Deep Learning trong khoa học dữ liệu
Bài tập thực hành 6 - DS201.M11.2

Đỗ Trọng Hợp, Lưu Thanh Sơn, Nguyễn Thành Luân
Sinh viên: Phạm Đức Thể - 19522253
Ngôn ngữ lập trình: Python

Thứ 4, ngày 08 tháng 12 năm 2021

Bài tập: ỨNG DỤNG MẠNG NEURAL HỒI QUY

Bài toán phân tích cảm xúc (sentiment analysis)

Setup

1. Download công cụ tách từ vncorenlp

```
!pip install vncorenlp

!mkdir -p vncorenlp/models/wordsegmenter
!wget https://raw.githubusercontent.com/vncorenlp/VnCoreNLP/master/
VnCoreNLP-1.1.1.jar
!wget https://raw.githubusercontent.com/vncorenlp/VnCoreNLP/master/
models/wordsegmenter/vi-vocab
!wget https://raw.githubusercontent.com/vncorenlp/VnCoreNLP/master/
models/wordsegmenter/wordsegmenter.rdr
!mv VnCoreNLP-1.1.1.jar vncorenlp/
!mv vi-vocab vncorenlp/models/wordsegmenter/
!mv wordsegmenter.rdr vncorenlp/models/wordsegmenter/
```

2. Download bộ Word embedding PhoW2V

```
!gdown --id 102-AO-5zKTi8_NHCNUaKurUxHnljy6wt
!unzip word2vec_vi_words_100dims.zip
```

3. Import các thư viện cần thiết

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sn
from vncorenlp import VnCoreNLP
from keras.models import Model, Input, Sequential
from keras.initializers import Constant
from tensorflow.keras.optimizers import Adam
from keras.losses import CategoricalCrossentropy
from sklearn.metrics import accuracy_score, recall_score, f1_score
from sklearn.metrics import precision_score, confusion_matrix
from keras.layers import Flatten, Dense, Dropout
from keras.utils.vis_utils import plot_model
from tensorflow.keras import Model, Input, optimizers
from tensorflow.keras.layers import LSTM, Embedding, Bidirectional
from tensorflow.keras.layers import TimeDistributed, SpatialDropout1D
from keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from keras.callbacks import EarlyStopping
sn.set() # Set theme
```

4. Chuẩn bị dữ liệu

```
path = '/content/drive/MyDrive/Dataset/UIT-VSFC'
# Train set
X_train = pd.read_csv(path + '/train/sents.txt',  sep='\n',
                      header=None, index_col=None)
X_train = X_train.iloc[:, 0]
y_train = pd.read_csv(path + '/train/sentiments.txt',  sep='\n',
                      header=None, index_col=None)
# Dev set
X_dev = pd.read_csv(path + '/dev/sents.txt', sep='\n',
                    header=None, index_col=None)
X_dev = X_dev.iloc[:, 0]
y_dev = pd.read_csv(path + '/dev/sentiments.txt', sep='\n',
                    header=None, index_col=None)
# Test set
X_test = pd.read_csv(path + '/test/sents.txt',  sep='\n',
                     header=None, index_col=None)
X_test = X_test.iloc[:, 0]
y_test = pd.read_csv(path + '/test/sentiments.txt',  sep='\n',
                     header=None, index_col=None)
# Flatten labels
y_train = y_train.values.flatten()
y_dev = y_dev.values.flatten()
y_test = y_test.values.flatten()
```

5. Load bộ word-embedding lên

```
EMBEDDING = '/content/word2vec_vi_words_100dims.txt'
EMBEDING_DIM = 100
MAX_FEATURE = 10000

word_dict = []
embeddings_index = {}
embedding_dim = EMBEDING_DIM
max_feature = MAX_FEATURE

f = open(EMBEDDING)
for line in f:
  values = line.split(' ')
  word = values[0]
  word_dict.append(word)
  try:
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
  except Exception as e:
    pass
f.close()

print('Embedding data loaded')
# Output: Embedding data loaded
```

6. Xây dựng bộ từ vựng

```
words = word_dict
num_words = len(words)

# Dictionary word:index pair
# word is key and its value is corresponding index
word2index = {w : i + 2 for i, w in enumerate(words)}
word2index["UNK"] = 1
word2index["PAD"] = 0

# Dictionary lable:index pair
idx2word = {i: w for w, i in word2index.items()}
```

7. Khởi tạo embedding matrix và load trọng số của pre-trained embedding vào

```
embedding_matrix = np.zeros((num_words, embedding_dim))

# for each word in out tokenizer lets try to find that work in our w2v model
for word, i in word2index.items():
  if i > max_feature:
    continue
  embedding_vector = embeddings_index.get(word)
  if embedding_vector is not None:
    # add that words vector to the matrix
    embedding_matrix[i] = embedding_vector
  else:
    # doesn't exist, assign a random vector
    embedding_matrix[i] = np.random.randn(embedding_dim)
```

8. Sử dụng bộ tách từ vncorenlp

```
vncorenlp = VnCoreNLP("vncorenlp/VnCoreNLP-1.1.1.jar",
                      annotators="wseg", max_heap_size='-Xmx500m')

def custom_tokenizer(text_data, tokenizer=True):
  if tokenizer:
    return " ".join(vncorenlp.tokenize(str(text_data))[0])
  return text_data
```

9. Tiến hành xây dựng hàm mã hoá cho các câu bình luận

```
MAX_LEN = 100
NUM_LABEL = 3
TOKENIZER = True

def encoding(X, y):
  sentences = []
  for t in X:
    sentences.append(custom_tokenizer(t,tokenizer=TOKENIZER))

  X = []
  for s in sentences:
    sent = []
    for w in s.split():
      try:
        w = w.lower()
        sent.append(word2index[w])
      except:
        sent.append(word2index["UNK"])
    X.append(sent)

  X = pad_sequences(maxlen = MAX_LEN, sequences = X,
                    padding = "post", value = word2index["PAD"])
  y = to_categorical(y, num_classes=NUM_LABEL)

  return (X,y)
```

10. Tiến hành mã hoá dữ liệu ban đầu

```
X_train_encoded,y_train_encoded=encoding(X_train.values.flatten(),y_train)
X_dev_encoded,y_dev_encoded=encoding(X_dev.values.flatten(),y_dev)
X_test_encoded,y_test_encoded=encoding(X_test.values.flatten(),y_test)
```

Bài 1: Hiện thực lại mô hình BiLSTM ở mục 1. Sử dụng các độ đo precision, recall và F1-score để cho biết khả năng dự đoán của mô hình. Đánh giá khả năng dự đoán của mô hình trên từng nhãn (confusion matrix).

1. Xây dựng mô hình

