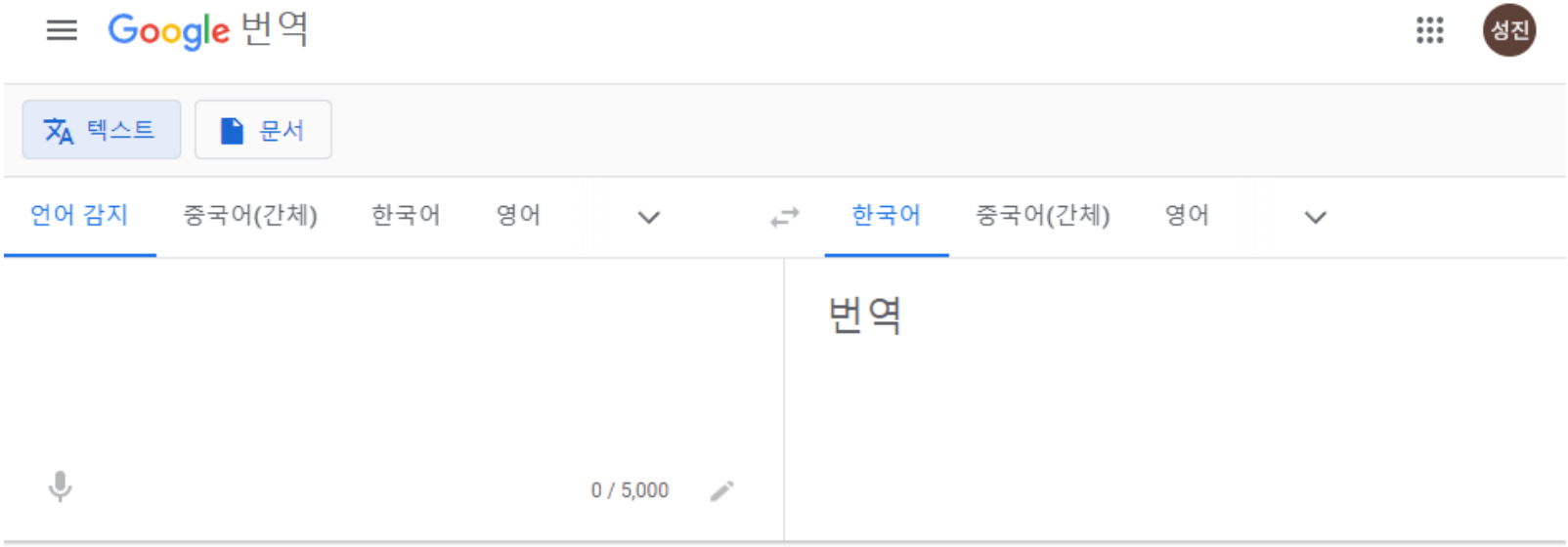
Multi-label classification

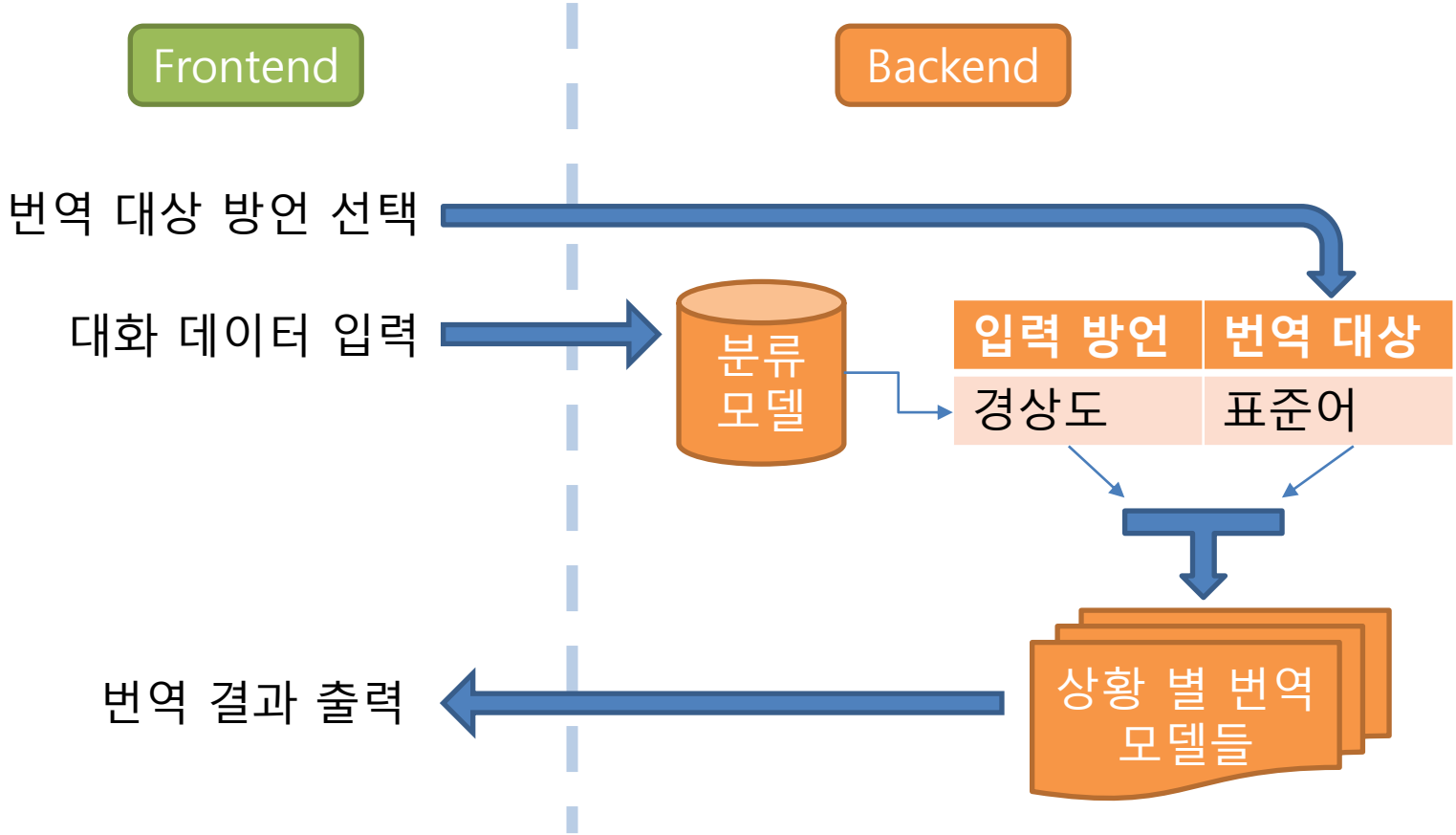
2022.11.10. HAI 1팀

우리가 만들 친구는 어떤 프로그램인가?

- 사람의 발화 텍스트를 입력하면, 표준어인지 아니면 특정 지역 방언인지 자동으로 분류해줘요(구글 번역기의 언어 자동 인식 기능).
- 현재 입력된 텍스트를 원하는 지역의 방언이나 표준어로 번역할 수 있어요.
- 웹 애플리케이션으로 구현하여 누구나 접근해서 사용 가능해요.



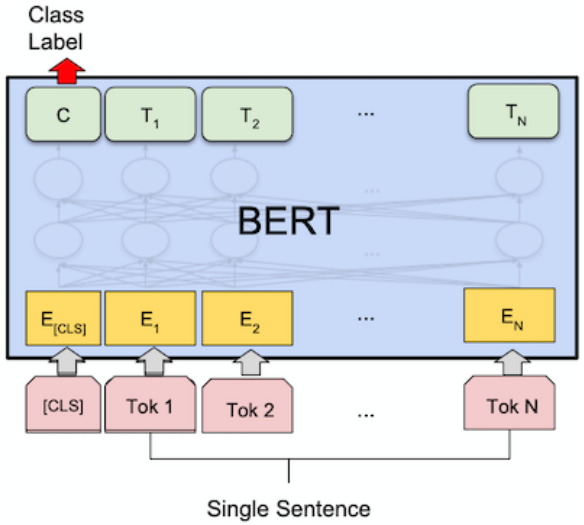
실제 개발해야 할 구조



지난 시간에 만들었던 모델을 업그레이드 해 봅시다!

Input: 대화 텍스트
Ex) 마! 니 밥 묵었나?

Output: 방언 라벨
Ex) 표준어: 0, 경상도: 1, 제주도: 2, 전라도: 3



Step 1. 데이터 준비

- Git repository에 표준어와 경상도 방언 뿐만 아니라 제주도와 전라도 방언 역시 구분할 수 있도록 총 4가지의 label을 가지는 발화 데이터셋을 만들어 업로드해 놓았어요.
- 지난 시간에는 표준어와 경상도 두 가지만 분류할 수 있는 모델이었는데, 이번에는 총 4가지 지역을 분류해야 하니 조금 더 어려운 task라고 할 수 있죠.
- 기존 표준어-경상도 데이터셋을 만드는 코드와 거의 동일한 코드를 사용했고 데이터셋을 생성하기 위한 코드는 multi_label_classification/datasets/sources 디렉토리 내부의 소스 코드를 참조하세요.

contents	label
막 이렇게 되고 아 이제까지 나하고 그랬던 거 다 뭐지?	**0** (표준어)
맞다 가시나야 요즘 어쩌케 사냐 맨날 이렇게 얘기 하고	**1** (경상도)
무슨 말인가 무슨 말이꽈 햄신게 눈 눈 보랭이 떠그넹예	**2** (제주도)
딸 있응께 여간 좋다 내가 그걸 느꼈당께 언니는 좋겄다	**3** (전라도)

Step 2. 모델 학습

- https://colab.research.google.com/github/GirinMan/HAI-DialectTranslator/blob/main/multi_label_classification/train/train_classifier.ipynb
- 생성된 데이터셋을 이용해 4개의 라벨을 분류할 수 있는 모델을 만드는 파이썬 노트북 입니다. Google colab에서 바로 실행 가능해요!
- 지난 주에 봤던 표준어/경상도 방언을 분류하는 모델을 학습 시키는 코드와 유사하지만 라벨의 개수가 달라져 num_labels 파라미터를 4로 설정한 것을 볼 수 있습니다.
- 실제로 학습을 진행해 보면 validation accuracy가 매우 낮게 나오는 것을 볼 수 있습니다. 이정도 정확도는 차라리 랜덤하게 뽑는게 더 나을 지경이네요. 어떻게 하면 정확도를 올릴 수 있을까요?

```
start training

training epoch 0: 100%|██████████| 7588/7588 [05:49<00:00, 21.72it/s]

start predict

1720it [00:26, 63.83it/s]

Epoch 0,  Average training loss: 0.7127 , Validation accuracy : 0.1751
```

Step 3. Inference

- https://colab.research.google.com/github/GirinMan/HAI-DialectTranslator/blob/main/multi_label_classification/train/load_infer_model.ipynb
- 학습이 완료된 모델을 가져와서 inference를 수행할 수 있는 코드입니다.
- 역시 지난 주에 사용했던 추론 과정과 매우 유사하고, 모델의 weight를 불러올 때 num_labels를 4로 지정하고 학습 과정에서 사용했던 것과 동일한 모델 체크포인트를 사용해야 정상적으로 불러올 수 있어요.
- 각자 학습시킨 모델을 이용하여 입력된 텍스트를 자동으로 분류해주는 작업(실제 서비스에서 동작해야 하는 부분)이 어떻게 이루어지는지 확인해 봅시다.

```
def infer(model, tokenizer, input_txt):
    input_tensor = torch.tensor([tokenizer.encode(input_txt)]).to(device)

    with torch.no_grad():
        preds = model(input_tensor).logits.cpu()

    result = np.argmax(preds, axis=1).item()

    region = ["경상도", "제주도", "전라도"]

    print('입력된 문장 "' + input_txt + '"은/는 ', end='')
    if result:
        print(region[result-1], "방언입니다.")
    else:
        print("표준어 발화입니다.")
```

더 나은 성능의 모델 만들기

- 다음 회합 전 까지, 각자 팀 repository를 fork한 뒤 기본적으로 제공된 코드를 수정하여 baseline보다 더 성능이 향상된 모델을 만들어 봅시다. 완성된 모델은 학습 과정에 사용된 체크포인트, 하이퍼파라미터와 함께 저장하여 메일로 보내주세요!
- 모델의 학습에 영향을 미치는 것은 어떤 것이 있을까요? 학습 과정에서 Validation accuracy가 최대한 높아지도록 해 봅시다.

HINT 1: 모델에 입력될 최대 길이(**maxlen**), 학습을 진행할 **epoch** 횟수, **DataLoader**가 데이터를 샘플링하는 방식(**Sampler**), 한 번에 학습을 진행하는데 사용될 배치 사이즈(**batch**) 또는 데이터 자체의 전처리 방식 등 다양한 시도를 하며 모델을 학습시켜 봐요!

HINT 2: 우리가 이전에 BERT에 대해 배웠던 것을 기억해 보면, 어떤 **PLM**(Pre-trained language model)을 사용하느냐 에 따라서 성능이 크게 달라질 수 있어요. 한국어 발화 데이터인 만큼, Huggingface hub에서 **한국어**로 **pre-train**된 BERT 계열 모델을 사용하는 것이 아마 더 좋은 성능을 낼 거에요.

Appendix. Git을 이용한 협력 작업

- Git을 활용하여 여러 사람들이 한 repository에서 작업을 하기 위해서는, 기존 repository를 fork하고 이를 바탕으로 작업하는 방식에 대해 익숙해져야 해요.

- 기존 repository를 fork하여 자신의 git 계정에 가져온 다음, 해당 repository를 로컬 pc에 clone받아 작업합니다. 원본 repository의 수정사항을 받아오기 위해, fork하여 작업중인 로컬 git repository에서 원본 git 링크를 remote branch(보통 upstream이라는 이름을 사용함)로 추가하면 됩니다.

- Fork된 repository의 수정 사항을 원본에 반영하고 싶다면, commit한 내용을 바탕으로 pull request를 진행하도록 합시다.

- multi_label_classification/train 폴더 아래에 train_classifier.ipynb 파일을 각자 하나씩 복사하여 train_classifier_이름.ipynb 형태로 변형한 뒤 작업하고, 적절히 성능이 잘 나왔다면 pull request를 만들어 원본 repository에 반영해보도록 합시다.

Appendix. Git을 이용한 협력 작업

- 기타 Git을 활용한 fork, clone, remote branch 추가 및 pull request를 하는 방법은 아래 링크를 참고해주세요. 또는 우리 Slack 채널에 언제든지 질문해도 됩니다!!

- github 에서 fork 후 작업하기 https://www.lesstif.com/gitbook/github-fork-20775062.html
- Git fork와 clone 의 차이점 https://velog.io/@imacoolgirlyo/Git-fork%EC%99%80-clone-%EC%9D%98-%EC%B0%A8%EC%9D%B4%EC%A0%90-5sjuhwfzgp
- git 초보를 위한 풀리퀘스트(pull request) 방법 https://wayhome25.github.io/git/2017/07/08/git-first-pull-request-story/