Introduction to Artificial Intelligence (Fall 2024)

Homework #2

Team #: _____5_____

Student IDs: 2021313549 / 2024319690 / 2020312668

[Notice]

The assignment consists of two parts: a mandatory problem and a bonus problem.

- Mandatory problem [100/100]

For this project, you will be given two files. One for binary classification and one for multi-class classification. Each task uses a different dataset, and you are required to prepare a report on the aspects provided below.

- Bonus problem [50/50]

The bonus problem is optional. You are required to classify automobile-related data through prompt writing and prompt engineering. Use the provided train.csv file as input for the prompt and evaluate performance using the test.csv file. Code for baseline is given.

[Submission]

Submit both a report and code for the assignment. The report should be converted to a PDF, and the code should be written in Google Colab and submitted as an .ipynb file.

For the bonus problem, ensure that the test data is not included in the prompt. If there is any indication that test data has been used in the training prompt in any form to improve performance, a score of zero will be given.

Name the report as **{Team#}_AI_PA2.pdf** and the code as **{Team#}_AI_PA2.ipynb**. Then, compress both files into a single zip file named **{Team#}_AI_PA2.zip** before submitting.

Points will be deducted if the required format is not followed.

Both Korean and English are allowed.

Any violations of academic integrity such as plagiarism will result in an F grade.

[Inquiry]

Please direct all assignment inquiries only to swe3011_g@g.skku.edu

[Mandatory Problem]

1. Fill out the blank in the code (50pts)

The code for binary classification is fully provided. Based on the binary classification code, fill out the blanks in the multi-classification code. Make sure to carefully refer to the comments provided in the code.

Problem no 1.

Import necessary libraries, load and pad the Reuters dataset with a vocabulary size of 1000 and a max sequence length of 100, then convert labels to categorical format.

```
[ ] ###
    from tensorflow.keras.datasets import reuters
    from tensorflow.keras.preprocessing.sequence import pad_sequences
    from tensorflow.keras.utils import to_categorical

    # limitations
    vocabulary_size = 1000
    maxlen = 100

    # load dataset
    (X_train, y_train), (X_test, y_test) = reuters.load_data(num_words=vocabulary_size, test_split=0.2)

    # sequence padding
    X_train = pad_sequences(X_train, maxlen=maxlen)
    X_test = pad_sequences(X_test, maxlen=maxlen)

    # transform label to categorical form
    y_train = to_categorical(y_train)
    y_test = to_categorical(y_test)
    ###
```

Problem no 2.

Define an LSTM model for multi-class classification, set up early stopping and model checkpoint callbacks, compile the model, and train with validation data.

```
    ###
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Embedding, LSTM, Dense
    from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
    from tensorflow.keras.layers import Dropout
    from tensorflow.keras.layers import Bidirectional
    from tensorflow.keras.optimizers import Adam

    # define model
    model = Sequential()
    model.add(Embedding(vocabulary_size, 128))
    model.add(LSTM(128))
    #model.add(Bidirectional(LSTM(128)))
    #model.add(Dropout(0.5))
    model.add(Dense(46, activation='softmax'))  # 레이블 클래스 수는 46개

    # early stopping and model checkpoint
    es = EarlyStopping(monitor='val_loss', patience=3)
    mc = ModelCheckpoint('best_model.keras', monitor='val_acc', mode='max', save_best_only=True)

    # compile
    optimizer = Adam(learning_rate=0.0045)
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['acc'])

    # learn
    history = model.fit(X_train, y_train, epochs=30, batch_size=64, validation_split=0.2, callbacks=[es, mc])
    ###
```

Problem no 3.

Load the best saved model from file and evaluate its accuracy on the test data.

```
###
from tensorflow.keras.models import load_model

# load best model
best_model = load_model('best_model.keras')

# evaluate with test data
loss, accuracy = best_model.evaluate(X_test, y_test, verbose=0)
print('Test Accuracy: {:.4f}'.format(accuracy))
print('Test Loss: {:.4f}'.format(loss))
###
```

Problem no 4.