```
model_1 = Sequential()
model_1.add(Input(shape=(MAX_LEN, ), dtype="float64"))
model_1.add(Embedding(input_dim=num_words,
                      output_dim=embedding_dim,
                      embeddings_initializer=Constant(embedding_matrix),
                      input_length=MAX_LEN,
                      trainable=False))
model_1.add(Bidirectional(LSTM(200,return_sequences=False)))
model_1.add(Dense(3, activation='softmax'))

model_1.summary()
```

2. Huấn luyện mô hình

```
# Compile model
optimizer = Adam(learning_rate=0.01)
loss = CategoricalCrossentropy()
callback = EarlyStopping(monitor='val_loss', patience=3)
model_1.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

# Training model
history_1 = model_1.fit(X_train_encoded, y_train_encoded),
                        validation_data=(X_dev_encoded, y_train_encoded),
                        batch_size=128, epochs=30, callbacks=[callback])
```

3. Visualization Loss & Accuracy training

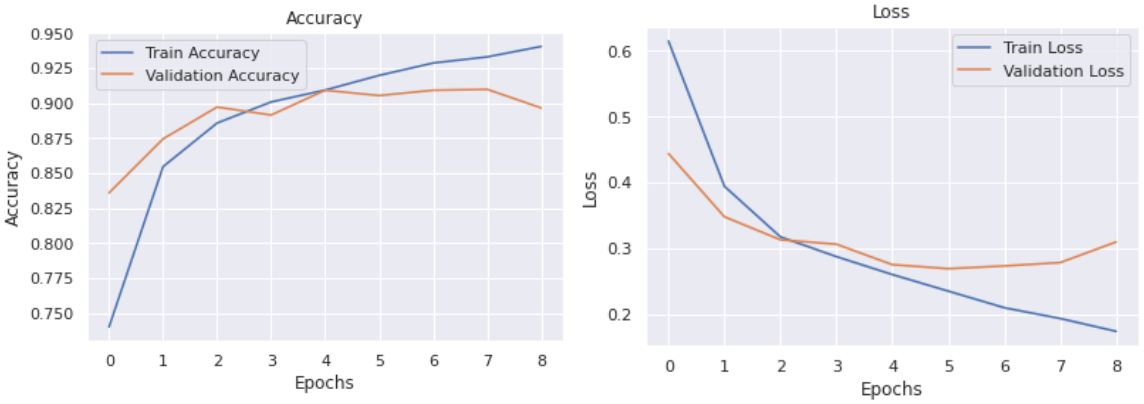
```
plt.plot(history_1.history['accuracy'], label = 'Train Accuracy')
plt.plot(history_1.history['val_accuracy'], label = 'Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

– Output: Hình 1a

```
plt.plot(history_1.history['loss'], label = 'Train Loss')
plt.plot(history_1.history['val_loss'], label = 'Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

– Output: Hình 1b

* **Nhận xét**:
  – Accuracy trên tập train và dev tăng nhanh. Accuracy trên cả 2 tập đều rất cao (accuracy train $\sim 94.0\%$, accuracy dev $\sim 89.6\%$).
  – Loss trên cả 2 tập train và dev đều giảm rất nhanh. Tuy nhiên loss ở tập dev lại tăng ở những epoch cuối (bắt đầu có hiện tượng underfiting).



(a) Accuracy

(b) Loss

Hình 1: Accuracy và Loss của model bài 1

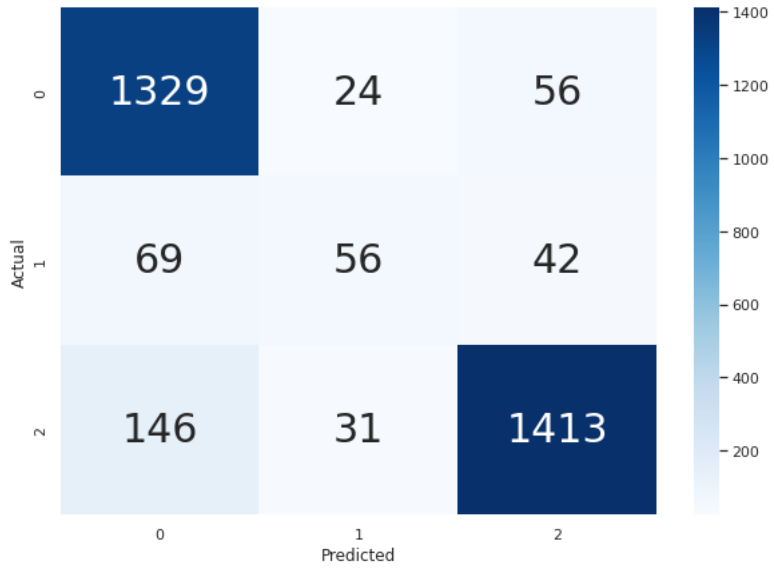
4. Predict

```
y_pred = model_1.predict(X_test_encoded)
y_pred = np.argmax(y_pred, axis=-1)

precision = round(precision_score(y_test, y_pred, average='macro')*100, 2)
print('Precision macro test set: {}%'.format(precision))
# Output: Precision macro test set: 76.68%
precision = round(precision_score(y_test, y_pred, average='micro')*100, 2)
print('Precision micro test set: {}%'.format(precision))
# Output: Precision micro test set: 88.38%
recall = round(recall_score(y_test, y_pred, average='macro')*100, 2)
print('Recall macro test set: {}%'.format(recall))
# Output: Recall macro test set: 72.24%
recall = round(recall_score(y_test, y_pred, average='micro')*100, 2)
print('Recall micro test set: {}%'.format(recall))
# Output: Recall micro test set: 88.38%
f1 = round(f1_score(y_test, y_pred, average='macro')*100, 2)
print('F1-score macro test set: {}%'.format(f1))
# Output: F1-score macro test set: 73.81%
f1 = round(f1_score(y_test, y_pred, average='micro')*100, 2)
print('F1-score micro test set: {}%'.format(f1))
# Output: F1-score micro test set: 88.38%
```

* **Nhận xét**:
  - Với các độ đo khác nhau model cho kết quả khá cao (Precision macro= 76.68%, Precision micro= 88.38%, Recall macro = 72.24%, Recall micro = 88.38%, F1-score macro = 73.81%, F1-score micro = 88.38%).
  - Có thể kết luận mô hình có khả năng dự đoán tốt.

