BÀI THỰC HÀNH 4: MẠNG NEURAL HỒI QUY

Hướng dẫn nộp bài: Nộp file jupyter notebook (đuôi là .jpynb), đặt tên là MSSV_BaiThucHanh3.jpynb + file PDF báo cáo trả lời các câu hỏi: MSSV_BaiThucHanh3.pdf

Nộp qua course, giảng viên sẽ tạo submission sau mỗi buổi học.

1. Bộ dữ liệu.

Sentiment analysis là một kỹ thuật nhằm xác định và phân tích được cảm xúc của người dùng dựa vào một đoạn hoặc là một câu văn bản.

Bộ dữ liệu IMDB Movies Reviews được xây dựng phục vụ cho bài toán phân tích cảm xúc của người dùng đối với các bộ phim trên IMDB dựa vào các bình luận (review) của họ. Có 2 nhãn cảm xúc chính trong bộ dữ liệu: tích cực (positive) - được ký hiệu là 1 và tiêu cực (negative) - được ký hiệu là 0.

Bộ dữ liệu này được hỗ trợ sẵn trong keras. Cách load dữ liệu như sau:

```
from keras.datasets import imdb

(training_data, training_targets), (testing_data,
testing_targets) = imdb.load_data(num_words=10000)
```

training data: gồm 25,000 câu bình luận đã được mã hoá theo thứ tự từ điển.

testing data: gồm 25,000 câu bình luận đã được mã hoá theo thứ tự từ điển.

Các bạn có thể in ra một câu dữ liệu để xem thử bằng lệnh: `training_data[0]`. Giá trị trả về là một danh sách các số thực. Như vậy, bộ dữ liệu đã được mã hoá sẵn.

Ví dụ: Giả sử tập từ vựng của một bộ dữ liệu có dạng sau:

{ "#": 0
"this": 1,
"is": 2,
"A": 3,
"B": 4,
"C": 5 }

Một câu văn bản: "this is C" sẽ được mã hoá thành vector như sau: [1, 2, 5].

Để chuyển câu dữ liệu `training_data[0]` về dạng văn bản, cần phải có tập từ vựng với các chỉ số đi kèm, được gọi là word_index.

Lấy word index của bộ dữ liệu như sau:

index = imdb.get_word_index()

Ta được kết quả:

```
{'fawn': 34701,
 'tsukino': 52006,
 'nunnery': 52007,
 'sonja': 16816,
 'vani': 63951,
 'woods': 1408,
 ......
}
```

Như vậy, dựa trên word index ta có thể chuyển câu dữ liệu về dạng văn bản như sau:

```
inverted_word_index = dict((i, word) for (word, i) in
word_index.items())
decoded_sequence = " ".join(inverted_word_index[i] for i
in training_data[0])
```

Hãy xem thử kết quả của câu bình luận sau khi đã decode nhé.

Padding chiều dài của sequence: để đảm bảo các sequence đều có độ dài như nhau.

Ở ví dụ này chọn **độ dài sequence là 200**.

```
from keras.preprocessing.sequence import pad_sequences

maxlen = 200
X_train = pad_sequences(training_data, maxlen=maxlen)
X_test = pad_sequences(testing_data, maxlen=maxlen)
```

Xử lý cho nhãn y_train và y_test:

```
from tensorflow.keras.utils import to_categorical

y_train = to_categorical(training_targets,num_classes = 2)
y_test = testing_targets
```

2. Xây dựng mô hình.

SimpleRNN: một lớp (layer) gồm các units kết nối với nhau theo phương pháp Fully connected. Các tham số quan trọng gồm:

+ *units*: số lượng units trong layer.

+ *activation*: hàm kích hoạt.

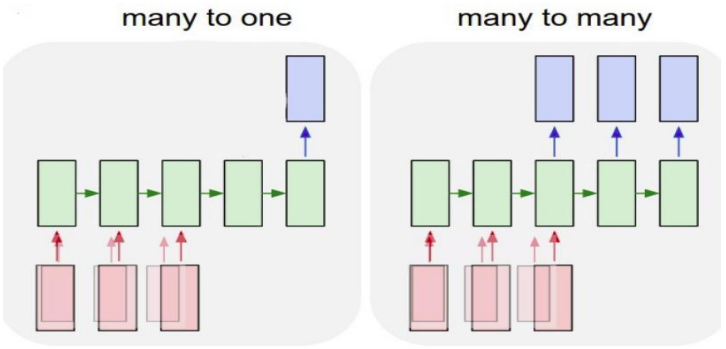
+ *dropout*: sử dụng dropount hay không.

+ *return_sequences*: trả về output là một sequence (đối với Many-to-many) hoặc là một giá trị (Many-to-one).

+ *recurrent_initializer, kernel_initializer và bias_initializer*: khởi tạo giá trị.

+ *recurrent_regularizer, kernel_regularizer*: chuẩn hoá mạng neural.

Xem thêm: https://keras.io/api/layers/recurrent_layers/simple_rnn/



Build mạng neural hồi quy đơn giản gồm lớp với 200 units (ứng với độ dài sequence).

