Họ và tên: Trần Vân Anh
MSV: B20DCCN075

BÀI TẬP 5: Xử lý ngôn ngữ tự nhiên

5.1. Hiển thị quá trình học với keras

a. Trình bày hiểu biết của bạn về hiển thị quá trình học với keras

1. Giới thiệu

Khi huấn luyện một mô hình học sâu, việc theo dõi và đánh giá hiệu suất của mô hình theo thời gian rất quan trọng để hiểu cách mô hình đang học và cải thiện.
Keras là một thư viện mạnh mẽ trong Python cung cấp một giao diện sạch sẽ để tạo ra các mô hình học sâu và bao gồm các backend kỹ thuật hơn như TensorFlow và Theano.
Để hiển thị và quan sát hiệu suất của mô hình học sâu theo thời gian, chúng ta sử dụng lịch sử huấn luyện của mô hình, được trả về từ hàm fit() trong Keras. Lịch sử huấn luyện bao gồm các thông tin về mất mát (loss) và độ chính xác (accuracy) trong suốt quá trình huấn luyện.

2. Access Model Training History in Keras

Keras cung cấp khả năng đăng ký các "callback" (gọi là "gọi lại" trong tiếng Việt) khi huấn luyện một mô hình học sâu.
Một trong những "callback" mặc định được đăng ký khi huấn luyện tất cả các mô hình học sâu là "History callback". "History callback" ghi lại các thông số huấn luyện cho mỗi epoch. Điều này bao gồm mất mát và độ chính xác (đối với các vấn đề phân loại), cũng như mất mát và độ chính xác cho tập validation nếu được thiết lập.
Đối tượng "history" (lịch sử) được trả về từ các cuộc gọi của hàm "fit()" được sử dụng để huấn luyện mô hình. Các chỉ số (metrics) được lưu trữ trong một từ điển (dictionary) trong thành viên "history" của đối tượng được trả về. Điều này cho phép quản lý và truy xuất thông tin quan trọng về quá trình huấn luyện, bao gồm mất mát và độ chính xác, để sau đó có thể sử dụng chúng để phân tích và trực quan hóa quá trình huấn luyện một cách hiệu quả.

Ví dụ:

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt

# Tạo dữ liệu giả lập
X = np.random.random((1000, 20))
y = np.random.randint(2, size=(1000, 1))

# Xây dựng mô hình
model = Sequential()
model.add(Dense(12, input_dim=20, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile mô hình
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Huấn luyện mô hình và lấy lịch sử huấn luyện
history = model.fit(X, y, validation_split=0.2, epochs=10, batch_size=16, verbose=1)

# Lấy thông tin từ lịch sử huấn luyện
training_loss = history.history['loss']
training_accuracy = history.history['accuracy']
validation_loss = history.history['val_loss']
validation_accuracy = history.history['val_accuracy']

# Hiển thị biểu đồ mất mát
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(training_loss, label='Training Loss')
plt.plot(validation_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
```

```
# Hiển thị biểu đồ độ chính xác
plt.subplot(1, 2, 2)
plt.plot(training_accuracy, label='Training Accuracy')
plt.plot(validation_accuracy, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')

plt.tight_layout()
plt.show()
```

Trong ví dụ này, chúng ta tạo một mô hình đơn giản và huấn luyện nó trên dữ liệu giả lập. Lịch sử huấn luyện được ghi lại vào biến history. Sau đó, chúng ta trích xuất thông tin về mất mát và độ chính xác từ lịch sử này và vẽ biểu đồ để trực quan hóa hiệu suất của mô hình trong quá trình huấn luyện.

3. Visualize Model Training History in Keras

Trong phần này, chúng ta sẽ tạo biểu đồ để trực quan hóa lịch sử huấn luyện mô hình sử dụng dữ liệu đã thu thập.

Biểu đồ độ chính xác (Accuracy):
Đầu tiên, chúng ta sẽ vẽ biểu đồ độ chính xác trên tập huấn luyện và tập validation theo số epoch. Điều này giúp chúng ta quan sát sự tiến bộ của mô hình trên cả tập huấn luyện và tập validation.

Biểu đồ mất mát (Loss):
Tiếp theo, chúng ta sẽ vẽ biểu đồ mất mát trên tập huấn luyện và tập validation theo số epoch. Điều này cho phép chúng ta xem xét sự giảm dần của mất mát và đánh giá việc học của mô hình.

Ví dụ:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Tạo dữ liệu giả lập
X, Y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.2, random_state=42)

# Xây dựng mô hình
model = Sequential()
model.add(Dense(12, input_dim=20, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile mô hình
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Huấn luyện mô hình và lấy lịch sử huấn luyện
history = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), epochs=150, batch_size=10, v

# Lấy thông tin từ lịch sử huấn luyện
training_loss = history.history['loss']
training_accuracy = history.history['accuracy']
validation_loss = history.history['val_loss']
validation_accuracy = history.history['val_accuracy']

# Hiển thị biểu đồ độ chính xác
plt.plot(training_accuracy, label='Training Accuracy')
plt.plot(validation_accuracy, label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Hiển thị biểu đồ mất mát
plt.plot(training_loss, label='Training Loss')
plt.plot(validation_loss, label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Trong ví dụ này, chúng ta sử dụng một tập dữ liệu giả lập với 1000 mẫu và 20 đặc trưng. Mô hình được huấn luyện và sau đó chúng ta trích xuất lịch sử huấn luyện để tạo và hiển thị các biểu đồ độ chính xác và mất mát trên tập huấn luyện và tập validation.

Bằng việc quan sát biểu đồ độ chính xác, ta có thể nhận thấy rằng mô hình có thể cần được huấn luyện thêm một chút, vì xu hướng độ chính xác trên cả tập huấn luyện và tập validation vẫn đang tăng trong vài epoch cuối cùng. Đồng thời, ta cũng thấy rằng mô hình chưa học quá mức trên tập huấn luyện, cho thấy khả năng tương đương trên cả hai tập dữ liệu.

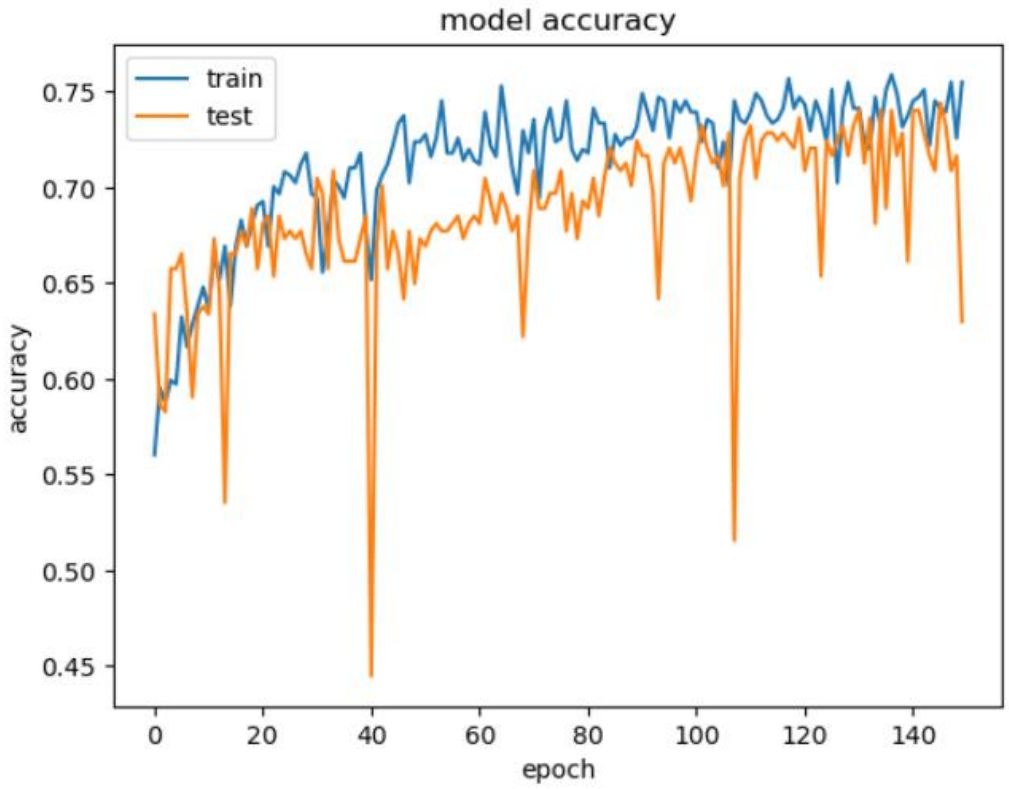
Thông qua việc phân tích này, ta có thể xác định liệu cần thêm epoch nữa để cải thiện hiệu suất, hay có thể dừng quá trình huấn luyện để tránh việc mô hình học quá mức (overfitting) trên dữ liệu huấn luyện. Điều này là quan trọng để điều chỉnh quá trình huấn luyện và đảm bảo mô hình học một cách hiệu quả.

b. Chạy ví dụ sau và giải thích

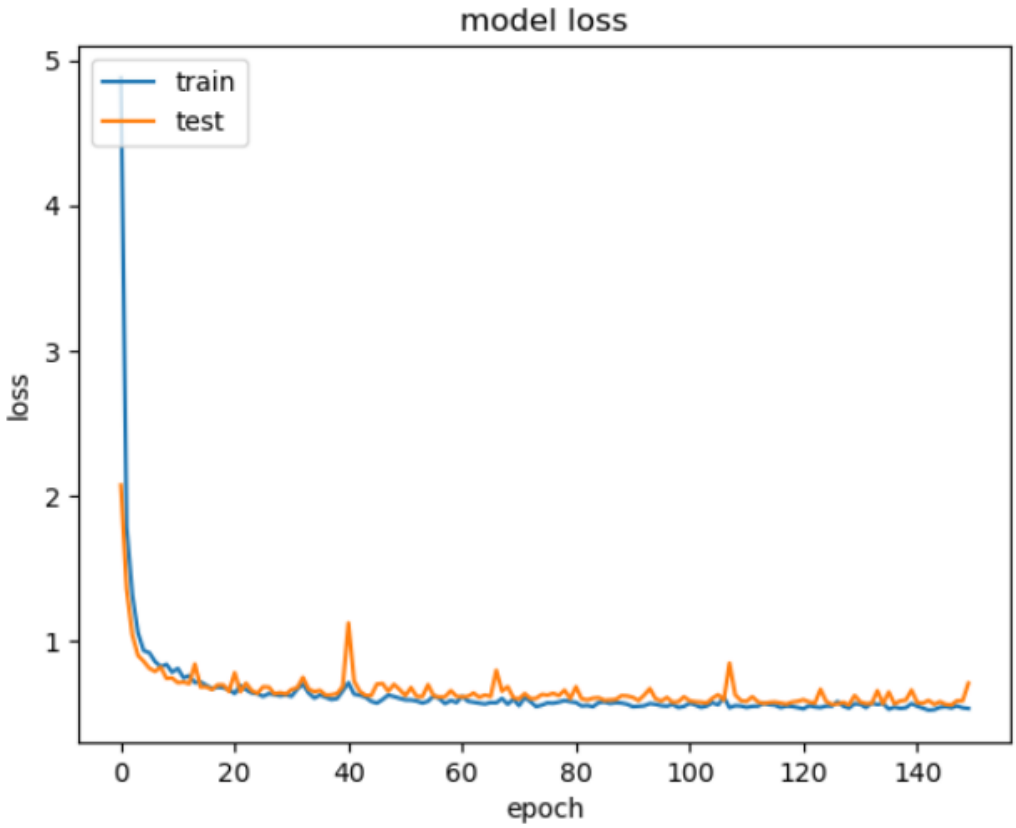
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
import numpy as np
# load pima indians dataset
dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter=",")
# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]
# create model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
history = model.fit(X, Y, validation_split=0.33, epochs=150, batch_size=10, verbose=0)
# list all data in history
print(history.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```




```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Giải thích code

1. Nhập các thư viện cần thiết:
Sequential: Đây là một mô hình mạng neural tuyến tính để xây dựng mô hình.
Dense: Đây là một lớp mạng fully connected.
matplotlib.pyplot: Thư viện dùng để tạo đồ thị.
numpy: Thư viện dùng cho các phép toán số học.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
import numpy as np
```

2. Tải dữ liệu từ tập tin CSV "pima-indians-diabetes.csv" bằng cách sử dụng hàm “loadtxt” từ NumPy. Dữ liệu được giả định là được phân tách bằng dấu phẩy.