Plot training and validation loss over epochs to visualize model performance.

```
[ ] ###
    import matplotlib.pyplot as plt

    # get loss data from train history
    history_dict = history.history
    loss = history_dict['loss']
    val_loss = history_dict['val_loss']

    # epoch range
    epochs = range(1, len(loss) + 1)

    # plot each loss
    plt.plot(epochs, loss, 'bo-', label='train loss')
    plt.plot(epochs, val_loss, 'ro-', label='validation loss')
    plt.title('loss of train and validation')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend()
    plt.show()
    ###
```

2. Capture of your code (15pts)

Write an analysis of the final accuracy and loss values, as well as the methods you used to improve performance.

```
###
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Bidirectional
from tensorflow.keras.optimizers import Adam

# define model
model = Sequential()
model.add(Embedding(vocabulary_size, 128))
model.add(LSTM(128))
#model.add(Bidirectional(LSTM(128)))
#model.add(Dropout(0.5))
model.add(Dense(46, activation='softmax'))  # 레이블 클래스 수는 46개

# early stopping and model checkpoint
es = EarlyStopping(monitor='val_loss', patience=3)
mc = ModelCheckpoint('best_model.keras', monitor='val_acc', mode='max', save_best_only=True)

# compile
optimizer = Adam(learning_rate=0.0045)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['acc'])

# learn
history = model.fit(X_train, y_train, epochs=30, batch_size=64, validation_split=0.2, callbacks=[es, mc])
###
```

```
Epoch 1/30
113/113 ━━━━━━━━━━━━━━━━━━━━ 37s 291ms/step - acc: 0.3802 - loss: 2.5037 - val_acc: 0.5086 - val_loss: 1.9383
Epoch 2/30
113/113 ━━━━━━━━━━━━━━━━━━━━ 41s 287ms/step - acc: 0.5620 - loss: 1.7618 - val_acc: 0.6244 - val_loss: 1.4752
Epoch 3/30
113/113 ━━━━━━━━━━━━━━━━━━━━ 42s 292ms/step - acc: 0.6511 - loss: 1.3974 - val_acc: 0.6706 - val_loss: 1.2861
Epoch 4/30
113/113 ━━━━━━━━━━━━━━━━━━━━ 41s 289ms/step - acc: 0.6945 - loss: 1.1873 - val_acc: 0.7051 - val_loss: 1.1728
Epoch 5/30
113/113 ━━━━━━━━━━━━━━━━━━━━ 41s 293ms/step - acc: 0.7393 - loss: 1.0026 - val_acc: 0.7190 - val_loss: 1.1326
Epoch 6/30
113/113 ━━━━━━━━━━━━━━━━━━━━ 41s 294ms/step - acc: 0.7777 - loss: 0.8585 - val_acc: 0.7262 - val_loss: 1.1166
Epoch 7/30
113/113 ━━━━━━━━━━━━━━━━━━━━ 41s 292ms/step - acc: 0.8047 - loss: 0.7617 - val_acc: 0.7268 - val_loss: 1.1376
Epoch 8/30
113/113 ━━━━━━━━━━━━━━━━━━━━ 41s 294ms/step - acc: 0.8360 - loss: 0.6403 - val_acc: 0.7312 - val_loss: 1.1125
Epoch 9/30
113/113 ━━━━━━━━━━━━━━━━━━━━ 34s 302ms/step - acc: 0.8629 - loss: 0.5457 - val_acc: 0.7474 - val_loss: 1.1173
Epoch 10/30
113/113 ━━━━━━━━━━━━━━━━━━━━ 41s 302ms/step - acc: 0.8718 - loss: 0.4878 - val_acc: 0.7401 - val_loss: 1.1613
Epoch 11/30
113/113 ━━━━━━━━━━━━━━━━━━━━ 42s 316ms/step - acc: 0.9008 - loss: 0.4014 - val_acc: 0.7429 - val_loss: 1.2192
```

Problem no 3.