```
# Visualization confusion matrix of test set
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10,7))
sn.heatmap(cm, annot=True, cmap = 'Blues', fmt='g',
           cbar=True, annot_kws={"size": 30})
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

– Output: Hình 2



Hình 2: Confusion Matrix

* **Nhận xét**:
  - Model dự đoán tốt ở các nhãn 0 và 2 tương ứng ứng negative và positive, dự đoán không tốt ở nhãn 1 tức là neutral. Nguyên nhân có thể là do bộ dữ liệu bị mất cân bằng giữa các nhãn (nhãn negative và positive chiếm hầu hết số lượng nhãn của tập dữ liệu còn nhãn neutral chiếm rất ít) nên model học các nhãn neutral không được tốt.
  - Model dự đoán nhầm nhãn 0 nhiều nhất vào nhãn 2, dự đoán nhầm nhãn 1 nhiều nhất vào nhãn 0, dự đoán nhầm nhãn 2 nhiều nhất vào nhãn 0.

Bài 2: Giống bài 1, nhưng sử dụng 2 lớp BiLSTM kết hợp nhau. So sánh 2 mô hình với nhau và cho biết kết quả.

1. Xây dựng mô hình

```
model_2 = Sequential()
model_2.add(Input(shape=(MAX_LEN, ), dtype="float64"))
model_2.add(Embedding(input_dim=num_words,
                      output_dim=embedding_dim,
                      embeddings_initializer=Constant(embedding_matrix),
                      input_length=MAX_LEN,
                      trainable=False))
model_2.add(Bidirectional(LSTM(200,return_sequences=True)))
model_2.add(Bidirectional(LSTM(100,return_sequences=False)))
model_2.add(Dense(3, activation='softmax'))

model_2.summary()
```

2. Huấn luyện mô hình

```
# Compile model
optimizer = Adam(learning_rate=0.01)
loss = CategoricalCrossentropy()
callback = EarlyStopping(monitor='val_loss', patience=3)
model_2.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

# Training model
history_2 = model_2.fit(X_train_encoded, y_train_encoded,
                        validation_data=(X_dev_encoded, y_dev_encoded),
                        batch_size=128, epochs=30, callbacks=[callback])
```

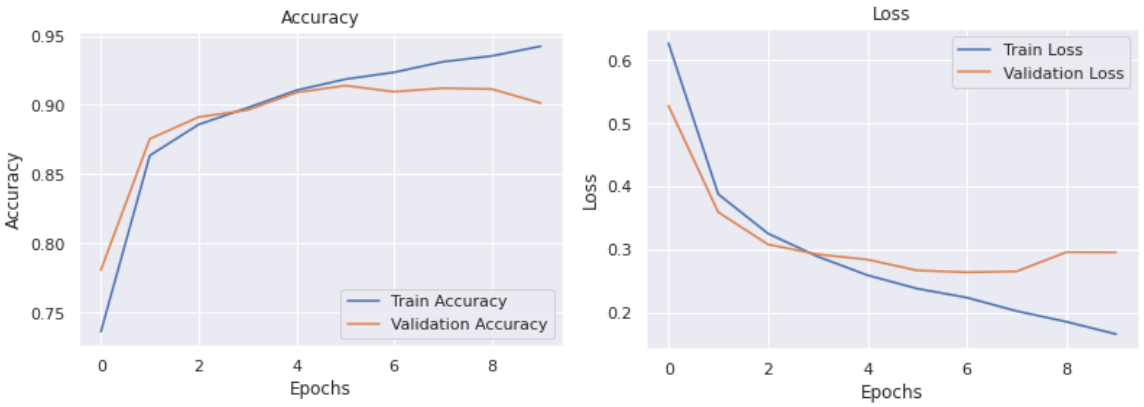
3. Visualization Loss & Accuracy training

```
plt.plot(history_2.history['accuracy'], label = 'Train Accuracy')
plt.plot(history_2.history['val_accuracy'], label = 'Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

– Output: Hình 3a

```
plt.plot(history_2.history['loss'], label = 'Train Loss')
plt.plot(history_2.history['val_loss'], label = 'Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

– Output: Hình 3b



(a) Accuracy

(b) Loss

Hình 3: Accuracy và Loss của model bài 2

* **Nhận xét**:

– Accuracy trên tập train và dev tăng nhanh. Accuracy trên cả 2 tập đều rất cao (accuracy train $\sim$ 94.3%, accuracy dev $\sim$ 90.0%).

– Loss trên cả 2 tập train và dev đều giảm rất nhanh. Tuy nhiên loss ở tập dev lại tăng ở những epoch cuối (bắt đầu có hiện tượng underfiting).

4. Predict

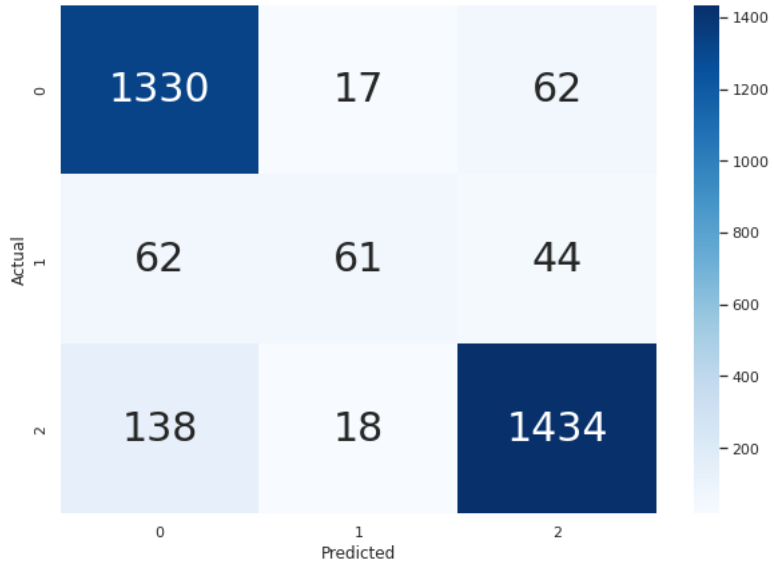
```
y_pred = model_2.predict(X_test_encoded)
y_pred = np.argmax(y_pred, axis=-1)

precision = round(precision_score(y_test, y_pred, average='macro')*100, 2)
print('Precision macro test set: {}%'.format(precision))
# Output: Precision macro test set: 81.2%
precision = round(precision_score(y_test, y_pred, average='micro')*100, 2)
print('Precision micro test set: {}%'.format(precision))
# Output: Precision micro test set: 89.23%
recall = round(recall_score(y_test, y_pred, average='macro')*100, 2)
print('Recall macro test set: {}%'.format(recall))
# Output: Recall macro test set: 73.7%
recall = round(recall_score(y_test, y_pred, average='micro')*100, 2)
print('Recall micro test set: {}%'.format(recall))
# Output: Recall micro test set: 89.23%
```

```
f1 = round(f1_score(y_test, y_pred, average='macro')*100, 2)
print('F1-score macro test set: {}%'.format(f1))
# Output: F1-score macro test set: 76.17%
f1 = round(f1_score(y_test, y_pred, average='micro')*100, 2)
print('F1-score micro test set: {}%'.format(f1))
# Output: F1-score micro test set: 89.23%
```

* **Nhận xét**:
  - Với các độ đo khác nhau model cho kết quả khá cao (Precision macro= 81.2%, Precision micro= 89.23%, Recall macro = 73.7%, Recall micro = 89.23%, F1-score macro = 76.17%, F1-score micro = 89.23%). Từ kết quả của các độ đo khác nhau thì có thể thấy model bài 2 tốt hơn model bài 1.
  - Có thể kết luận mô hình có khả năng dự đoán tốt.

```
# Visualization confusion matrix of test set
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10,7))
sn.heatmap(cm, annot=True, cmap = 'Blues', fmt='g',
           cbar=True, annot_kws={"size": 30})
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

– Output: Hình 4



Hình 4: Confusion Matrix

* **Nhận xét**:
  - Model dự đoán tốt ở các nhãn 0 và 2 tương ứng ứng negative và positive, dự đoán không tốt ở nhãn 1 tức là neutral. Nguyên nhân có thể là do bộ dữ liệu bị mất cân bằng giữa các nhãn (nhãn negative và positive chiếm hầu hết số lượng nhãn của tập dữ liệu còn nhãn neutral chiếm rất ít) nên model học các nhãn neutral không được tốt.
  - Model dự đoán nhầm nhãn 0 nhiều nhất vào nhãn 2, dự đoán nhầm nhãn 1 nhiều nhất vào nhãn 0, dự đoán nhầm nhãn 2 nhiều nhất vào nhãn 0.
  - Tuy model bài 2 có tốt hơn model bài 1, những kết quả thể hiện thông qua ma trận nhầm lẫn (confusion matrix) không được cải thiện đáng kể.

Bài toán nhận diện thực thể có tên (named entities recognition)

Setup

1. Cài đặt thư viện để đánh giá mô hình

```
!pip install seqeval
```

2. Download bộ dữ liệu

```
!git clone https://github.com/VinAIResearch/PhoNER_COVID19
```

3. Đọc dữ liệu

```
def load_data(filename, encoding='utf-8'):
  sents, labels = [], []
  words, tags = [], []

  with open(filename, encoding=encoding) as f:
    for line in f:
      try:
        line = line.strip()
        if line:
          word, tag = line.split(" ")
          words.append(word)
          tags.append(tag)
        else:
          sents.append(words)
          labels.append(tags)
          words, tags = [], []
      except Exception as e:
        pass
  return sents, labels

train_set_word=load_data('/content/PhoNER_COVID19/data/word/train_word.conll')
dev_set_word=load_data('/content/PhoNER_COVID19/data/word/dev_word.conll')
test_set_word=load_data('/content/PhoNER_COVID19/data/word/test_word.conll')
```

4. Load bộ word-embedding lên

```
EMBEDDING = '/content/word2vec_vi_words_100dims.txt'
EMBEDING_DIM = 100
MAX_FEATURE = 10000
MAX_LEN = 100

word_dict = []
embeddings_index_word = {}
embedding_dim = EMBEDING_DIM
max_feature = MAX_FEATURE

f = open(EMBEDDING)
for line in f:
  values = line.split(' ')
  word = values[0]
  word_dict.append(word)
  try:
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index_word[word] = coefs
  except Exception as e:
    pass

f.close()

print('Embedding data loaded')
# Output: Embedding data loaded
```

5. Lấy ra bộ nhãn (tags)

```
tags = list(set(np.concatenate((
    np.concatenate(train_set_word[1]),
    np.concatenate(dev_set_word[1])))))
```

6. Xây dựng word_index và tag_index

```
words = word_dict
num_words = len(words)
words2index = {w:i for i,w in enumerate(words)}
tags2index = {t:i for i,t in enumerate(tags)}
```

7. Khởi tạo embedding matrix và load trọng số của pre-trained embedding vào

```
embedding_matrix_word = np.zeros((num_words, embedding_dim))
# for each word in out tokenizer lets try to find that work in our w2v model
for word, i in words2index.items():
  if i > max_feature:
    continue
  embedding_word_vector = embeddings_index_word.get(word)
  if embedding_word_vector is not None:
    # add that words vector to the matrix
    embedding_matrix_word[i] = embedding_word_vector
  else:
    # doesn't exist, assign a random vector
    embedding_matrix_word[i] = np.random.randn(embedding_dim)
```

8. Mã hoá dữ liệu

```
def encoding(data):
  X = [[words2index.get(w, 0) for w in t] for t in data[0]]
  X = pad_sequences(maxlen = MAX_LEN, sequences = X,
                    padding = "post", value = 0)

  y = [[tags2index[w] for w in s] for s in data[1]]
  y = pad_sequences(maxlen=MAX_LEN, sequences=y,
                    padding="post", value=tags2index["O"])

  return X, y

X_train_word, y_train_word = encoding(train_set_word)
X_dev_word, y_dev_word = encoding(dev_set_word)
X_test_word, y_test_word = encoding(test_set_word)
y_train_word = to_categorical(y_train_word, num_classes=len(tags))
y_dev_word = to_categorical(y_dev_word, num_classes=len(tags))
```

Bài 3: Hiện thực lại mô hình BiLSTM cho tác vụ NER trên bộ dữ liệu COVID-19 NER ở mức độ từ (word-level).

1. Xây dựng mô hình