```
dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter=",")
```

3. Chuẩn bị dữ liệu đầu vào và đầu ra
Tập dữ liệu được chia thành đặc trưng đầu vào (X) và nhãn đầu ra (Y).

```
X = dataset[:,0:8]   # Đặc trưng (cột từ 0 đến 7)
Y = dataset[:,8]  # Nhãn (cột 8)
```

4.Tạo mô hình

```
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Định nghĩa một mô hình tuần tự. Bao gồm ba lớp:
Lớp đầu tiên có 12 nơ-ron, sử dụng hàm kích hoạt ReLU và có 8 chiều đầu vào.
Lớp thứ hai có 8 nơ-ron và sử dụng hàm kích hoạt ReLU.
Lớp thứ ba (là lớp đầu ra) có 1 nơ-ron (được sử dụng cho phân loại nhị phân) và sử dụng hàm kích hoạt sigmoid.

5. Biên soạn mô hình

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

Mô hình được biên soạn với hàm mất mát binary cross-entropy (phù hợp cho phân loại nhị phân), tối ưu hóa bằng Adam, và độ chính xác được sử dụng làm chỉ số theo dõi trong quá trình huấn luyện.

6. Huấn luyện mô hình

```
history = model.fit(X, Y, validation_split=0.33, epochs=150, batch_size=10, verbose=0)
```

Mô hình được huấn luyện trên dữ liệu đã cho (X và Y) sử dụng phân chia kiểm tra 33%, trong 150 epoch, với kích thước batch là 10. Lịch sử huấn luyện được lưu trong đối tượng history.

