Natural Language Processing

HW4 -

Computer Software Engineering(컴퓨터소프트웨어학부)

2020081421

Kim Sung Hyuk(김성혁)

1. [Understanding the task]

- **The Importance of Multi-Intent Detection**
  Multi-intent detection is a significant problem in real-world applications.

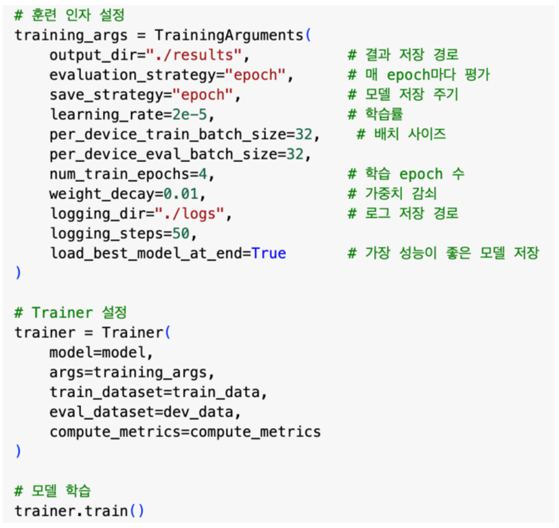
- **Conversational AI**:
  Chatbots need to understand multiple requests in a single sentence.
  Example: Handling "Play some music" and "Check my today’s class" simultaneously.

2. [Model Information]

- **Model: bert-large-uncased**
  - A large model with 340 million parameters.
  - Offers more precise language understanding due to its deeper and wider architecture.
  - Uncased - Does not distinguish between uppercase and lowercase letters.

In this experiment, training our model via A100GPU with 40GB meomory.
This training condition efficiently handles the training of bert-large, which has more parameters than the base BERT model.
And, considering the dataset size and GPU performance, the risk of overfitting is low

3. [Training Tactic]



```python
# 훈련 인자 설정
training_args = TrainingArguments(
    output_dir="./results",             # 결과 저장 경로
    evaluation_strategy="epoch",        # 매 epoch마다 평가
    save_strategy="epoch",              # 모델 저장 주기
    learning_rate=2e-5,                 # 학습률
    per_device_train_batch_size=32,      # 배치 사이즈
    per_device_eval_batch_size=32,
    num_train_epochs=4,                 # 학습 epoch 수
    weight_decay=0.01,                  # 가중치 감쇠
    logging_dir="./logs",               # 로그 저장 경로
    logging_steps=50,
    load_best_model_at_end=True         # 가장 성능이 좋은 모델 저장
)

# Trainer 설정
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_data,
    eval_dataset=dev_data,
    compute_metrics=compute_metrics
)

# 모델 학습
trainer.train()
```

- **Learning Rate:**
  The recommended values for BERT-based models are either 2e-5 or 5e-5.

- **Per Device Train Batch Size:**
  The batch size per GPU during training. Larger batch sizes improve parallel training efficiency but increase memory usage.

- **Number of Training Epochs:**
  The number of times the training process iterates over the entire dataset. If the model's performance is insufficient, additional training epochs can be applied.

- **Weight_decay:**
  Used to regularize model weights to prevent overfitting.

- **load_best_model_at_end:**
  Automatically loads the best-performing checkpoint at the end of training.
  → Helps prevent a drop in performance during the training process

4. [Trial and error]

- Uploading HW.json to Colab

```python
# Upload and load the dataset
uploaded = files.upload()
file_path = "HW4.json"
```

```
파일 선택 HW4.json
• HW4.json(application/json) - 4930413 bytes, last modified: 2024. 12. 18. - 100% done
Saving HW4.json to HW4.json
```

Using the files library from google.colab, the file can be uploaded via files.upload().

After uploading, the file can be accessed and used as if it were a local file.

- Data Preprocessing

The challenge is deciding how to handle and compare multi-label intents during preprocessing.