```
model_3 = Sequential()
model_3.add(Input(shape=(MAX_LEN,)))
model_3.add(Embedding(input_dim = num_words,
                      output_dim = EMBEDING_DIM,
                      embeddings_initializer=Constant(embedding_matrix_word),
                      input_length = MAX_LEN,
                      mask_zero = True,
                      trainable=False))
model_3.add(Bidirectional(LSTM(units=200,return_sequences=True,
                               recurrent_dropout=0.2, dropout=0.2)))
model_3.add(TimeDistributed(Dense(len(tags), activation="softmax")))

model_3.summary()
```

2. Huấn luyện mô hình

```
# Compile model
optimizer = Adam(learning_rate=0.01)
loss = CategoricalCrossentropy()
callback = EarlyStopping(monitor='val_loss', patience=3)
model_3.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

# Training model
history_3 = model_3.fit(X_train_word, y_train_word,
                        validation_data=(X_dev_word, y_dev_word),
                        batch_size=128, epochs=30,
                        callbacks=[callback])
```

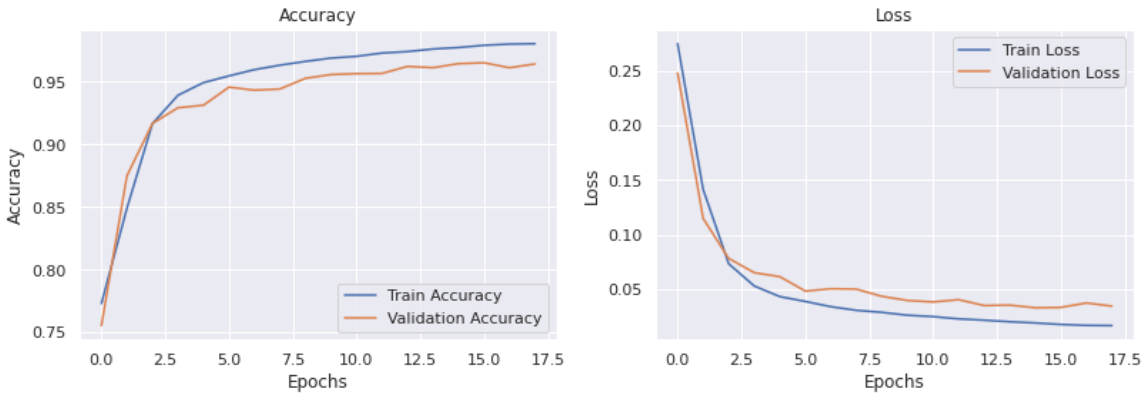
3. Visualization Loss & Accuracy training

```
plt.plot(history_3.history['accuracy'], label = 'Train Accuracy')
plt.plot(history_3.history['val_accuracy'], label = 'Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

– Output: Hình 5a

```
plt.plot(history_3.history['loss'], label = 'Train Loss')
plt.plot(history_3.history['val_loss'], label = 'Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

– Output: Hình 5b



(a) Accuracy (b) Loss

Hình 5: Accuracy và Loss của model bài 3

* **Nhận xét**:

– Accuracy trên tập train và dev tăng nhanh. Accuracy trên cả 2 tập đều rất cao (accuracy train $\sim 98.0\%$, accuracy dev $\sim 96.4\%$).

– Loss trên cả 2 tập train và dev đều giảm rất nhanh.

4. Predict

```
y_true = []
y_pred = []
for i in range(y_test_word.shape[0]):
  y_true.append(y_test_word[i])
  p = model_3.predict(np.array([X_test_word[i]]))
  p = np.argmax(p, axis=-1)
  y_pred.extend(p)

Y_true = []
Y_pred = []
for i, j in zip(range(len(y_true)), range(len(y_pred))):
  true = []
  pred = []
  for k in range(len(y_true[i])):
    true.append(tags[y_true[i][k]])
    pred.append(tags[y_pred[j][k]])
  Y_true.append(true)
  Y_pred.append(pred)

from seqeval.metrics import f1_score
from seqeval.metrics import accuracy_score
accuracy = round(accuracy_score(Y_true, Y_pred)*100,2)
print('Accuracy test set: {}%'.format(accuracy))
# Output: Accuracy test set: 98.67%
f1 = round(f1_score(Y_true, Y_pred, average='micro')*100,2)
print('F1-score micro test set: {}%'.format(f1))
# Output: F1-score micro test set: 85.06%
f1 = round(f1_score(Y_true, Y_pred, average='macro')*100,2)
print('F1-score macro test set: {}%'.format(f1))
# Output: F1-score macro test set: 75.43%
```

* **Nhận xét**:
  - Accuracy của mô hình rất cao (accuracy = 98.67%). Tuy nhiên với bài toán NER thì độ đo accuracy không thể hiện được chính xác hiệu suất của mô hình. Vì vậy, chúng ta sẽ đánh giá hiệu suất của mô hình dựa trên độ đo F1-score. Mô hình cho kết quả trên độ đo F1-score rất khá tốt (F1-score micro = 85.06%, F1-score macro = 75.43%)
  - Có thể kết luận mô hình có khả năng dự đoán tốt.

```
from seqeval.metrics import classification_report as seqeval_cs

print(seqeval_cs(Y_true, Y_pred, digits =4))
```

– Output: Hình 6

	precision	recall	f1-score	support
AGE	0.9662	0.8970	0.9303	573
DATE	0.9627	0.9697	0.9662	1650
GENDER	0.9846	0.8477	0.9110	453
JOB	0.5606	0.4302	0.4868	172
LOCATION	0.8467	0.8509	0.8488	4434
NAME	0.8207	0.4748	0.6016	318
ORGANIZATION	0.6283	0.7017	0.6630	771
PATIENT_ID	0.9432	0.9488	0.9460	1994
SYMPTOM_AND_DISEASE	0.7592	0.7245	0.7414	1136
TRANSPORTATION	0.8788	0.3005	0.4479	193
micro avg	0.8626	0.8389	0.8506	11694
macro avg	0.8351	0.7146	0.7543	11694
weighted avg	0.8634	0.8389	0.8470	11694

Hình 6: Precision, Recall và F1-score của từng nhãn thực thể

* **Nhận xét**:
  - Mô hình dự đoán tốt ở hầu hết các thực thể, tuy nhiên có 2 thực thể mô hình dự đoán chưa được tốt là JOB và TRANSPORTATION. Nguyên nhân có thể là do số lượng các thực thể này chiếm tỉ lệ ít trong bộ dữ liệu nên mô hình học chưa được tốt các thực thể này.

Bài 4: Tương tự bài 3, nhưng sử dụng bộ dữ liệu COVID-19 NER trên mức độ tiếng (syllabus). So sánh kết quả giữa 2 loại mức độ từ và mức độ tiếng.

1. Chuẩn bị dữ liệu

(a) Download bộ Syllables embedding PhoW2V

```
!gdown --id 1yj4-YPJ33Nck2GbKshxgM28PeT-mqyC6
!unzip word2vec_vi_syllables_100dims.zip
```

(b) Load bộ syllables-embedding lên

```
EMBEDDING = '/content/word2vec_vi_syllables_100dims.txt'
EMBEDING_DIM = 100
MAX_FEATURE = 10000
MAX_LEN = 100

syllable_dict = []
embeddings_index_syllable = {}
embedding_dim = EMBEDING_DIM
max_feature = MAX_FEATURE

f = open(EMBEDDING)
for line in f:
  values = line.split(' ')
  syllable = values[0]
  syllable_dict.append(syllable)
  try:
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index_syllable[syllable] = coefs
  except Exception as e:
    pass

f.close()

print('Embedding data loaded')
# Output: Embedding data loaded
```

(c) Đọc dữ liệu

```
def load_datas(filename, encoding='utf-8'):
  sents, labels = [], []
  syllables, tags = [], []
  with open(filename, encoding=encoding) as f:
    for line in f:
      try:
        line = line.strip()
        if line:
          syllable, tag = line.split(" ")
          syllables.append(syllable)
          tags.append(tag)
        else:
          sents.append(syllables)
          labels.append(tags)
          syllables, tags = [], []
      except Exception as e:
        pass
  return sents, labels
```

```
train_set_syllable=load_data('/content/PhoNER_COVID19/data/syllable/
                              train_syllable.conll')
dev_set_syllable=load_data('/content/PhoNER_COVID19/data/syllable/
                            dev_syllable.conll')
test_set_syllable=load_data('/content/PhoNER_COVID19/data/syllable/
                             test_syllable.conll')
```

(d) Lấy ra bộ nhãn (tags)

```
tags_syllable = list(set(np.concatenate((
    np.concatenate(train_set_syllable[1]),
    np.concatenate(dev_set_syllable[1])))))
```

(e) Xây dựng syllable_index và tag_syllable_index

```
syllables = syllable_dict
num_syllables = len(syllables)

syllable2index = {w:i for i,w in enumerate(syllables)}
tags_syllable2index = {t:i for i,t in enumerate(tags_syllable)}
```

(f) Khởi tạo embedding matrix và load trọng số của pre-trained embedding vào

```
embedding_matrix_syllable = np.zeros((num_syllables, embedding_dim))

# for each syllable in out tokenizer lets try to find that work in our
# s2v model
for syllable, i in syllable2index.items():
  if i > max_feature:
    continue
  embedding_syllable_vector = embeddings_index_syllable.get(syllable)
  if embedding_syllable_vector is not None:
    # add that syllables vector to the matrix
    embedding_matrix_syllable[i] = embedding_syllable_vector
  else:
    # doesn't exist, assign a random vector
    embedding_matrix_syllable[i] = np.random.randn(embedding_dim)
```

(g) Mã hoá dữ liệu

```
def encoding(data):
  X = [[syllable2index.get(w, 0) for w in t] for t in data[0]]
  X = pad_sequences(maxlen = MAX_LEN, sequences = X,
                    padding = "post", value = 0)

  y = [[tags_syllable2index[w] for w in s] for s in data[1]]
  y = pad_sequences(maxlen=MAX_LEN, sequences=y,
                    padding="post", value=tags_syllable2index["O"])

  return X, y

X_train_syllable, y_train_syllable = encoding(train_set_syllable)
X_dev_syllable, y_dev_syllable = encoding(dev_set_syllable)
X_test_syllable, y_test_syllable = encoding(test_set_syllable)


y_train_syllable = to_categorical(y_train_syllable,
                                  num_classes=len(tags_syllable))
y_dev_syllable = to_categorical(y_dev_syllable,
                                num_classes=len(tags_syllable))
```

2. Xây dựng mô hình

```
model_4 = Sequential()
model_4.add(Input(shape=(MAX_LEN,)))
model_4.add(Embedding(input_dim = num_syllables, output_dim = EMBEDDING_DIM,
                      embeddings_initializer=Constant(embedding_matrix_syllable),
                      input_length = MAX_LEN, mask_zero = True, trainable=False))
model_4.add(Bidirectional(LSTM(units=200,return_sequences=True,
                               recurrent_dropout=0.2, dropout=0.2)))
model_4.add(TimeDistributed(Dense(len(tags_syllable), activation="softmax")))
model_4.summary()
```

3. Huấn luyện mô hình

```
# Compile model
optimizer = Adam(learning_rate=0.01)
loss = CategoricalCrossentropy()
callback = EarlyStopping(monitor='val_loss', patience=3)
model_4.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
# Training model
history_4 = model_4.fit(X_train_syllable, y_train_syllable,
                        validation_data=(X_dev_syllable, y_dev_syllable),
                        batch_size=128, epochs=30, callbacks=[callback])
```

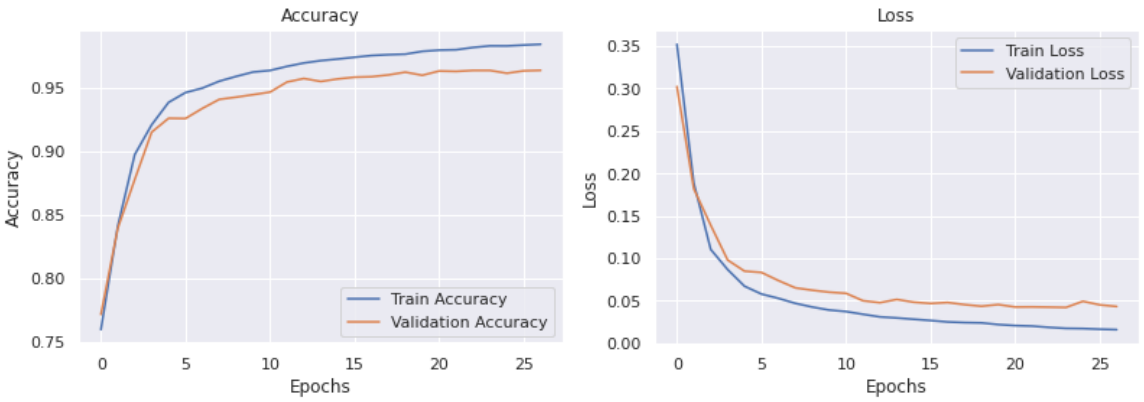
4. Visualization Loss & Accuracy training

```
plt.plot(history_4.history['accuracy'], label = 'Train Accuracy')
plt.plot(history_4.history['val_accuracy'], label = 'Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

– Output: Hình 7a

```
plt.plot(history_4.history['loss'], label = 'Train Loss')
plt.plot(history_4.history['val_loss'], label = 'Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

– Output: Hình 7b



(a) Accuracy (b) Loss

Hình 7: Accuracy và Loss của model bài 4

* **Nhận xét**:
  - Accuracy trên tập train và dev tăng nhanh. Accuracy trên cả 2 tập đều rất cao (accuracy train $\sim$ 98.5%, accuracy dev $\sim$ 96.4%).
  - Loss trên cả 2 tập train và dev đều giảm rất nhanh.

5. Predict

```
y_true = []
y_pred = []
for i in range(y_test_syllable.shape[0]):
  y_true.append(y_test_syllable[i])
  p = model_4.predict(np.array([X_test_syllable[i]]))
  p = np.argmax(p, axis=-1)
  y_pred.extend(p)

Y_true = []
Y_pred = []

for i, j in zip(range(len(y_true)), range(len(y_pred))):
  true = []
  pred = []
  for k in range(len(y_true[i])):
    true.append(tags_syllable[y_true[i][k]])
    pred.append(tags_syllable[y_pred[j][k]])
  Y_true.append(true)
  Y_pred.append(pred)

from seqeval.metrics import f1_score
from seqeval.metrics import accuracy_score
accuracy = round(accuracy_score(Y_true, Y_pred)*100,2)
print('Accuracy test set: {}%'.format(accuracy))
# Output: Accuracy test set: 98.44%
f1 = round(f1_score(Y_true, Y_pred, average='micro')*100,2)
print('F1-score micro test set: {}%'.format(f1))
# Output: F1-score micro test set: 84.61%
f1 = round(f1_score(Y_true, Y_pred, average='macro')*100,2)
print('F1-score macro test set: {}%'.format(f1))
# Output: F1-score macro test set: 75.86%
```

* **Nhận xét**:
  - Accuracy của mô hình rất cao (accuracy = 98.44%). Tuy nhiên với bài toán NER thì độ đo accuracy không thể hiện được chính xác hiệu suất của mô hình. Vì vậy, chúng ta sẽ đánh giá hiệu suất của mô hình dựa trên độ đo F1-score. Mô hình cho kết quả trên độ đo F1-score rất khá tốt (F1-score micro = 84.61%, F1-score macro = 75.86%)
  - Có thể kết luận mô hình có khả năng dự đoán tốt.
  - Mô hình sử dụng bộ dữ liệu COVID-19 NER trên mức độ tiếng (syllabus) cho kết quả F1-macro cao hơn mô hình sử dụng bộ dữ liệu COVID-19 NER ở mức độ từ (word-level) nhưng lại cho kết quả F1-micro thấp hơn. Tuy có sự chênh lệch, nhưng sự chênh lệch này không đáng kể (<0.5%). Không thể khẳng định được mô hình nào tốt hơn mô hình nào.

```
from seqeval.metrics import classification_report as seqeval_cs

print(seqeval_cs(Y_true, Y_pred, digits =4))
```

– Output: Hình 8

	precision	recall	f1-score	support
AGE	0.9339	0.9421	0.9380	570
DATE	0.9513	0.9739	0.9625	1645
GENDER	0.9252	0.9087	0.9169	449
JOB	0.5128	0.4651	0.4878	172
LOCATION	0.8265	0.8648	0.8452	4429
NAME	0.7055	0.6478	0.6754	318
ORGANIZATION	0.5607	0.7022	0.6236	769
PATIENT_ID	0.9560	0.9627	0.9593	1985
SYMPTOM_AND_DISEASE	0.7187	0.7218	0.7202	1136
TRANSPORTATION	0.7949	0.3212	0.4576	193
micro avg	0.8356	0.8568	0.8461	11666
macro avg	0.7885	0.7510	0.7586	11666
weighted avg	0.8387	0.8568	0.8454	11666

Hình 8: Precision, Recall và F1-score của từng nhãn thực thể

* **Nhận xét**:
  - Tương tự như mô hình sử dụng bộ dữ liệu COVID-19 NER ở mức độ từ (word-level) của bài 3, mô hình dự đoán tốt ở hầu hết các thực thể, tuy nhiên có 2 thực thể mô hình dự đoán chưa được tốt là JOB và TRANSPORTATION. Nguyên nhân có thể là do số lượng các thực thể này chiếm tỉ lệ ít trong bộ dữ liệu nên mô hình học chưa được tốt các thực thể này.

Bài 5: Sử dụng 2 lớp BiLSTM kết hợp nhau trên bộ dữ liệu COVID-19 NER ở mức độ từ (word).

1. Xây dựng mô hình

```
model_5 = Sequential()
model_5.add(Input(shape=(MAX_LEN,)))
model_5.add(Embedding(input_dim = num_words, output_dim = EMBEDING_DIM,
                      embeddings_initializer=Constant(embedding_matrix_word),
                      input_length = MAX_LEN, mask_zero = True, trainable=False))
model_5.add(Bidirectional(LSTM(units=200,return_sequences=True,
                               recurrent_dropout=0.2, dropout=0.2)))
model_5.add(Bidirectional(LSTM(units=100,return_sequences=True,
                               recurrent_dropout=0.2, dropout=0.2)))
model_5.add(TimeDistributed(Dense(len(tags), activation="softmax")))
model_5.summary()
```

2. Huấn luyện mô hình

```
# Compile model
optimizer = Adam(learning_rate=0.01)
loss = CategoricalCrossentropy()
callback = EarlyStopping(monitor='val_loss', patience=3)
model_5.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
# Training model
history_5 = model_5.fit(X_train_word, y_train_word,
                        validation_data=(X_dev_word, y_dev_word),
                        batch_size=128, epochs=30, callbacks=[callback])
```

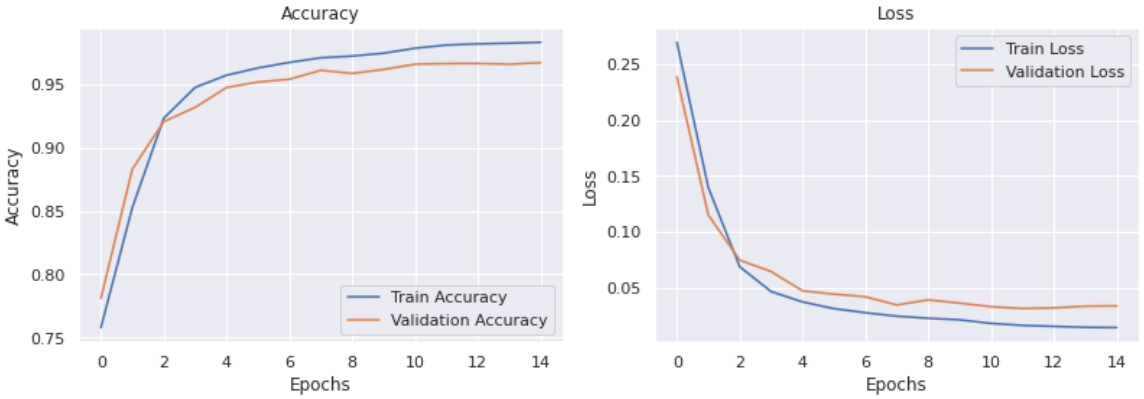
3. Visualization Loss & Accuracy training

```
plt.plot(history_5.history['accuracy'], label = 'Train Accuracy')
plt.plot(history_5.history['val_accuracy'], label = 'Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

– Output: Hình 9a

```
plt.plot(history_5.history['loss'], label = 'Train Loss')
plt.plot(history_5.history['val_loss'], label = 'Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

– Output: Hình 9b



(a) Accuracy (b) Loss

Hình 9: Accuracy và Loss của model bài 5

* **Nhận xét**:

– Accuracy trên tập train và dev tăng nhanh. Accuracy trên cả 2 tập đều rất cao (accuracy train $\sim 98.3\%$, accuracy dev $\sim 96.7\%$).

– Loss trên cả 2 tập train và dev đều giảm rất nhanh.

4. Predict

```
y_true = []
y_pred = []
for i in range(y_test_word.shape[0]):
  y_true.append(y_test_word[i])
  p = model_5.predict(np.array([X_test_word[i]]))
  p = np.argmax(p, axis=-1)
  y_pred.extend(p)

Y_true = []
Y_pred = []
for i, j in zip(range(len(y_true)), range(len(y_pred))):
  true = []
  pred = []
  for k in range(len(y_true[i])):
    true.append(tags[y_true[i][k]])
    pred.append(tags[y_pred[j][k]])
  Y_true.append(true)
  Y_pred.append(pred)

from seqeval.metrics import f1_score
from seqeval.metrics import accuracy_score
accuracy = round(accuracy_score(Y_true, Y_pred)*100,2)
print('Accuracy test set: {}%'.format(accuracy))
# Output: Accuracy test set: 98.79%
f1 = round(f1_score(Y_true, Y_pred, average='micro')*100,2)
print('F1-score micro test set: {}%'.format(f1))
# Output: F1-score micro test set: 86.11%
f1 = round(f1_score(Y_true, Y_pred, average='macro')*100,2)
```

```
print('F1-score macro test set: {}%'.format(f1))
# Output: F1-score macro test set: 76.12%
```

* **Nhận xét**:
  - Accuracy của mô hình rất cao (accuracy = 98.79%). Tuy nhiên với bài toán NER thì độ đo accuracy không thể hiện được chính xác hiệu suất của mô hình. Vì vậy, chúng ta sẽ đánh giá hiệu suất của mô hình dựa trên độ đo F1-score. Mô hình cho kết quả trên độ đo F1-score rất khá tốt (F1-score micro = 86.11%, F1-score macro = 76.12%).
  - Có thể kết luận mô hình có khả năng dự đoán tốt, tốt hơn đôi chút so với mô hình ở bài 3 chỉ sử dụng 1 lớp BiLSTM.

```
from seqeval.metrics import classification_report as seqeval_cs

print(seqeval_cs(Y_true, Y_pred, digits =4))
```

– Output: Hình 10

	precision	recall	f1-score	support
AGE	0.9067	0.9668	0.9358	573
DATE	0.9626	0.9673	0.9649	1650
GENDER	0.9071	0.9272	0.9170	453
JOB	0.5263	0.5233	0.5248	172
LOCATION	0.8547	0.8931	0.8735	4434
NAME	0.7989	0.4497	0.5755	318
ORGANIZATION	0.6605	0.6913	0.6755	771
PATIENT_ID	0.9059	0.9604	0.9323	1994
SYMPTOM_AND_DISEASE	0.7728	0.7394	0.7557	1136
TRANSPORTATION	0.9077	0.3057	0.4574	193
micro avg	0.8577	0.8645	0.8611	11694
macro avg	0.8203	0.7424	0.7612	11694
weighted avg	0.8570	0.8645	0.8566	11694

Hình 10: Precision, Recall và F1-score của từng nhãn thực thể

* **Nhận xét**:
  - Mô hình dự đoán tốt ở hầu hết các thực thể, tuy nhiên có 1 thực thể mô hình dự đoán chưa được tốt là TRANSPORTATION (cải thiện được kết quả so với bài 3 là tăng được độ chính xác F1-score của thực thể JOB từ 48.68% lên 52.48%). Nguyên nhân có thể là do số lượng thực thể này chiếm tỉ lệ ít trong bộ dữ liệu nên mô hình học chưa được tốt thực thể này.