7. Vẽ biểu đồ lịch sử huấn luyện

```
(history.history.keys())
# Vẽ đồ thị độ chính xác
plt.plot(history.history['accuracy'])
plt.plprintot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
```

```
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# Vẽ đồ thị mất mát
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

- Phương thức keys() in ra các khóa có sẵn trong đối tượng lịch sử (history), thường bao gồm 'loss', 'accuracy', 'val_loss', và 'val_accuracy'.
- Hai đồ thị được tạo ra: một cho độ chính xác của huấn luyện và kiểm tra, và một cho mất mát của huấn luyện và kiểm tra, sử dụng Matplotlib. Các đồ thị này giúp trực quan hóa sự thay đổi của hiệu suất mô hình qua các epoch trong quá trình huấn luyện. 'Train' đề cập đến tập huấn luyện và 'Test' đề cập đến tập kiểm tra.

5.2. Xử lý ngôn ngữ tự nhiên

a. Trình bày hiểu biết về xử lý ngôn ngữ tự nhiên

1. Word embeddings là gì?

Word embeddings, hay còn gọi là vectơ từ, là một phương pháp biểu diễn tài liệu và từ trong xử lý ngôn ngữ tự nhiên. Nó được định nghĩa như một dạng biểu diễn dưới dạng vectơ số cho phép các từ có nghĩa tương tự có cùng một biểu diễn số học. Điều này thường áp dụng một biểu diễn chiều thấp hơn để mô tả một từ trong không gian số học. Word embeddings có khả năng được đào tạo nhanh chóng hơn nhiều so với các mô hình xây dựng thủ công sử dụng cơ sở dữ liệu như WordNet.

Ví dụ: Một từ được biểu diễn bằng một vectơ có 50 giá trị có khả năng biểu thị 50 đặc trưng riêng biệt. Nhiều người thường sử dụng các mô hình word embeddings được đào tạo trước độ như Flair, fastText, SpaCy và các mô hình khác.

Đặc điểm chung của word embeddings là khả năng biểu diễn các từ dưới dạng vectơ số, và nó có thể áp dụng cho bất kỳ khía cạnh nào liên quan đến từ ngữ, ví dụ: tuổi,

thể thao, thể lực, công việc, và nhiều khía cạnh khác. Mỗi vectơ từ sẽ có các giá trị tương ứng với các đặc trưng này.

Mục tiêu của việc sử dụng word embeddings bao gồm:

1. Giảm kích thước biểu diễn từ để tiết kiệm không gian lưu trữ.

2. Sử dụng vectơ từ để dự đoán các từ liên quan trong ngữ cảnh.

3. Bắt chước ý nghĩa ngữ nghĩa của các từ.

Triển khai tính năng của word embeddings:

Word embeddings là một phương pháp để trích xuất các đặc trưng từ văn bản để sử dụng trong các mô hình học máy xử lý dữ liệu văn bản. Chúng cố gắng bảo tồn thông tin cú pháp và ngữ nghĩa của văn bản. Các phương pháp như Bag of Words (BOW), CountVectorizer và TFIDF dựa vào sự xuất hiện của các từ trong câu, nhưng không lưu trữ thông tin về cú pháp hoặc ngữ nghĩa. Trong các thuật toán này, kích thước của vectơ là số lượng từ trong từ vựng, có thể dẫn đến ma trận thưa nếu hầu hết các phần tử bằng 0. Việc sử dụng vectơ đầu vào lớn có thể dẫn đến tăng cường đáng kể về khối lượng tính toán trong quá trình đào tạo mô hình. Tính năng của word embeddings giúp giải quyết những thách thức này.-***Thuật toán nhúng từ:***

Các phương pháp nhúng từ nghiên cứu cách biểu diễn vectơ với giá trị thực cho từ vựng cố định từ kho văn bản trước đây. Quá trình học tập kết hợp với mô hình mạng thần kinh trong nhiều nhiệm vụ, ví dụ như phân loại tài liệu hoặc trong các quá trình không giám sát sử dụng số liệu thống kê về tài liệu. Phần này sẽ xem xét ba kỹ thuật khác nhau có thể áp dụng để học cách nhúng từ từ dữ liệu văn bản.

+ Embedding Layer:

Lớp nhúng, còn được gọi là lớp nhúng từ (hoặc Embedding Layer), là một phần quan trọng trong mô hình mạng thần kinh dành cho xử lý ngôn ngữ tự nhiên. Lớp nhúng này thường được học cùng với mô hình mạng thần kinh trong các nhiệm vụ cụ thể của xử lý ngôn ngữ tự nhiên, chẳng hạn như mô hình hóa ngôn ngữ hoặc phân loại tài liệu.

Để sử dụng lớp nhúng, trước hết, văn bản tài liệu phải được xử lý và chuẩn bị sao cho mỗi từ được mã hóa một lần. Kích thước của không gian vectơ được định trước và thường được xác định trong mô hình, ví dụ như kích thước 50, 100, hoặc 300. Các vectơ này thường được khởi tạo với các giá trị ngẫu nhiên nhỏ. Lớp nhúng được

đặt ở phần đầu của mạng thần kinh và được cập nhật theo cách được điều chỉnh thông qua thuật toán Lan truyền ngược trong quá trình đào tạo.

Các từ, sau khi được mã hóa một lần, được ánh xạ thành các vectơ từ. Nếu sử dụng mô hình Perceptron nhiều lớp, các vectơ từ thường được nối với nhau trước khi được sử dụng làm đầu vào cho mô hình. Trong trường hợp sử dụng mạng nơ-ron hồi quy, mỗi từ có thể được sử dụng như một đầu vào riêng lẻ trong chuỗi.

Cách tiếp cận này trong việc học lớp nhúng đòi hỏi một lượng lớn dữ liệu đào tạo và có thể mất thời gian đào tạo, tuy nhiên, nó sẽ học cách nhúng mục tiêu vào dữ liệu văn bản cụ thể và nhiệm vụ NLP đang được thực hiện.

+ Bag of words (BOW):

Túi từ là một trong những kỹ thuật nhúng từ phổ biến của văn bản trong đó mỗi giá trị trong vectơ sẽ biểu thị số từ trong tài liệu/câu. Nói cách khác, nó trích xuất các đặc điểm từ văn bản. Chúng tôi cũng gọi nó là vector hóa.

Cách tiến hành tạo BOW.

Ở bước đầu tiên, bạn phải mã hóa văn bản thành câu.

Tiếp theo, các câu được mã hóa ở bước đầu tiên có thêm các từ được mã hóa.

Loại bỏ bất kỳ từ dừng hoặc dấu câu.

Sau đó, chuyển đổi tất cả các từ thành chữ thường.

Cuối cùng di chuyển để tạo biểu đồ phân bố tần suất của các từ.

Chúng ta sẽ thảo luận về BOW với ví dụ thích hợp trong túi lựa chọn từ liên tục bên dưới.

+Word2Vec:

Phương pháp Word2Vec được Google phát triển vào năm 2013. Hiện tại, chúng sử dụng kỹ thuật này cho tất cả các bài toán xử lý ngôn ngữ tự nhiên (NLP) nâng cao. Nó được phát minh để đào tạo cách nhúng từ và dựa trên giả thuyết phân phối. Trong Word2Vec mỗi từ được gán một vectơ. Chúng ta bắt đầu với một random vector hoặc one-hot vector.

One-hot vector: Biểu diễn trong đó chỉ có một bit trong vectơ là 1. Nếu có 500 từ trong kho văn bản thì độ dài vectơ sẽ là 500. Sau khi gán vectơ cho mỗi từ, chúng ta lấy kích thước cửa sổ và lặp qua toàn bộ kho văn bản . Trong khi chúng tôi thực hiện việc này, có hai phương pháp nhúng thần kinh được sử dụng:

Trong giả thuyết này, nó sử dụng Skip-gram hoặc một túi từ liên tục (CBOW).

Về cơ bản, đây là các mạng thần kinh nông có lớp đầu vào, lớp đầu ra và lớp chiếu. Nó tái tạo lại bối cảnh ngôn ngữ của các từ bằng cách xem xét cả trật tự của các từ trong lịch sử cũng như tương lai.

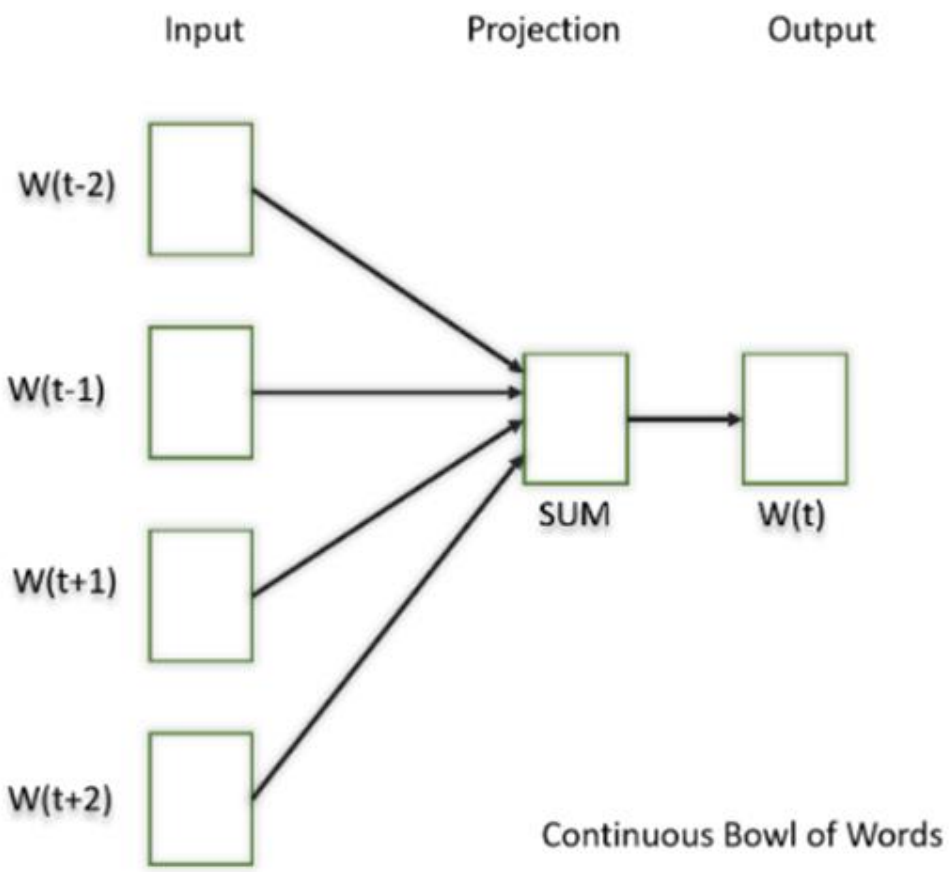
Phương pháp này bao gồm việc lặp lại một tập văn bản để tìm hiểu mối liên hệ giữa các từ. Nó dựa trên giả thuyết rằng các từ lân cận trong văn bản có những điểm tương đồng về ngữ nghĩa với nhau. Nó hỗ trợ ánh xạ các từ tương tự về mặt ngữ nghĩa với các vectơ nhúng gần về mặt hình học.

Word2Vec có hai biến thể dựa trên mạng thần kinh: Túi từ liên tục (CBOW) và Skip-gram.

a. CBOW

- Biến thể túi từ liên tục bao gồm nhiều đầu vào khác nhau được mô hình mạng thần kinh lấy . Ngoài ra, nó dự đoán từ được nhắm mục tiêu có liên quan chặt chẽ đến ngữ cảnh của các từ khác nhau được cung cấp làm đầu vào. Đó là một cách nhanh chóng và tuyệt vời để tìm cách biểu diễn bằng số tốt hơn cho các từ xuất hiện thường xuyên.

Trong CBOW, chúng tôi xác định kích thước cửa sổ. Từ ở giữa là từ hiện tại và các từ xung quanh (từ quá khứ và tương lai) là ngữ cảnh. CBOW sử dụng ngữ cảnh để dự đoán các từ hiện tại. Hay là cố gắng khớp các từ lân cận trong cửa sổ với từ trung tâm. Mỗi từ được mã hóa bằng One Hot Encoding trong từ vựng được xác định và gửi đến mạng nơ-ron CBOW.

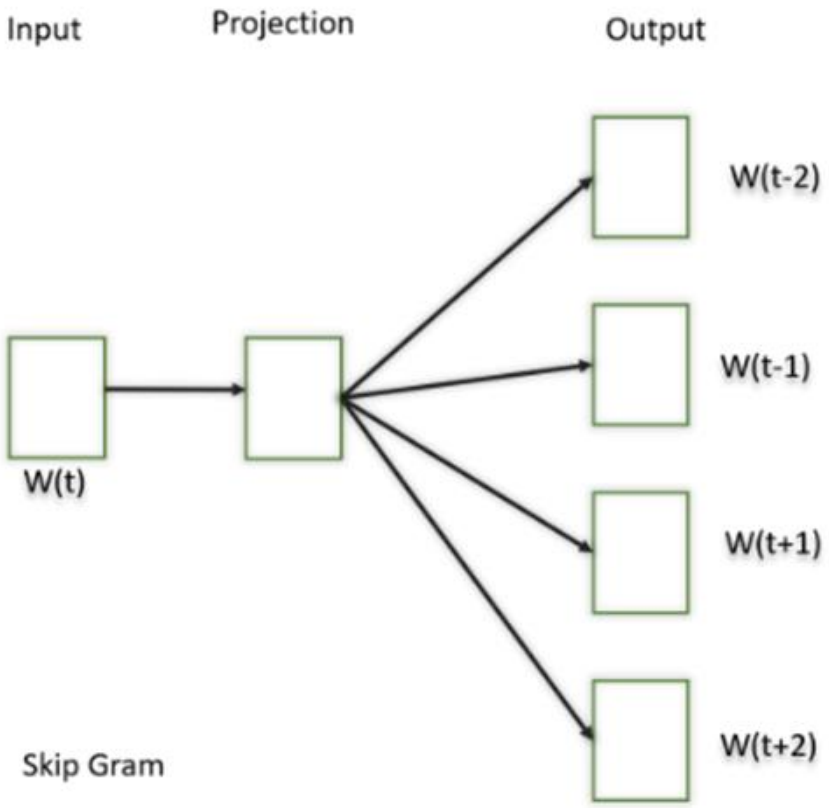


Lớp ẩn là lớp dày đặc được kết nối đầy đủ tiêu chuẩn. Lớp đầu ra tạo ra xác suất cho từ mục tiêu từ từ vựng.

Để tạo BOW, Sử dụng CountVectorizer của Scikit-learn, công cụ mã hóa một bộ sưu tập tài liệu văn bản, xây dựng vốn từ vựng của các từ đã biết và mã hóa tài liệu mới bằng từ vựng đó.

b. Skip Gram

Skip-gram là một kỹ thuật nhúng từ ngược lại so với CBOW vì nó không dự đoán từ hiện tại dựa trên ngữ cảnh. Thay vào đó, mỗi từ hiện tại được sử dụng làm đầu vào cho bộ phân loại log-tuyến tính cùng với lớp chiếu liên tục. Bằng cách này, nó dự đoán các từ trong một phạm vi nhất định trước và sau từ hiện tại. Trong mô hình này, chúng tôi cố gắng làm cho từ trung tâm gần hơn với các từ lân cận. Nó cho thấy rằng phương pháp này tạo ra các phần nhúng có ý nghĩa hơn.



Biến thể này chỉ lấy một từ làm đầu vào và sau đó dự đoán các từ ngữ cảnh có liên quan chặt chẽ. Đó là lý do nó có thể biểu diễn những từ hiếm một cách hiệu quả.

Sau khi áp dụng các phương pháp nhúng thần kinh ở trên, chúng ta thu được các vectơ đã huấn luyện của từng từ sau nhiều lần lặp trong kho văn bản. Các vectơ được huấn luyện này bảo toàn thông tin cú pháp hoặc ngữ nghĩa và được chuyển đổi sang các chiều thấp hơn. Các vectơ có ý nghĩa hoặc thông tin ngữ nghĩa tương tự nhau được đặt gần nhau trong không gian.

-GloVe

Phương pháp nhúng từ GloVe trong NLP được phát triển tại Stanford bởi Pennington và cộng sự. Nó được gọi là vectơ toàn cầu vì số liệu thống kê kho văn bản toàn cầu được mô hình nắm bắt trực tiếp. Nó tìm thấy hiệu suất tuyệt vời trong sự tương tự thế giới và các vấn đề nhận dạng thực thể được đặt tên.

Đây là một phương pháp khác để tạo từ nhúng. Trong phương pháp này, GloVe xây dựng một ma trận ngữ cảnh từ hoặc từ xuất hiện rõ ràng bằng cách sử dụng số liệu thống kê trên toàn bộ kho văn bản. Chúng tôi nhận được ma trận sự xuất hiện thông

qua điều này. Các từ xuất hiện cạnh nhau có giá trị là 1, nếu chúng cách nhau một từ thì 1/2, nếu chúng cách nhau hai từ thì 1/3, v.v.

Chúng ta hãy lấy một ví dụ để hiểu cách tạo ma trận. Chúng tôi có một kho văn bản nhỏ:

```
Corpus:
Thật là một buổi tối đẹp trời.
Buổi tối vui vẻ!
Đây là một buổi tối đẹp trời phải không?
```

	Nó	là	Một	Đẹp	buổi tối	Tốt
Nó	0					
là	1+1	0				
Một	1/2+1	1+1/2	0			
Đẹp	1/3+1/2	1/2+1/3	1+1	0		
buổi tối	1/4+1/3	1/3+1/4	1/2+1/2	1+1	0	
Tốt	0	0	0	0	1	0

Nửa trên của ma trận sẽ phản ánh nửa dưới. Chúng ta cũng có thể xem xét khung cửa sổ để tính toán số lần xuất hiện đồng thời bằng cách dịch chuyển khung cho đến

hết kho văn bản. Điều này giúp thu thập thông tin về ngữ cảnh mà từ đó được sử dụng.

Ban đầu, các vectơ cho mỗi từ được gán ngẫu nhiên. Sau đó, chúng ta lấy hai cặp vectơ và xem chúng gần nhau như thế nào trong không gian. Nếu chúng xuất hiện cùng nhau thường xuyên hơn hoặc có giá trị cao hơn trong ma trận sự xuất hiện và cách xa nhau trong không gian thì chúng sẽ được đưa đến gần nhau. Nếu chúng ở gần nhau nhưng hiếm khi hoặc không thường xuyên được sử dụng cùng nhau thì chúng sẽ bị dịch chuyển ra xa nhau hơn trong không gian.

Sau nhiều lần lặp lại quy trình trên, chúng ta sẽ có được biểu diễn không gian vectơ gần đúng với thông tin từ ma trận sự xuất hiện. Hiệu suất của GloVe tốt hơn Word2Vec về cả khả năng nắm bắt ngữ nghĩa và cú pháp.

Mô hình nhúng từ được đào tạo trước:

Mọi người thường sử dụng các mô hình được đào tạo trước để nhúng từ. Một vài trong số đó là:

SpaCy

văn bản nhanh

Sự tinh tế, v.v.

Lợi ích của việc sử dụng tính năng nhúng từ:

Đào tạo nhanh hơn nhiều so với các mô hình xây dựng thủ công như WordNet (sử dụng nhúng biểu đồ)

Hầu như tất cả các ứng dụng NLP hiện đại đều bắt đầu bằng lớp nhúng

Nó lưu trữ một ý nghĩa gần đúng

Hạn chế của việc nhúng từ:

Nó có thể tốn nhiều bộ nhớ

Nó phụ thuộc vào ngữ liệu. Bất kỳ sự thiên vị cơ bản nào cũng sẽ có ảnh hưởng đến mô hình của bạn

Nó không thể phân biệt giữa các từ đồng âm. Ví dụ: phanh/ngắt, di động/bán, thời tiết/liệu, v.v.

b. Chạy code và giải thích

· Chạy code

```
# Import các thư viện cần thiết từ Keras
from keras.datasets import imdb
from keras import preprocessing
from keras.layers import Embedding
from keras.models import Sequential
from keras.layers import Flatten, Dense
import matplotlib.pyplot as plt
import numpy as np

# Số lượng từ tối đa trong từ điển
max_features = 10000

# Độ dài tối đa cho mỗi bình luận
maxlen = 20

# Tải dữ liệu IMDB và giới hạn số lượng từ vựng sử dụng (max_features)
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# Chuyển đổi độ dài các mẫu thành cùng một độ dài (maxlen) bằng cách thêm padding hoặc cắt bớt
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)

# Tạo mô hình Sequential
model = Sequential()

# Thêm lớp Embedding với kích thước đầu vào là 10000, kích thước vector output là 8, và độ dài đầu vào là maxlen
model.add(Embedding(10000, 8, input_length=maxlen))

# Thêm lớp Flatten để chuyển đổi tensor thành vector
model.add(Flatten())

# Thêm lớp Dense với 1 đơn vị đầu ra và hàm kích hoạt sigmoid để thực hiện phân loại nhị phân
model.add(Dense(1, activation='sigmoid'))

# Compile mô hình với bộ tối ưu hóa rmsprop, hàm mất mát binary_crossentropy và độ đo là accuracy
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

# Hiển thị thông tin về mô hình
model.summary()
```

o Output

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 20, 8)             80000

 flatten_4 (Flatten)         (None, 160)               0

 dense_4 (Dense)             (None, 1)                 161

=================================================================
Total params: 80161 (313.13 KB)
Trainable params: 80161 (313.13 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

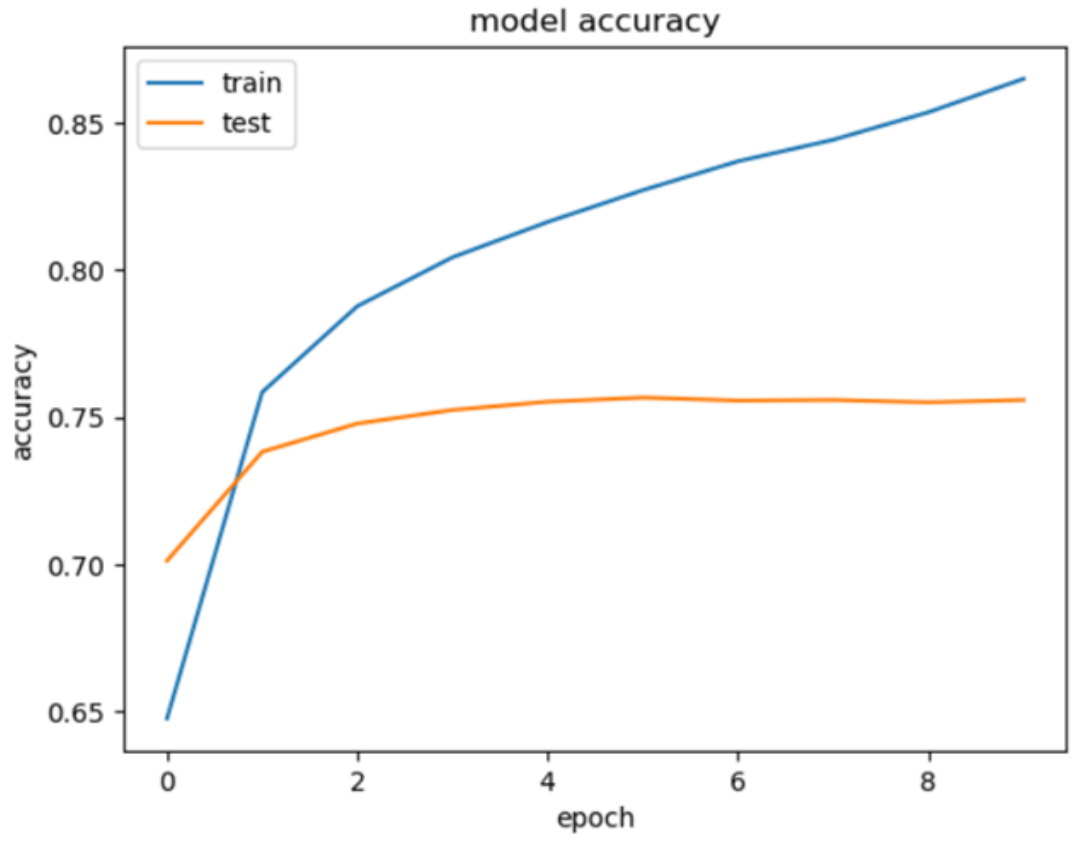
```
# Huấn luyện mô hình trên tập dữ liệu x_train và y_train trong 10 epochs và batch_size là 32
history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

o Output

```
Epoch 1/10
625/625 [==============================] - 3s 4ms/step - loss: 0.6571 - acc: 0.6477 - val_loss: 0.5945 - val_acc: 0.7012
Epoch 2/10
625/625 [==============================] - 2s 3ms/step - loss: 0.5220 - acc: 0.7584 - val_loss: 0.5153 - val_acc: 0.7382
Epoch 3/10
625/625 [==============================] - 2s 3ms/step - loss: 0.4561 - acc: 0.7876 - val_loss: 0.4974 - val_acc: 0.7478
Epoch 4/10
625/625 [==============================] - 2s 3ms/step - loss: 0.4252 - acc: 0.8043 - val_loss: 0.4954 - val_acc: 0.7524
Epoch 5/10
625/625 [==============================] - 2s 3ms/step - loss: 0.4050 - acc: 0.8163 - val_loss: 0.4949 - val_acc: 0.7552
Epoch 6/10
625/625 [==============================] - 2s 3ms/step - loss: 0.3875 - acc: 0.8270 - val_loss: 0.4978 - val_acc: 0.7566
Epoch 7/10
625/625 [==============================] - 2s 3ms/step - loss: 0.3713 - acc: 0.8368 - val_loss: 0.5011 - val_acc: 0.7556
Epoch 8/10
625/625 [==============================] - 7s 12ms/step - loss: 0.3559 - acc: 0.8442 - val_loss: 0.5059 - val_acc: 0.7558
Epoch 9/10
625/625 [==============================] - 2s 3ms/step - loss: 0.3394 - acc: 0.8535 - val_loss: 0.5099 - val_acc: 0.7550
Epoch 10/10
625/625 [==============================] - 2s 3ms/step - loss: 0.3231 - acc: 0.8648 - val_loss: 0.5154 - val_acc: 0.7558
```

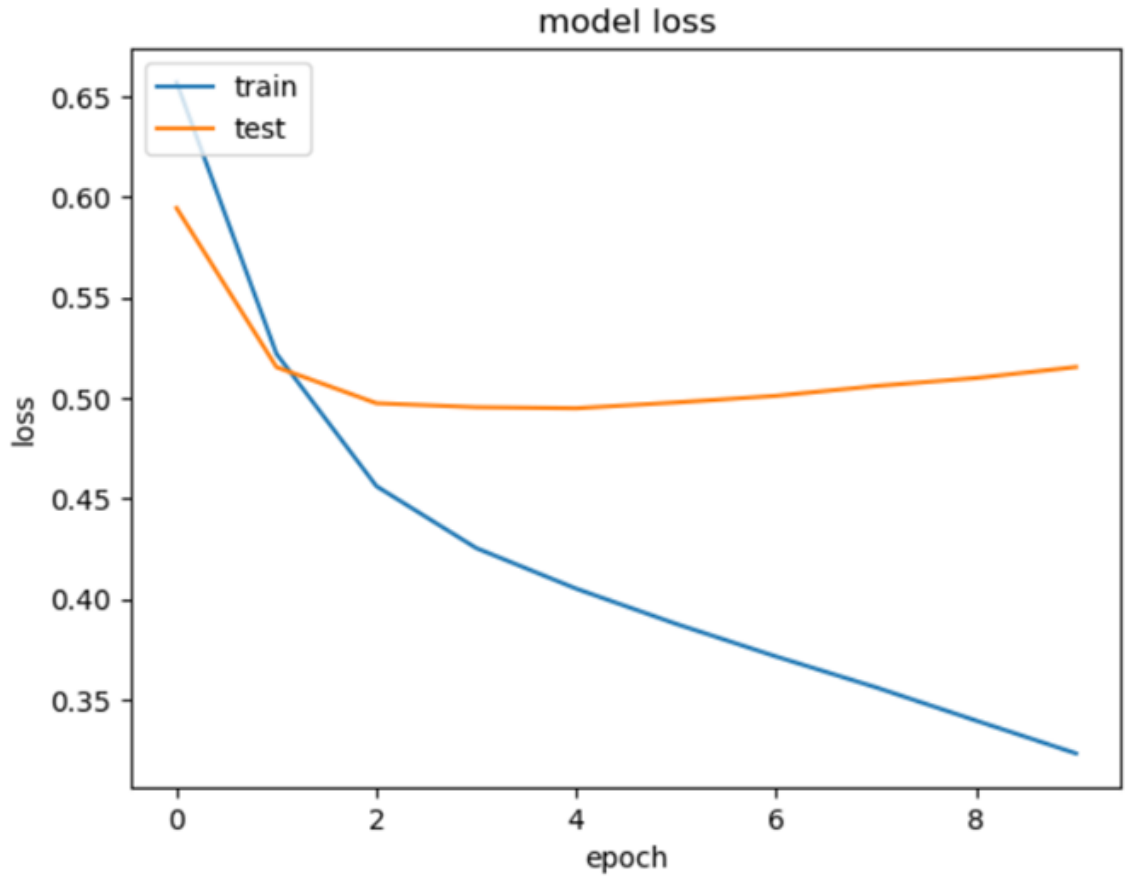
```python
# Vẽ biểu đồ độ chính xác trên tập huấn luyện và tập kiểm tra
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

- Output