Since there are multiple intents per sample, the total number of unique intents (147) was determined, and the problem was framed as a multi-label classification task.

```python
# HW4.json 파일 로드 및 고유 레이블 추출
with open("HW4.json", "r") as f:
    raw_data = [json.loads(line) for line in f]

# 고유 레이블 추출
unique_labels = set()
for row in raw_data:
    intents = eval(row['intent'])  # 문자열을 리스트로 변환
    unique_labels.update(intents)

unique_labels = sorted(list(unique_labels))  # 레이블 정렬
num_labels = len(unique_labels)
```

```python
# 데이터를 Dataset 객체로 변환
data = Dataset.from_list(raw_data)

# 레이블 인코딩 함수: 문자열 intent를 이진 벡터로 변환
def encode_labels(batch):
    intents = eval(batch['intent'])  # 문자열을 리스트로 변환
    labels = [1.0 if label in intents else 0.0 for label in unique_labels] # Float type으로 변환
    return {"labels": labels}

# 레이블 인코딩 적용
data = data.map(encode_labels)

# 훈련 및 검증 데이터셋 분리
train_data = data.filter(lambda x: x['split'] == 'train')
dev_data = data.filter(lambda x: x['split'] == 'dev')
```

- **Steps for Preprocessing:**
  1. Convert the JSON file into a list of Python dictionaries
  2. Extract the intent field from each row
  3. Extract unique labels

- **Unique Label Extraction:**
  1. Convert the intent field (string) into a list. Extract the intent field from each row
  2. Add intents to a set to maintain unique labels.
  3. Convert the set into a sorted list to generate unique_labels.

⇨ **In this approach resolved the issue!**

- **WandB-Related Error**

An error message indicated that classifier.weight and classifier.bias were not initialized.

```
# WANDB 오류 메세지 무시
os.environ["WANDB_DISABLED"] = "true"
```

W&B (Weights & Biases) is a tool for visualizing training logs.
This error occurred when attempting to evaluate a pre-trained model without training it first.
To bypass this issue, os.environ["WANDB_DISABLED"] = "true" was used to disable W&B logging.

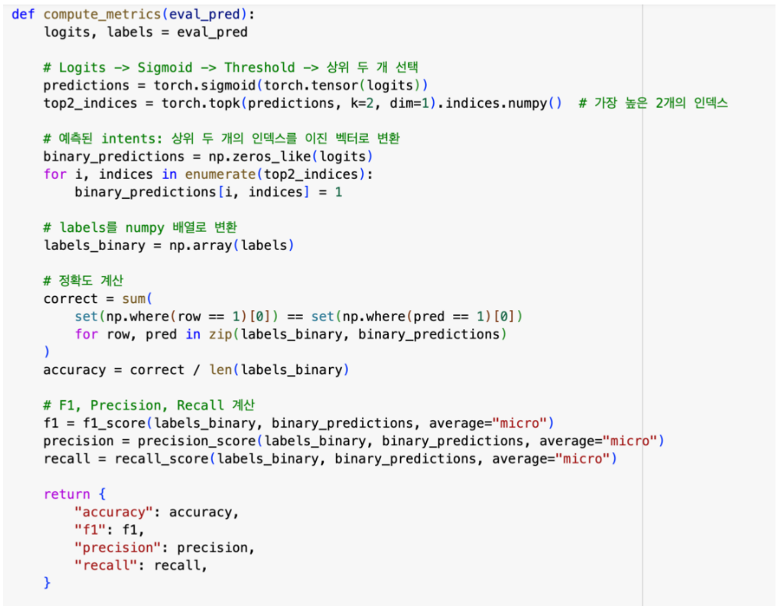
- **Data Type Error**

  Input and target for binary_cross_entropy_with_logits must be of **type float.**

  Therefore, the labels (target data) were explicitly converted to float to resolve the issue.

5. [Resulting Model]

- ### Computation Metrics



```python
def compute_metrics(eval_pred):
    logits, labels = eval_pred

    # Logits -> Sigmoid -> Threshold -> 상위 두 개 선택
    predictions = torch.sigmoid(torch.tensor(logits))
    top2_indices = torch.topk(predictions, k=2, dim=1).indices.numpy()  # 가장 높은 2개의 인덱스

    # 예측된 intents: 상위 두 개의 인덱스를 이진 벡터로 변환
    binary_predictions = np.zeros_like(logits)
    for i, indices in enumerate(top2_indices):
        binary_predictions[i, indices] = 1

    # labels를 numpy 배열로 변환
    labels_binary = np.array(labels)

    # 정확도 계산
    correct = sum(
        set(np.where(row == 1)[0]) == set(np.where(pred == 1)[0])
        for row, pred in zip(labels_binary, binary_predictions)
    )
    accuracy = correct / len(labels_binary)

    # F1, Precision, Recall 계산
    f1 = f1_score(labels_binary, binary_predictions, average="micro")
    precision = precision_score(labels_binary, binary_predictions, average="micro")
    recall = recall_score(labels_binary, binary_predictions, average="micro")

    return {
        "accuracy": accuracy,
        "f1": f1,
        "precision": precision,
        "recall": recall,
    }
```

1. **Accuracy:**

   The proportion of total samples correctly predicted by the model.

2. **Precision:**

   The proportion of predicted positive samples that are actually positive.

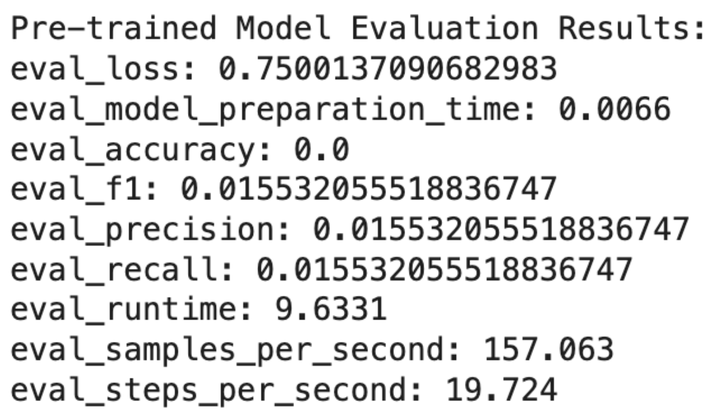
3. **Recall:**

   The proportion of actual positive samples that are correctly predicted by the model.

4. **F1 Score:**

   The harmonic mean of Precision and Recall..

1. Pre-trained model evaluation



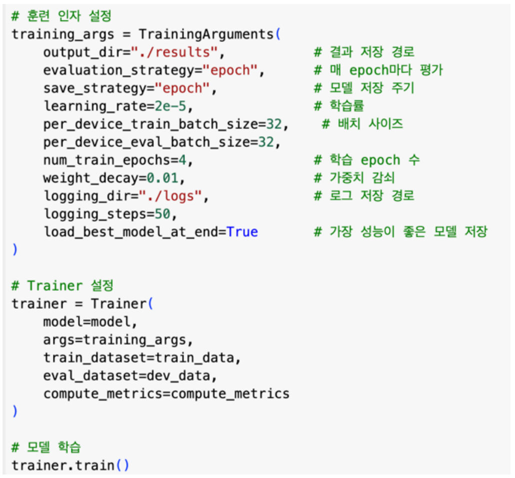
```
Pre-trained Model Evaluation Results:
eval_loss: 0.7500137090682983
eval_model_preparation_time: 0.0066
eval_accuracy: 0.0
eval_f1: 0.015532055518836747
eval_precision: 0.015532055518836747
eval_recall: 0.015532055518836747
eval_runtime: 9.6331
eval_samples_per_second: 157.063
eval_steps_per_second: 19.724
```

- **Analysis result**

As we expected, pre-trained model is not sufficient to use in this task.

We need to Fine-tuned this model!

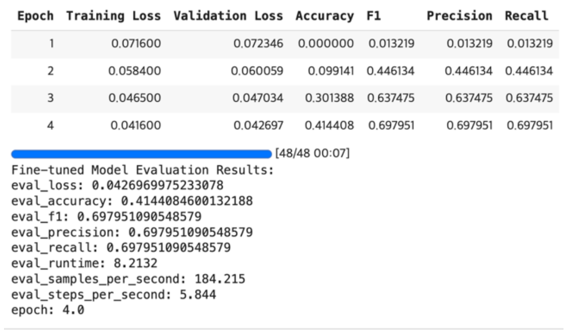
2. Fine-tuned model evaluation after 1st training



```python
# 훈련 인자 설정
training_args = TrainingArguments(
    output_dir="./results",          # 결과 저장 경로
    evaluation_strategy="epoch",     # 매 epoch마다 평가
    save_strategy="epoch",           # 모델 저장 주기
    learning_rate=2e-5,              # 학습률
    per_device_train_batch_size=32,   # 배치 사이즈
    per_device_eval_batch_size=32,
    num_train_epochs=4,              # 학습 epoch 수
    weight_decay=0.01,               # 가중치 감쇠
    logging_dir="./logs",            # 로그 저장 경로
    logging_steps=50,
    load_best_model_at_end=True      # 가장 성능이 좋은 모델 저장
)

# Trainer 설정
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_data,
    eval_dataset=dev_data,
    compute_metrics=compute_metrics
)

# 모델 학습
trainer.train()
```



Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.071600	0.072346	0.000000	0.013219	0.013219	0.013219
2	0.058400	0.060059	0.099141	0.446134	0.446134	0.446134
3	0.046500	0.047034	0.301388	0.637475	0.637475	0.637475
4	0.041600	0.042697	0.414408	0.697951	0.697951	0.697951

```
[48/48 00:07]
Fine-tuned Model Evaluation Results:
eval_loss: 0.0426969975233078
eval_accuracy: 0.4144084600132188
eval_f1: 0.697951090548579
eval_precision: 0.697951090548579
eval_recall: 0.697951090548579
eval_runtime: 8.2132
eval_samples_per_second: 184.215
eval_steps_per_second: 5.844
epoch: 4.0
```

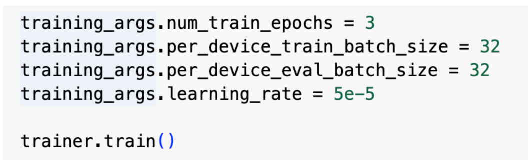
- **Analysis result**

The evaluation scores show significant improvement after the first training step. However, the accuracy remains relatively low, indicating that the model is not yet fully optimized. To enhance performance, **additional training is necessary.**

- **2nd Traning tactic :**

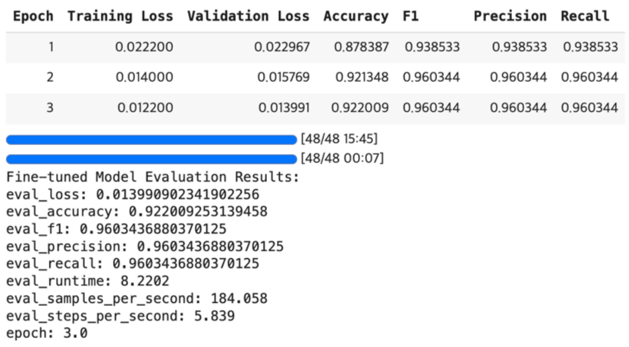
1. Increase the batch size to 32 to better utilize the GPU.
2. Adjust the learning rate to 5e-5 for more stable and efficient training.

3. Fine-tuned model evaluation after 2st training



```python
training_args.num_train_epochs = 3
training_args.per_device_train_batch_size = 32
training_args.per_device_eval_batch_size = 32
training_args.learning_rate = 5e-5

trainer.train()
```



Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.022200	0.022967	0.878387	0.938533	0.938533	0.938533
2	0.014000	0.015769	0.921348	0.960344	0.960344	0.960344
3	0.012200	0.013991	0.922009	0.960344	0.960344	0.960344

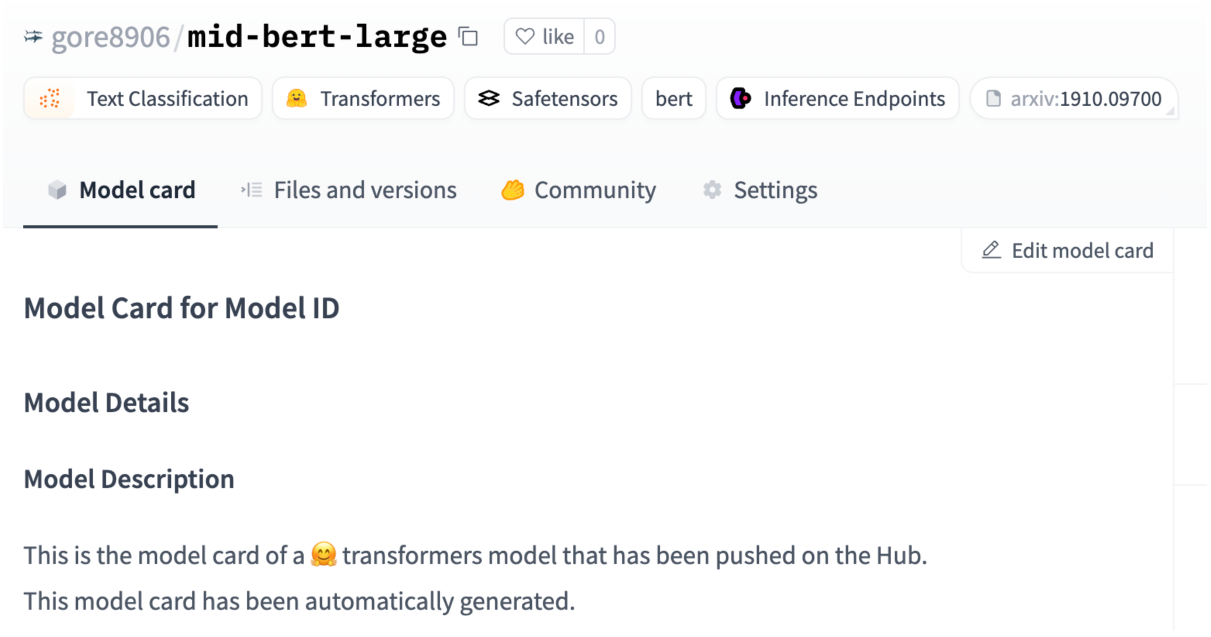
```
[48/48 15:45]
[48/48 00:07]
Fine-tuned Model Evaluation Results:
eval_loss: 0.013990902341902256
eval_accuracy: 0.922009253139458
eval_f1: 0.9603436880370125
eval_precision: 0.9603436880370125
eval_recall: 0.9603436880370125
eval_runtime: 8.2202
eval_samples_per_second: 184.058
eval_steps_per_second: 5.839
epoch: 3.0
```

- **Analysis result**

Result was really interesting.
Validation loss and accuracy improved compared to the first step.
The second training step successfully improved the model's performance, achieving high accuracy, F1 score, precision, and recall.

The model appears to have fully converged, and additional training may not result in significant improvements.

- **Upload model Hugging Face**


gore8906/mid-bert-large

Model Card for Model ID

Model Details

Model Description

This is the model card of a 🤗 transformers model that has been pushed on the Hub. This model card has been automatically generated.

- Model name: mid-bert-large

- **Hugging face Link:**

https://huggingface.co/gore8906/mid-bert-large

[Referrence]

A notebook on how to Finetune BERT for multi-label classification using PyTorch.:

https://colab.research.google.com/github/abhimishra91/transformers-tutorials/blob/master/transformers_multi_label_classification.ipynb

BERT:

https://huggingface.co/docs/transformers/model_doc/bert