch.no_grad(): # no_grad : 역전파를 하지 않는다
    while 1: # 무한 반복
        q = input("user > ").strip() # 사용자 입력 # 인자로 전달된 문자를 String의 왼쪽과 오른쪽에서 제거합니다. strip() 함수 : 문자열 앞뒤의 공백 또는 특별한 문자 삭제.
        if q == "quit": # quit : 종료
            break # 반복문 종료
        a = "" # 답변
        while 1: # 무한반복
            input_ids = torch.LongTensor(koGPT2_TOKENIZER.encode(Q_TKN + q + SENT + sent + A_TKN + a)).unsqueeze(dim=0) # encode: 입력문자를 인코딩
            pred = model(input_ids) # input_ids를 넣어서 예측
            pred = pred.logits # 출력값을 logit으로 변환
            gen = koGPT2_TOKENIZER.convert_ids_to_tokens(torch.argmax(pred, dim=-1).squeeze().numpy().tolist())[-1]
            # argmax : 최댓값을 찾는다, squeeze : 차원을 줄인다, numpy : numpy로 변환 # tolist()를 사용하면 반환 타입은 list
            if gen == EOS: # EOS : 문장의 끝
                break # 반복문 종료
            a += gen.replace("_", " ") # 답변 출력
        print("Chatbot > {}".format(a.strip())) # 답변 출력

```

답변 출력



## 1. zero\_grad()

**Gradient 초기화**  
= Sets gradients of all model parameters to zero.

파이토치는 미분을 통해 얻은 기울기를 이전에 계산된 기울기 값에 누적시키는 특징이 있습니다.

```
import torch  
w = torch.tensor(2.0, requires_grad=True)
```

“  
Neural Network model 인스턴스를 만든 후, 역전파 단계를 실행하기 전에 변화도를 0으로 만든다.

autograd의 추적기록을 피하기 위해 학습 가능한 매개변수를 갖는 Tensor를 직접 0으로 조작하여 모델의 가중치를 갱신할 때 사용한다.

.backward()를 호출할 때마다 변화도가 누적되기 때문에 변화도를 업데이트 하기 전에 초기화

수식을 w로 미분한 값	: 2.0
수식을 w로 미분한 값	: 4.0
수식을 w로 미분한 값	: 6.0
수식을 w로 미분한 값	: 8.0
수식을 w로 미분한 값	: 10.0
수식을 w로 미분한 값	: 12.0
수식을 w로 미분한 값	: 14.0
수식을 w로 미분한 값	: 16.0
수식을 w로 미분한 값	: 18.0
수식을 w로 미분한 값	: 20.0
수식을 w로 미분한 값	: 22.0
수식을 w로 미분한 값	: 24.0
수식을 w로 미분한 값	: 26.0
수식을 w로 미분한 값	: 28.0
수식을 w로 미분한 값	: 30.0
수식을 w로 미분한 값	: 32.0
수식을 w로 미분한 값	: 34.0
수식을 w로 미분한 값	: 36.0
수식을 w로 미분한 값	: 38.0
수식을 w로 미분한 값	: 40.0
수식을 w로 미분한 값	: 42.0

## 2. unsqueeze/squeeze

Squeeze 함수

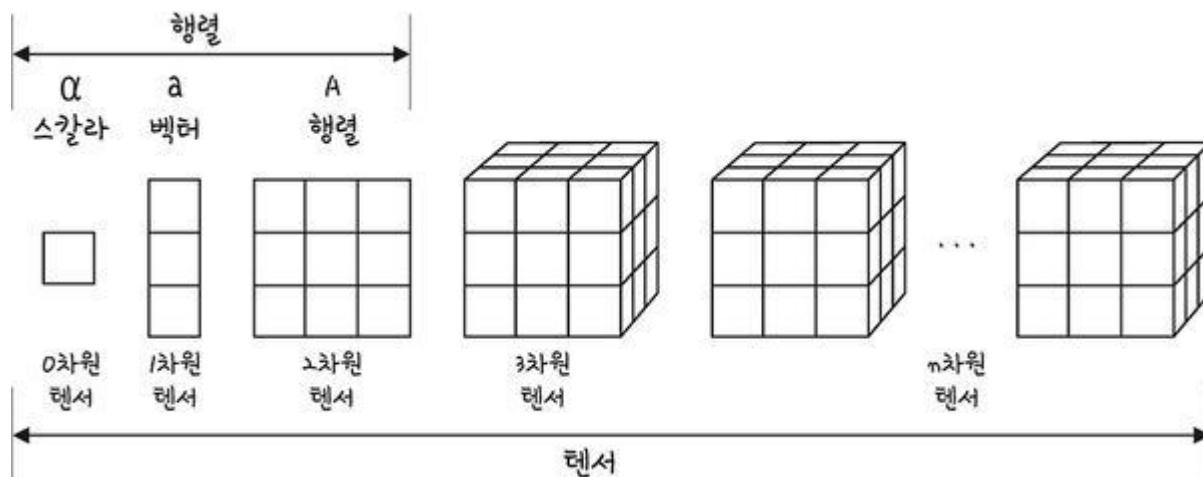
1) 1인 차원을 제거

Unsqueeze 함수

1) 1인 차원을 생성하는 함수(생성 위치 지정)