```
# Vẽ biểu đồ hàm mất mát trên tập huấn luyện và tập kiểm tra
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

- Output



Giải thích code

Code này sử dụng dữ liệu từ bộ dữ liệu IMDB để xây dựng mô hình phân loại văn bản dựa trên nội dung các bình luận phim. Code tiến hành tiền xử lý dữ liệu, xây dựng mô hình mạng nơ-ron, và sau đó huấn luyện mô hình và vẽ biểu đồ để theo dõi độ chính xác và hàm mất mát qua các epoch.

1. keras.datasets: Thư viện này cung cấp các tập dữ liệu mẫu để huấn luyện mô hình. Trong code này, nó được sử dụng để tải dữ liệu IMDB, một tập dữ liệu về các bình luận phim.

2. keras.preprocessing: Thư viện này cung cấp các công cụ cho việc tiền xử lý dữ liệu. Trong code này, nó được sử dụng để tiền xử lý văn bản, bao gồm cắt và thêm padding cho các bình luận.

3. keras.layers: Thư viện này chứa các lớp mạng nơ-ron chuẩn được sử dụng để xây dựng mô hình. Trong code này, nó được sử dụng để thêm các lớp như Embedding, Flatten và Dense vào mô hình.

4. keras.models: Thư viện này chứa Sequential, một kiểu mô hình mạng nơ-ron phổ biến trong Keras, cho phép bạn xây dựng mô hình mạng nơ-ron bằng cách thêm lần lượt các lớp vào mô hình.

5. matplotlib.pyplot: Thư viện này được sử dụng để vẽ biểu đồ và hiển thị đồ thị. Trong code này, nó được sử dụng để vẽ biểu đồ độ chính xác và hàm mất mát sau quá trình huấn luyện.

6. numpy: Thư viện cho tính toán số học. Trong code này, nó có thể được sử dụng để xử lý các phần của dữ liệu một cách thuận tiện.

7. max_features: Biến này xác định số lượng từ vựng tối đa được sử dụng trong tập dữ liệu. Chỉ các từ xuất hiện thường xuyên nhất trong tập dữ liệu sẽ được sử dụng, và số lượng từ này không vượt quá giới hạn của max_features.

8. maxlen: Độ dài tối đa cho mỗi bình luận. Các bình luận dài hơn sẽ bị cắt bớt, và các bình luận ngắn hơn sẽ được thêm padding để có độ dài maxlen.

9. (x_train, y_train), (x_test, y_test): Dữ liệu huấn luyện và kiểm tra được chia thành hai phần, mỗi phần bao gồm các mẫu và nhãn tương ứng. x_train và x_test chứa các mẫu dữ liệu, và y_train và y_test chứa nhãn tương ứng cho mỗi mẫu.

10. Sequential: Lớp này được sử dụng để xây dựng và định nghĩa mô hình mạng nơ-ron tuần tự. Mô hình mạng nơ-ron tuần tự là một kiểu mô hình mạng nơ-ron phổ biến trong deep learning, nơi các lớp mạng nơ-ron được xếp chồng lên nhau một cách tuần tự, từ lớp đầu tiên đến lớp cuối cùng.

11. Embedding: Lớp này thực hiện biểu diễn từ điển dưới dạng các vector số thực có kích thước 8 cho mỗi từ trong văn bản đầu vào. Lớp Embedding được sử dụng để biểu diễn các từ dưới dạng các vector có chiều thấp hơn để đưa vào mạng nơ-ron.

12. Flatten: Lớp này được sử dụng để chuyển đổi tensor thành vector. Trong trường hợp này, nó được sử dụng để làm phẳng dữ liệu sau khi áp dụng lớp Embedding.

13. Dense: Lớp này thêm một lớp fully connected layer với 1 đơn vị đầu ra và hàm kích hoạt sigmoid để thực hiện phân loại nhị phân. Trong trường hợp này, nó dự đoán xem một bình luận có tích cực (positive) hay tiêu cực (negative).

14. compile(): Phương thức này cấu hình mô hình với các thông số tối ưu hóa (ở đây là 'rmsprop'), hàm mất mát (binary_crossentropy) và các độ đo (accuracy) được sử dụng trong quá trình huấn luyện.

15. fit(): Phương thức này huấn luyện mô hình trên tập dữ liệu huấn luyện (x_train, y_train) trong một số lượng epochs cụ thể (ở đây là 10) với kích thước batch cụ thể (ở đây là 32). Nó trả về lịch sử của quá trình huấn luyện, bao gồm các giá trị của độ chính xác và hàm mất mát trên tập huấn luyện và tập kiểm tra sau mỗi epoch.

16. plt.plot(): Hàm này được sử dụng để vẽ biểu đồ độ chính xác và hàm mất mát trên tập huấn luyện và tập kiểm tra sau quá trình huấn luyện. Nó sử dụng dữ liệu lịch sử trả về từ fit() để vẽ biểu đồ.