Load the best saved model from file and evaluate its accuracy on the test data.

```
###
from tensorflow.keras.models import load_model

# load best model
best_model = load_model('best_model.keras')

# evaluate with test data
loss, accuracy = best_model.evaluate(X_test, y_test, verbose=0)
print('Test Accuracy: {:.4f}'.format(accuracy))
print('Test Loss: {:.4f}'.format(loss))
###
```

```
Test Accuracy: 0.7293
Test Loss: 1.1823
```

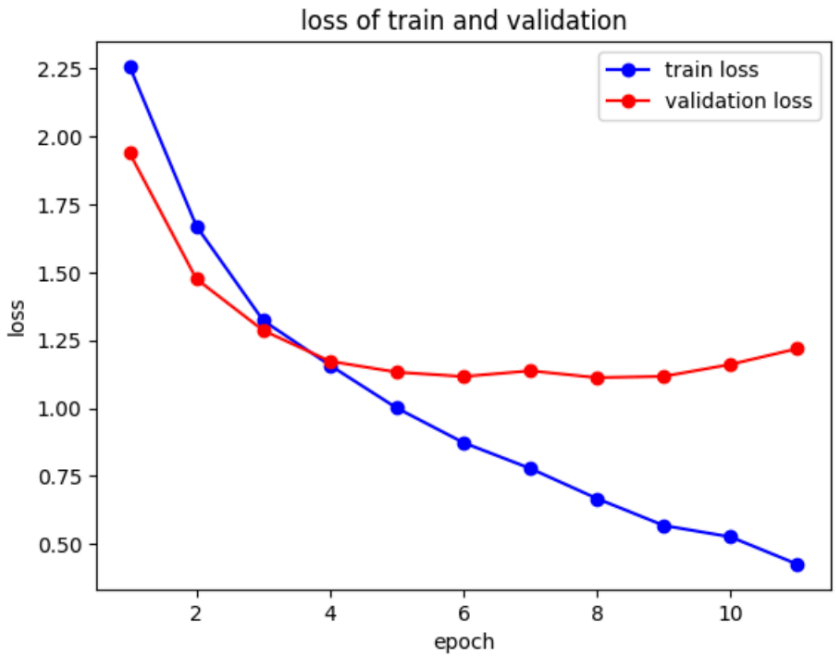
Plot training and validation loss over epochs to visualize model performance.

```
###
import matplotlib.pyplot as plt

# get loss data from train history
history_dict = history.history
loss = history_dict['loss']
val_loss = history_dict['val_loss']

# epoch range
epochs = range(1, len(loss) + 1)

# plot each loss
plt.plot(epochs, loss, 'bo-', label='train loss')
plt.plot(epochs, val_loss, 'ro-', label='validation loss')
plt.title('loss of train and validation')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
###
```



3. Analysis (35pts)

Write an analysis of the final accuracy and loss values, as well as the methods you used to improve performance

Our final accuracy and loss values are 0.7293 & 1.1823. Accuracy means that our model classified 72.93% of the test data accurately. Unlike accuracy, loss value's number itself doesn't seem to have specific meaning, it's just some relative value that we have to minimize.

We made several versions to maximize accuracy by changing parameters. This is our history of changings.

Version 1.0

Problem no 2.

Define an LSTM model for multi-class classification, set up early stopping and model checkpoint callbacks, compile the model, and train with validation data.

```
###
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.layers import Dropout

# 모델 정의
model = Sequential()
model.add(Embedding(1000, 128))
model.add(LSTM(128))
model.add(Dense(46, activation='softmax'))  # 레이블 클래스 수는 46개

# 조기 종료와 모델 체크포인트 콜백 설정
es = EarlyStopping(monitor='val_loss', patience=3)
mc = ModelCheckpoint('best_model.keras', monitor='val_acc', mode='max', save_best_only=True)

# 모델 컴파일
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])

# 모델 학습
history = model.fit(X_train, y_train, epochs=20, batch_size=128, validation_split=0.2, callbacks=[es, mc])
###
```

embedding dimensionality: 128

LSTM units: 128

early stopping patience: 3

epochs: 20

batch size: 128

=======RESULT=======

Problem no 3.