```
x = torch.rand(3, 1, 20, 128)
x = x.squeeze() #[3, 1, 20, 128] -> [3, 20, 128]
```

```
x = torch.rand(3, 20, 128)
x = x.unsqueeze(dim=1) #[3, 20, 128] -> [3, 1, 20, 128]
```



행렬은 레코드가 여럿인 데이터 집합  
텐서는 크기가 같은 행렬이 여러 개 있는 것

자연어 처리는 보통 (batch size, 문장 길이, 단어 벡터의 차원)이라는 3차원 텐서를 사용

### 3. `zero_grad()`, `loss.backward()`, `optimizer.step()`

`loss.backward()`

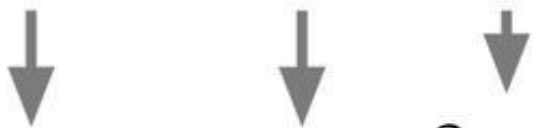
- 모델의 매개변수들에 대한 손실의 변화도를 계산

`optimizer.step()`

- `step` 함수를 호출하면 역전파 단계에서 수집된 변화도로 매개변수를 조정

학습 단계(loop)에서 최적화는 세단계로 이뤄집니다:

- `optimizer.zero_grad()` 를 호출하여 모델 매개변수의 변화도를 재설정합니다. 기본적으로 변화도는 더해지기(add up) 때문에 중복 계산을 막기 위해 반복할 때마다 명시적으로 0으로 설정합니다.
- `loss.backward()` 를 호출하여 예측 손실(prediction loss)을 역전파합니다. PyTorch는 각 매개변수에 대한 손실의 변화도를 저장합니다.
- 변화도를 계산한 뒤에는 `optimizer.step()` 을 호출하여 역전파 단계에서 수집된 변화도로 매개변수를 조정합니다.


$$W(t+1) = W_t - \frac{\partial \text{오차}}{\partial W}$$

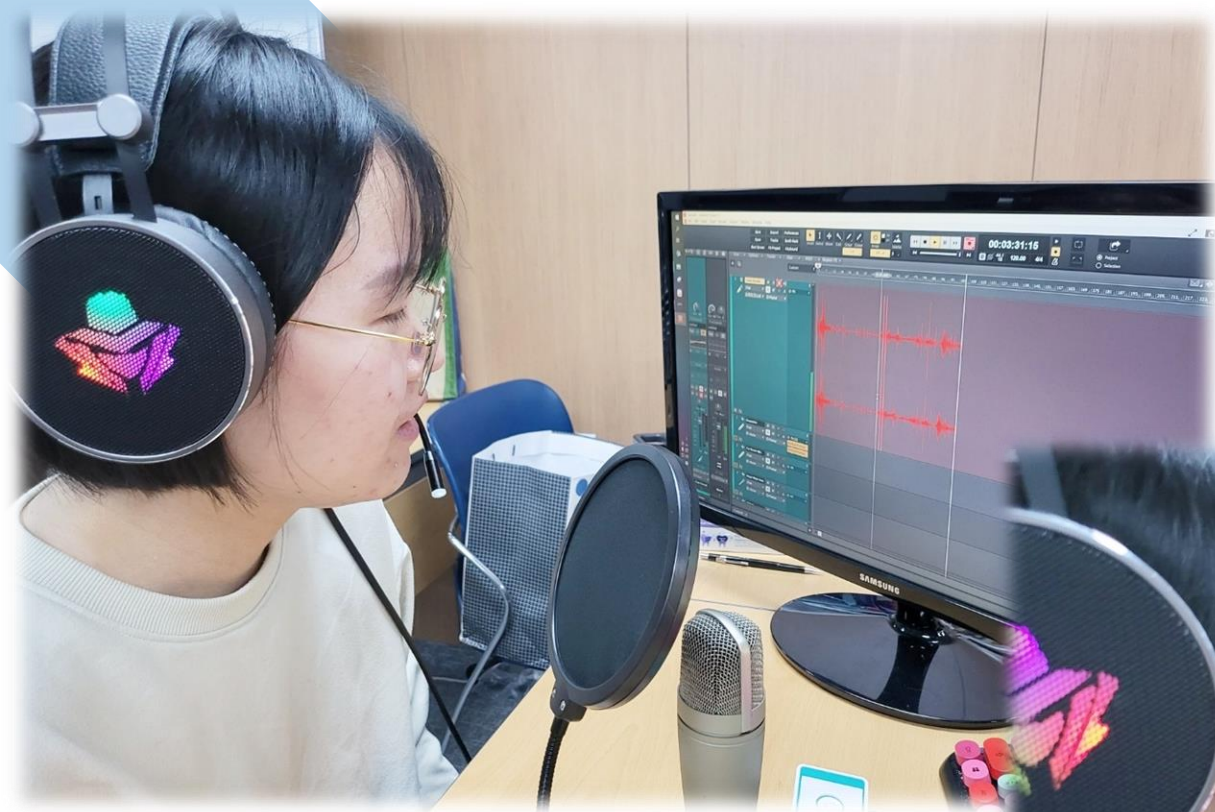
### 3.진행 예정 프로세스 소개



Team 연어유희

팀원: 이소담 이한 장세종 남궁지희







# 멀티턴 챗봇 구현

ScienceON Chatbot

네~ 저는 항상 질문을 기다리고있습니다~  
오후 05:31

오후 05:32

저의 이름은 최강지희 입니다

ScienceON Chatbot

아래와 같은 말들은 잘 이해해요. 참고부탁드려요.

- 국가과학기술데이터본부 서비스 안내 >
- 어떤 R&D 과제가 있지? >
- 국가슈퍼컴퓨팅본부 대표서비스 >
- 검색 입력 TIP >

오후 05:32

오후 05:32

제 이름은 뭘까요?

ScienceON Chatbot

ScienceON Bot이라고 합니다.  
오후 05:32



## 문맥을 이해하는 답변

마지막 대화를 “그렇까?”로 고정하고, 문맥변화

우울해	신난다
무슨 일 있어요?	무슨 일 있어요?
크리스마스인데 나만 솔로야	오늘 날씨가 너무 좋아
좋은 사람 만날 수 있을 거예요	한강이라도 가요
그렇까?	그렇까?
멀티턴 리액션 😊	멀티턴 리액션 😊
네 그럴 거예요 (0.4163)	같이 갈래요? (0.4062)

문맥을 반영하면서 답변이 가능하다 😊

오! 문맥이 달라지면 답변이 달라져! 😊

# 유희지희(챗봇) 스킨 구현

Python3

```
# importing the require package
from py_avataaars import PyAvataaar

# assigning various parameters to our avatar
avatar = PyAvataaar()

# rendering the avatar in png format
avatar.render_png_file("AVATAR_1.png")
```

Output:



The above program will generate the *AVATAR\_1.png* file in the folder where you've kept the above program. If the program is running properly, then we will generate the avatars according to our *PyAvataaar()* method.

Syntax:



# Q&A



Team 연어유희

팀원: 이소담 이한 장세종 남궁지희