Ghi chú: SimpleRNN cần đầu vào có shape như sau: [batch, timesteps, feature]. Ý nghĩa như sau:

+ ***batch***: số lượng sample - điểm dữ liệu (đã chia theo kích thước mỗi batch. Có thể xem lại batch and mini-batch training để hiểu rõ hơn).

+ ***timesteps***: Số lượng từ, ký tự trong câu. Ở đây timesteps chính là độ dài chuỗi.

+ ***features***: số lượng features đưa vào mỗi steps trong sequence. features sẽ bằng với ***chiều (dimension) của word embedding***.

Mô hình đơn giản ban đầu sẽ dùng features với kích thước là 1. Do đó, ta sẽ expand kích thước của dữ liệu ban đầu bằng cách dùng np.expand_dims trong thư viện numpy (xem lại bài thực hành 3).

```
import numpy as np

X_train_new = np.expand_dims(X_train, axis=2)
X_test_new = np.expand_dims(X_test, axis=2)
```

Xây dựng mạng neural RNN đơn giản:

```
model = Sequential()
```

```
model.add(Input(shape=(None, 1), dtype="float64"))
model.add(SimpleRNN(200, return_sequences=False,
return_state=False, activation='relu'))
model.add(Dense(2, activation='sigmoid'))

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

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

Hàm loss: BinaryCrossEntropy

Optimizer: Adam

Kết quả build model:

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_20 (SimpleRNN)    (None, 200)               40400
_________________________________________________________________
dense_11 (Dense)             (None, 2)                 402
=================================================================
Total params: 40,802
Trainable params: 40,802
Non-trainable params: 0
_________________________________________________________________
```

3. LSTM, BiLSTM và Embedding.

Embedding sẽ chuyển từ thành vector đặc (dense vector). Đây là một trong các kỹ thuật được dùng nhiều trong NLP nhằm biểu diễn ngữ nghĩa cho một từ.

Lớp Embedding trong keras có các thông số quan trọng sau:

+ *input_dim*: kích thước từ vựng. thường là giá trị: len(word_index)+1.
+ *output_dim*: chiều của vector (đôi khi còn được gọi là embedding_dim).
+ *embeddings_initializer*: phương pháp khởi tạo giá trị của embedding.
+ *embeddings_regularizer*: chuẩn hoá giá trị của embedding.

Xem thêm về Embedding: https://keras.io/api/layers/core_layers/embedding/

Lớp LSTM: Xây dựng một lớp LSTM cho mạng neural. Các thông số quan trọng:

+ *units*: số lượng units trong layer..

+ *activation*: hàm kích hoạt.

+ *dropout*: sử dụng dropount hay không.

+ *return_sequences*: trả về output là một sequence (đối với Many-to-many) hoặc là một giá trị (Many-to-one).

+ *recurrent_initializer, kernel_initializer và bias_initializer*: khởi tạo giá trị.

+ *recurrent_regularizer, kernel_regularizer*: chuẩn hoá mạng neural.

Xem thêm: https://keras.io/api/layers/recurrent_layers/lstm/

```
model = Sequential()
model.add(Input(shape=(None, ), dtype="float64"))
model.add(Embedding(len(word_index), 128))
model.add(LSTM(200, return_sequences=False))
model.add(Dense(2, activation='sigmoid'))
```

Hãy thực hiện lại và xem kết quả cải thiện như thế nào nhé :))

Lớp Bidirectional: sử dụng để xây dựng BiLSTM hoặc là BiGRU. Có 2 thông số quan trọng gồm:

+ *layer*: Một lớp (layer) của keras, có thể là LSTM hoặc GRU.

+ *merge_mode*: phương pháp dùng để merge giá trị forward và backward (xem thêm về LSTM và GRU để rõ hơn). Mặc định là "concat".

Xem thêm: https://keras.io/api/layers/recurrent_layers/bidirectional/

Sử dụng Bidirectional để xây dựng lớp BiLSTM:

```
model = Sequential()
model.add(Input(shape=(None, ), dtype="float64"))
model.add(Embedding(len(word_index), 128))
model.add(Bidirectional(LSTM(200,
return_sequences=False)))
model.add(Dense(2, activation='sigmoid'))
```

BÀI TẬP:

Bài 1: Tăng số lượng epoch lên 20 và thực hiện huấn luyện mô hình. Cho biết độ chính xác của mô hình là bao nhiêu? Vẽ đồ thị học.

Bài 2: Hãy thử thêm 1 lớp Simple RNN nữa vào mô hình và cho biết kết quả độ chính xác của mô hình như thế nào?

Ghi chú: Thêm vào trước lớp SimpleRNN hiện tại, đặt return_sequence = True.

Bài 3: Xây dựng mô hình gồm 1 lớp Embedding và 1 lớp LSTM kết nối với nhau. Cho biết độ chính xác của mô hình là bao nhiêu? Vẽ đồ thị học.

Bài 4: Xây dựng mô hình gồm 1 lớp Embedding và 2 lớp LSTM kết nối với nhau. Cho biết độ chính xác của mô hình là bao nhiêu? Vẽ đồ thị học.

Bài 5: Tìm hiểu GRU và xây dựng mô hình với lớp GRU (mô hình giống Bài 3, nhưng thay lớp LSTM thành GRU). So sánh kết quả của 2 mô hình.