Wrap Up Report

<기술적인 도전>

LB 점수 0.732, 115등

검증(Validation) 전략

제공된 Dataset에 대해 KFold 5를 적용했습니다. 이것을 적용하기 위해 Dataset Class에서 해당 데이터의 label도 따로 모아 후에 학습할 때 넘겨줬습니다.

사용한 모델 아키텍처 및 하이퍼 파라미터

1. 아키텍처: bert-multilingual-base-cased
    A. LB 점수 : 0.7320
    B. training time augmentation
        max length : 150
    C. 추가 시도
        - Batch Size를 처음에는 32으로 시작했다가 16, 64 등 다양한 batch를 시도해 최종적으로 32를 적용해 Local minimum에 빠지는 문제를 해결해 Accuracy를 올렸습니다.
        - Learning rate를 1e-5, 5e-05, 5e-06등 다양하게 적용을 해 가장 나은 값이 1e-05라는 것을 알게 되었습니다.
        - Optimizer 부분에서 Adam과 AdamW을 두고 여러 번 반복 시도한 결과 AdamW의 결과가 좋다는 결론을 얻었습니다.

2. 직접 Model의 Pipeline을 구성

```
def train():
  split = 5
  #skfold = StratifiedKFold(n_splits=split)
  kfold = KFold(n_splits=split, shuffle=True, random_state=100)
  # load model and tokenizer
  MODEL_NAME = "xlm-roberta-base"
  tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

  # load dataset
  dataset = load_data("/opt/ml/input/data/train/train.tsv")
  # for idx, num in dataset['label'].value_counts().items():
  #   if num < split:
  #     # df1 = dataset[dataset['label'] == idx].sample(n=split-num, replace=True, random_state=
  #     # dataset = dataset.append(df)
  #     dataset.drop(dataset[dataset['label']==idx].index, inplace=True)
  labels = dataset['label'].values
  # tokenizing dataset
  # make dataset for pytorch.
  for fold, (train_idx, valid_idx) in enumerate(kfold.split(dataset)):
    train_subsampler = torch.utils.data.SubsetRandomSampler(train_idx)
    valid_subsampler = torch.utils.data.SubsetRandomSampler(valid_idx)

    train_labels = labels[train_idx]
    tokenized_train_data = tokenized_dataset(dataset.iloc[train_idx], tokenizer)
    RE_train_dataset = RE_Dataset(tokenized_train_data, train_labels)

    dev_labels = labels[valid_idx]
    tokenized_dev_data = tokenized_dataset(dataset.iloc[valid_idx], tokenizer)
    RE_dev_dataset = RE_Dataset(tokenized_dev_data, dev_labels)
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

    # setting model hyperparameter
    bert_config = BertConfig.from_pretrained(MODEL_NAME)
    bert_config.num_labels = 42
    model = BertForSequenceClassification.from_pretrained(MODEL_NAME, config=bert_config)
    model.to(device)
```

```
for epoch in range(EPOCHS):
    train_loss, valid_loss = AverageMeter(), AverageMeter()

    model.train()
    for iter, batch in enumerate(train_loader):
        optim.zero_grad()
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        train_labels = batch['labels'].to(device)
        outputs = model(input_ids, attention_mask=attention_mask, labels=train_labels)
        loss = outputs[0]
        loss.backward()
        optim.step()

        train_loss.update(loss.item(), len(train_labels))
        print("\rEpoch [%3d/%3d] | Iter [%3d/%3d] | Train Loss %.4f" % (epoch+1, EPOCHS, iter+

    model.eval()
    correct, total = 0, 0
    for batch in valid_loader:
      with torch.no_grad():
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        valid_labels = batch['labels'].to(device)
        outputs = model(input_ids, attention_mask=attention_mask, labels = valid_labels)
        logits = outputs[1]
        logits = torch.argmax(logits, dim=1)
        correct += (logits==valid_labels).sum().item()
        total += len(valid_labels)
        valid_loss.update(loss.item(), len(valid_labels))

    log_dir='./results'
    accr_val = (100*correct / total)
    print("Epoch [%3d/%3d] | Valid Loss %.4f | Accuracy : %.2f" % (epoch+1, EPOCHS, valid_loss
    if best_accr < accr_val:
      best_accr = accr_val
      if not os.path.exists(log_dir):
```

A. 주어진 Baseline에 있는 TrainArgument와 Trainer를 이용하는 것은 사용에 간편하였지만 제가 원하는 방법을 적용하거나 Hyper Parameter 수정에 어려움을 느꼈습니다.

B. 관련 자료를 보더라도 각 argument의 코드 작동이나 연결에 대한 설명에 이해가 어려워 제가 직접 Train과 Valid에 대한 코드를 구현하기로 했습니다,

C. 모델 구성에 성공해 Train을 적용해 매 checkpoint 마다 model을 저장하는 것이 아닌 Valid에 대한 Best model만을 저장하도록 해 저장 공간 사용을 줄일 수 있고 쉽게 파라미터 수정이 가능해졌습니다.

3. 앙상블(Ensemble) 방법

각 KFold 모델 마다 나온 예측 Label을 K로 나눠 ans에 저장한 뒤 최종적으로 Test가 끝나면 해당 sentence entity relation에 대한 예측 분포에서 Argmax를 이용한 가장 큰 값을 정답으로 사용하였습니다.

4. 시도했으나 잘 되지 않았던 것들

1. 다양한 Hyper Parameter의 수정을 통해 Model의 성능을 올리려 했으나 결국 중요한 건 Data의 다양성과 Base Model의 Task 적합도라는 결론이 나왔습니다.
2. Model을 변환하는 과정에서 중요한 부분의 코드가 변환이 되지 않았는지 기존에 주어진 Baseline Trainer에 비해 같은 Parameter여도 성능이 좋지 않았습니다.