Load the best saved model from file and evaluate its accuracy on the test data.

```
###
from tensorflow.keras.models import load_model

# 가장 성능이 좋은 모델 로드
best_model = load_model('best_model.keras')

# 테스트 데이터에서 모델 평가
loss, accuracy = best_model.evaluate(X_test, y_test, verbose=0)
print('테스트 정확도: {:.4f}'.format(accuracy))
###
```

```
테스트 정확도: 0.6687
```

Interpretation: This is our first model setting. We are going to improve this accuracy. (also the capture of code is the old version, so it seems little bit different from final code)

Version 2.0

changed:

- epoch: 30

- batch size: 64

======**RESULT**======

```
테스트 정확도: 0.7257
```

Interpretation: Increased epoch and decreased batch size significantly improved accuracy. Due to the early stopping, learning epoch stops at 19~21 in average, so changing epoch doesn't affect much but we assumed that it is okay to be large rather than small. Decreased batch size made each epoch to have more steps. The number of steps has increased from 57 to 113.

Version 3.0

changed:

- embedding dimensionality: 256

- LSTM units: 256

======**RESULT**======

```
테스트 정확도: 0.7035
```

Interpretation: We assumed that increased dimensionality and more LSTM units could improve the accuracy. But these changes degraded the accuracy. So, we reject these changes.

Version 4.0

changed:

- embedding dimensionality: 64

- LSTM units: 128

======**RESULT**======

```
Test Accuracy: 0.6919
Test Loss: 1.2686
```

Interpretation: Decreasing dimensionality is not helpful.

Version 4.1

changed:

- embedding dimensionality: 128

- LSTM units: 64

=======RESULT=======

```
Test Accuracy: 0.6901
Test Loss: 1.2811
```

Interpretation: Decreasing LSTM units is not helpful.

Version 4.2

changed:

- embedding dimensionality: 64

- LSTM units: 64

=======RESULT=======

```
Test Accuracy: 0.6919
Test Loss: 1.2663
```

Interpretation: Decreasing both is not helpful too.

Version 5.0

changed:

- Using Bidirectional LSTM (128 units)

```
#model.add(LSTM(128))
model.add(Bidirectional(LSTM(128)))
```

=======RESULT=======

```
테스트 정확도: 0.7133
```

Interpretation: Imported Bidirectional LSTM like ELMo. We expected more precise classification with more context-concerning model. But the result seems not quite improved.

Version 5.1

changed:

- Bidirectional LSTM units: 256

=======RESULT=======

```
테스트 정확도: 0.7017
```

Interpretation: Increasing bidirectional LSTM units is not helpful

Version 5.2

changed:

- Bidirectional LSTM units: 64

=======RESULT=======

```
Test Accuracy: 0.6995
Test Loss: 1.3320
```

Interpretation: Decreasing bidirectional LSTM units is not helpful.

Version 6.0

changed:

- Using Dropout

```
model.add(Dropout(0.5))
```

- Single direction LSTM(128) and 128 dimensions

=======RESULT=======

```
테스트 정확도: 0.6972
```

Interpretation: Dropout seems not effective with single LSTM.

Version 6.1

changed:

- Using Dropout

- Bidirectional LSTM(128) and 128 dimensions

=======RESULT=======

```
테스트 정확도: 0.7137
```

Interpretation: The accuracy has slightly improved. We may assume that dropout is useful with bidirectional LSTM. From here on, every change will be tested with both Version 2.0 and Version 6.1

Version 7.0

changed:

- Batch size: 32

- Single directional LSTM without Dropout (v2.0)

=======RESULT=======

```
테스트 정확도: 0.6911
```

Interpretation: We tried to decrease batch size even more since decreased batch size from 128 to 64 have improved accuracy last time. In single directional LSTM, this change was not helpful.

Version 7.1

changed:

- Batch size: 32

- Bidirectional LSTM with Dropout (v6.1)

=======RESULT=======

```
테스트 정확도: 0.7061
```

Interpretation: In bidirectional LSTM with dropout, this change wasn't helpful either. So, we reject this one.

Version 8.0 - 8.7

changed:

- Adjusting learning rate of optimizer.

```
# compile
optimizer = Adam(learning_rate=0.0045)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['acc'])
```

- Single directional LSTM without Dropout (v2.0)

=======RESULTS======= (so long to put them all here, so summarized them)

Learning rate: 0.0005 => Accuracy: 0.6870

Learning rate: 0.001 => Accuracy: 0.7018

Learning rate: 0.002 => Accuracy: 0.7146

Learning rate: 0.003 => Accuracy: 0.7235

Learning rate: 0.004 => Accuracy: 0.7260

Learning rate: 0.005 => Accuracy: 0.7222

Learning rate: 0.01 => Accuracy: 0.7004

Learning rate: 0.0045 => Accuracy: **0.7293**

Interpretation: We observed each accuracy while adjusting the learning rate, we found that 0.0045 was the optimal learning rate. And interestingly, as the learning rate increased, the epoch took to find the best model was significantly decreased (even to 8-10)

Version 9.0

changed:

- Learning rate: 0.0045

- Bidirectional LSTM

=======RESULT=======

```
Test Accuracy: 0.7280
Test Loss: 1.2368
```

Interpretation: This version is also usable, but it is slightly worse than version 8.7 (single directional LSTM with learning rate 0.0045).

Final model:

```
# define model
model = Sequential()
model.add(Embedding(vocabulary_size, 128))
model.add(LSTM(128))
#model.add(Bidirectional(LSTM(128)))
#model.add(Dropout(0.5))
model.add(Dense(46, activation='softmax'))  # 레이블 클래스 수는 46개

# early stopping and model checkpoint
es = EarlyStopping(monitor='val_loss', patience=3)
mc = ModelCheckpoint('best_model.keras', monitor='val_acc', mode='max', save_best_only=True)

# compile
optimizer = Adam(learning_rate=0.0045)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['acc'])
```

[Bonus Problem]

The guidelines are as follows:

- ✓ Model: GPT-3.5-turbo-0125
- ✓ Prompt Token Limit: A maximum of 16,000 tokens, including both the system and user prompts
- ✓ Model Output Rules: For each of the 40 sample rows in the evaluation data, output only the prediction (0 or 1) per row
- ✓ Temperature: 0.4

1. Problem Definition (10pts)

Define any anticipated issues when running the code in a naïve manner.

<Issues of Original Code>

There are two main issues with the original code.

First, the code doesn't use train.csv. GPT model will classify test.csv's data entries only with their own trained language model. This can also make quite useful results, but we may use train.csv to give the model more specified context to improve F1 score.

Second, the code often generates more or less than 40 lines. This issue is very critical. We assumed that the main cause of the issue is that we are using old legacy model, GPT-3.5-turbo with quite high temperature(0.4). But our assignment is to maximize likelihood in given environment. So, we did many tries to deal with the issue which are described afterwards.

There are also minor issues such as calculating token usage.

Now we aim to deal with described issues.

2. References (10pts)

Include references to any papers consulted, specifying which strategies from theses references were used.

Prompt Engineering (=> instructing step by step)

https://www.aiground.co.kr/mastering-chatgpt-prompt-engineering/

Prompt Engineering (=> several ways to improve GPT's response. e.g. menacing, reward and punishment, avoiding negative sentences, and alignment of instructions)

https://kr.linkedin.com/posts/jocoding_%EC%B1%97gpt%EC%97%90%EA%B2%8C-%ED%8C%81%EC%9D%84-%EC%A4%80%EB%8B%A4%EA%B3%A0-%ED%95%98%EA%B1%B0%EB%82%98-%ED%98%91%EB%B0%95%ED%95%98%EB%A9%B4-%EB%8D%94-%EC%A2%8B%EC%9D%80-%EB%8B%B5%EB%B3%80%EC%9D%84-%EC%A4%80%EB%8B%A4%EB%8A%94-%EC%82%AC%EC%8B%A4-%EC%95%8C%EA%B3%A0-activity-7167405031309209600-Koni

https://www.youtube.com/watch?v=mC2b57u_s0k

Classification model using GPT (=> Referenced the way of classifying)

https://medium.com/@hugmanskj/hands-on-%EA%B1%B0%EB%8C%80%EC%96%B8%EC%96%B4%EB%AA%A8%EB%8D%B8%EC%9D%84-%ED%99%9C%EC%9A%A9%ED%95%9C-%ED%94%84%EB%A1%AC%ED%94%84%ED%8A%B8-%EA%B8%B0%EB%B0%98-%ED%85%8D%EC%8A%A4%ED%8A%B8-%EB%B6%84%EB%A5%98-6e9537243eec

