BÀI THỰC HÀNH 6: ỨNG DỤNG MẠNG NEURAL HỒI QUY

Hướng dẫn nộp bài: Nộp file jupyter notebook (đuôi là .jpynb), đặt tên là MSSV_BaiThucHanh6.jpynb + file PDF báo cáo trả lời các câu hỏi: MSSV_BaiThucHanh6.pdf
Nộp qua course, giảng viên sẽ tạo submission sau mỗi buổi học.

1. Bài toán phân tích cảm xúc (sentiment analysis).

Mô tả bài toán: Trong giáo dục, các tổ chức giáo dục sẽ thu thập các phản hồi của người học sau một khoá học hoặc môn học để làm căn cứ nâng cao chất lượng giảng dạy và đào tạo. Một trong các căn cứ đó là mức độ phản hồi tích cực hay tiêu cực của người học.
Bộ dữ liệu: **UIT-VSFC** [1].
Số lượng dữ liệu: khoảng hơn 16,000 câu.
Số lượng nhãn: 3 nhãn gồm tích cực (*positive*), tiêu cực (*negative*) và trung tính (*neural*).
Link:
https://drive.google.com/drive/folders/1xclbjHHK58zk2X6iqbvMPS2rcy9y9E0X

Đọc dữ liệu: dữ liệu được tổ chức ở dạng các file txt (đọc thêm file README để rõ hơn về cấu trúc và thông tin trong bộ dữ liệu).
Word embedding: Sử dụng các bộ **word embedding đã được huấn luyện sẵn** (pre-trained) thay vì phải đi xây dựng lại lớp Embedding dựa trên dữ liệu. Việc này sẽ giúp tiết kiệm thời gian huấn luyện và tăng khả năng nhận diện ngữ nghĩa trong câu do các bộ word embedding được huấn luyện sẵn đã được xây dựng dựa trên các tập dữ liệu rất lớn trước đó, nên khả năng nhận diện ngữ nghĩa sẽ rất cao và đa dạng.
Trong bài thực hành này, chúng ta sử dụng bộ PhoW2V của tác giả Nguyen và các đồng sự [2]. Link tải ở đây: https://github.com/datquocnguyen/PhoW2V.

Bộ word embedding này có 2 phiên bản: 100 chiều và 300 chiều, được huấn luyện trên mức độ ***từ*** và mức độ ***tiếng***. Ta sẽ dùng bộ 100 chiều được huấn luyện trên mức độ từ.
Load bộ word-embedding lên:

```
EMBEDDING = 'drive/My
Drive/DL/embedding/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')
```

Xây dựng bộ từ vựng để chuyển từ thành số (**word_index**). Xem lại bài thực hành 4 để nhớ lại khái niệm word_index

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

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

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

Để sử dụng mã hoá ở mức độ từ, ta cần phải tách các từ trong 1 câu tiếng việc ra thành một danh sách những từ (word). Tác vụ này được gọi là tách từ (word segmentation). Các thư viện hay phần mềm dùng để tách từ được gọi là bộ tách từ (word tokenizer).

VD: câu (a) "học sinh học sinh học" sau khi tách từ sẽ là: (a') "học_sinh học sinh_học". Khi tách chuỗi (a') trên bằng khoảng trắng, ta được danh sách gồm các từ: "học_sinh", "học", "sinh_học".

Trong bài thực hành này, ta sử dụng bộ tách từ *vncorenlp* của tác giả Vu các các đồng sự [3].

Để sử dụng, cần cài đặt các thư viện bằng lệnh pip như sau:

```
pip install vncorenlp
```

Cài đặt và cấu hình thêm các thư viện phụ trợ:

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

Khai báo thư viện và sử dụng bộ tách từ ***vncorenlp*** - hàm `custom_tokenizer`:

```
from vncorenlp import 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
```

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

```
from keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
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(word_to_index[w])
            except:
                sent.append(word_to_index["UNK"])
        X.append(sent)

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

    return (X,y)
```

Lúc này, các câu bình luận được mã hoá bằng word_index từ bộ word_embedding đã được xây dựng sẵn thay vì phải tự xây dựng trên tập từ vựng.

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)
```

Xây dựng mô hình: INPUT -> EMBEDDING (pre-trained) -> BiLSTM -> Dense(3).