3. Analysis (10pts)

Provide an analysis of the final prompt used, along with a discussion of potential methods to improve performance further.

```
[249] # this prompt generates un-refined response
      hardcode_system_message = f"""
      You are an expert in classifying data entries as automotive-related (1) or not automotive-related (0).

      ###Task
      - Classify each data entry as either "1" (automotive-related) or "0" (not automotive-related).
      - You must output exactly 40 lines, with one prediction (0 or 1) on each line.

      ###Examples
      - Here are some examples of labeled data entries:
      {examples_all}

      ###Instructions
      1. Classify the following 40 data entries labeled from ID: TEST_00 to TEST_39.
      2. Consider the title, notes, and the language of each entry when making the classification.
      3. Automotive-related entries often mention roads, vehicles, transportation systems, or traffic data in the title or notes using their own language.
      4. Non-automotive entries usually describe other general topics like taxation, geography, or unrelated statistics.

      **IMPORTANT**
      - Your output MUST have exactly 40 lines.
      - Respond format:
      TEST_00: 1
      TEST_01: 0
      TEST_02: 0
      TEST_03: 1
      ...
      """
```

This is the final prompt.

```
[240] # All entries from train.csv
      train_df = pd.read_csv('train.csv')
      examples_all = ''
      for idx, row in train_df.iterrows():
          entry_id = row['ID']
          title = str(row['title']).strip()
          notes = str(row['notes']).strip()
          target = row['target']
          content = f"ID: {entry_id}\nTitle: {title}\nNotes: {notes}\nLabel: {target}"
          examples_all += content + '\n\n'
      #print(examples_all)
```

And this is {examples_all}. Using all train.csv data.

```
Accuracy: 0.95
Macro F1 Score: 0.95

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.95      0.95        20
           1       0.95      0.95      0.95        20

    accuracy                           0.95        40
   macro avg       0.95      0.95      0.95        40
weighted avg       0.95      0.95      0.95        40
```

```
Accuracy: 0.9
Macro F1 Score: 0.9

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.90      0.90        20
           1       0.90      0.90      0.90        20

    accuracy                           0.90        40
   macro avg       0.90      0.90      0.90        40
weighted avg       0.90      0.90      0.90        40
```

left: Highest among 10 times run.

right: Lowest among 10 times run.

<Analysis>

Our prompt is built upon numerous trials and errors. As mentioned at reference part, we used several prompt engineering methods. We tried to threaten GPT, and tell them we will prize you if you successfully classify all the data entries but they didn't actually lead to better result. No matter how much we threatened and rewarded the model to print exactly 40 results, the model would often print 39 or 41 results.

We assumed that the problem is occurring because of the absence of visible 'tag' for each result. So we built a new prompt based on our best-by-far prompt, to make result with format of "TEST_XX: 0 or 1". With this kind of prompt, GPT always output exact 40 results. Then

we parse the output format and get the specific part we want.

<Potential methods to improve performance further>

Current model actually shows pretty good performance, but of course it's not perfect. So these are the potential methods we could try to further enhance the results.

1. Provide more train data to the model.
2. Lower the temperature of the model (currently 0.4)
3. Upgrade the LLM model from GPT-3.5-turbo to GPT-4o or GPT-o1-preview
4. Some more prompt engineering techniques such as re-aligning sentence sequence, or tell GPT that we will prize or punish according to the accuracy of results.

4. Score (20pts)

This part does not need to be included in the report. Rankings will be assigned based on F1 score values, and points will be distributed by range.