```
model = Sequential()
model.add(Input(shape=(MAX_LEN, ), dtype="float64"))
model.add(Embedding(input_dim=num_words,
                    output_dim=embedding_dim,

embeddings_initializer=Constant(embedding_matrix),
                    input_length=MAX_LEN,
                    trainable=False))
```

```
model.add(Bidirectional(LSTM(200,
return_sequences=False)))
model.add(Dense(3, activation='softmax'))

optimizer = Adam(learning_rate=0.01)
loss = CategoricalCrossentropy()

model.compile(optimizer=optimizer, loss=loss,
metrics=['accuracy'])
```

Tiến hành huấn luyện mô hình và đánh giá kết quả nhé.

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

Mô tả: Trong các đoạn text, người ta có nhu cầu trích xuất ra các thông tin về các thực thể có tên như: tên người, tên tổ chức, tên địa danh. Đối với đại dịch COVID, các như cầu trích xuất ra các thông tin về người bệnh, độ tuổi, giới tính, triệu chứng, ... là rất quan trọng, góp phần vào công tác phòng chống dịch bệnh COVID trong công đồng. Bài toán rút trích các thực thể liên quan đến COVID 19 như trên được gọi là bài toán *nhận diện thực thể có tên (named entity recognition - NER).*

Bộ dữ liệu: **PhoNER COVID 19** [4]

Số lượng dữ liệu: 10K câu.

Bộ dữ liệu này có 2 mức độ: tiếng (syllables) và từ (word).

Bộ nhãn (tag):

+ PATIENT_ID: mã bệnh nhân.
+ PERSON_NAME: tên bệnh nhân hoặc người liên hệ với bệnh nhân.
+ AGE: tuổi bệnh nhân hoặc người liên hệ với bệnh nhân.
+ GENDER: giới tính bệnh nhân hoặc người liên hệ với bệnh nhân.
+ OCCUPUTATION: nghề nghiệp bệnh nhân.
+ ORGANIZATION: tổ chức liên quan đến bệnh nhân.
+ SYMTOM&DISEASE: triệu chứng của bệnh nhân, hoặc bệnh nền trước đó.
+ TRANSPORTATION: phương tiện di chuyển của bệnh nhân.
+ DATE: Ngày xuất hiện trong văn bản.

Chúng ta vẫn sử dụng bộ word embedding đã huấn luyện sẵn PhoW2V [2] trên mức độ từ cho bộ dữ liệu này. Do bộ dữ liệu này đã chia mức độ sẵn nên ta không cần tách từ.

Đọc dữ liệu: Bộ dữ liệu này được lưu trữ theo cấu trúc của bộ CoNLL 2003 [5]

```
def load_data_and_labels(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 = load_data_and_labels(train)
dev_set = load_data_and_labels(dev)
test_set = load_data_and_labels(test)
```

Load bộ word embedding lên: như mục 1.

Lấy ra bộ nhãn (tags):

```
import numpy as np

tags = list(set(np.concatenate((
    np.concatenate(train_set[1]),
    np.concatenate(dev_set[1])))
))
```

Xây dựng word_index và tag_index.

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

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, y_train = encoding(train_set)
X_dev, y_dev = encoding(dev_set)
X_test, y_test = encoding(test_set)

y_train = to_categorical(y_train, num_classes=len(tags))
y_dev = to_categorical(y_dev, num_classes=len(tags))
```

Xây dựng mô hình: INPUT -> EMBEDDING (pre-trained) -> BiLSTM (512) -> TimeDistributed (Dense).

```
model = Sequential()
model.add(Input(shape=(MAX_LEN,)))
model.add(Embedding(input_dim = len(words) + 2,
                    output_dim = EMBED_DIM,
                    input_length = MAX_LEN,
                    mask_zero = True,
                    trainable=False))
model.add(Bidirectional(LSTM(units=512,
                return_sequences=True,
                recurrent_dropout=0.2, dropout=0.2)))
model.add(TimeDistributed(Dense(len(tags),
activation="softmax")))

optimizer = Adam(learning_rate=0.01)
loss = CategoricalCrossentropy()

model.compile(optimizer=optimizer, loss=loss,
metrics=['accuracy'])
```

Độ đo đánh giá: tỉ lệ dự đoán đúng tên thực thể trên tổng số câu.

```
import numpy as np
from sklearn.metrics import accuracy_score
acc = []

for i in range(0, len(y_test)):
    acc.append(accuracy_score(y_test[i], y_pred[i]))

np.mean(acc)
```

BÀI TẬP:

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).

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.

Bài 3: Hiện thực lại mô hình BiLSTM cho tác vụ NER trên bộ dữ liệu COVID-19 NER.

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.

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

TÀI LIỆU THAM KHẢO

[1] K. V. Nguyen, V. D. Nguyen, P. X. V. Nguyen, T. T. H. Truong and N. L. Nguyen, "UIT-VSFC: Vietnamese Students' Feedback Corpus for Sentiment Analysis.

[2] Nguyen, A. T., Dao, M. H., & Nguyen, D. Q. (2020). A pilot study of text-to-SQL semantic parsing for Vietnamese.

[3] Vu, T., Nguyen, D. Q., Nguyen, D. Q., Dras, M., & Johnson, M. (2018). VnCoreNLP: A Vietnamese natural language processing toolkit.

[4] Truong, T. H., Dao, M. H., & Nguyen, D. Q. (2021). COVID-19 Named Entity Recognition for Vietnamese.

[5] Sang, E. F., & De Meulder, F. (2003). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition.