Câu 5.1: Hiển thị quá trình học với keras

a. Hiển thị quá trình học với keras

Hiển thị quá trình đào tạo mô hình Deep Learning trong Keras :

- Import thư viện và chuẩn bị dữ liệu: Bạn bắt đầu bằng việc import các thư viện như NumPy để xử lý dữ liệu và Matplotlib để vẽ biểu đồ (nếu cần). Sau đó, bạn cần chuẩn bị dữ liệu đào tạo và kiểm tra.
- Xây dựng mô hình: Sử dụng Sequential model của Keras hoặc API chức năng để định nghĩa kiến trúc mô hình. Bạn có thể thêm các lớp như Dense layers, Convolutional layers, hoặc RNN layers tùy thuộc vào loại mô hình bạn muốn xây dựng.
- Biên soạn mô hình: Bạn cần chọn hàm mất mát (loss function) và thuật toán tối ưu hóa (optimizer) cho mô hình của mình bằng cách sử dụng phương thức compile. Điều này cũng có thể bao gồm đánh giá các độ đo hiệu suất như accuracy nếu bạn làm việc với bài toán phân loại.
- Đào tạo mô hình: Sử dụng phương thức fit để bắt đầu quá trình đào tạo. Truyền vào dữ liệu đào tạo và kiểm tra, số epochs (vòng lặp đào tạo), và các callbacks nếu cần thiết. Callbacks là các hàm được gọi sau mỗi epoch hoặc batch để thực hiện các tác vụ như lưu mô hình hoặc theo dõi mức độ học.
- Theo dõi quá trình đào tạo: Trong quá trình đào tạo, bạn có thể sử dụng các callbacks để theo dõi các thông số quan trọng như giá trị mất mát và accuracy. Bạn có thể lưu giá trị này và vẽ biểu đồ để hình dung sự phát triển của mô hình theo thời gian.
- Kiểm tra và đánh giá mô hình: Sau khi đào tạo xong, bạn có thể sử dụng mô hình để dự đoán dữ liệu mới hoặc đánh giá hiệu suất trên tập kiểm tra. Điều này giúp bạn đảm bảo mô hình hoạt động tốt trên dữ liệu không nhìn thấy trước.
- Tinh chỉnh và cải thiện: Dựa vào kết quả kiểm tra, bạn có thể tinh chỉnh mô hình bằng cách thay đổi kiến trúc, hàm mất mát, optimizer hoặc sử dụng kỹ thuật như tăng dữ liệu đào tạo hoặc sử dụng kỹ thuật Regularization để cải thiện hiệu suất.
- Triển khai và sử dụng mô hình: Cuối cùng, sau khi bạn đã hài lòng với mô hình của mình, bạn có thể triển khai nó để sử dụng trong các ứng dụng thực tế. Điều này có thể bao gồm việc tích hợp mô hình vào sản phẩm hoặc dịch vụ của bạn.

Điều quan trọng là quá trình này có thể lặp đi lặp lại nhiều lần để cải thiện mô hình của bạn và đảm bảo nó hoạt động tốt trên dữ liệu thực tế.

Bạn có thể tìm hiểu nhiều điều về mạng lưới thần kinh và mô hình học sâu bằng cách quan sát hiệu suất của chúng theo thời gian trong quá trình đào tạo.

Keras là một thư viện mạnh mẽ trong Python, cung cấp giao diện rõ ràng để tạo các mô hình học sâu và bao gồm các chương trình phụ trợ TensorFlow và Theano kỹ thuật hơn.

Bài này, bạn sẽ khám phá cách bạn có thể xem xét và hình dung hiệu suất của các mô hình học sâu theo thời gian trong quá trình đào tạo bằng Python với Keras.

Hãy bắt đầu dự án của bạn với cuốn sách mới của tôi "Deep Learning With Python", bao gồm hướng dẫn từng bước và tệp mã nguồn Python cho tất cả các ví dụ.

Cập nhật tháng 3/2017: Đã cập nhật cho Keras 2.0.2, TensorFlow 1.0.1 và Theano 0.9.0.

Cập nhật tháng 3/2018: Đã thêm liên kết thay thế để tải xuống tập dữ liệu.

Cập nhật tháng 9/2019: Đã cập nhật cho API Keras 2.2.5.

Cập nhật tháng 10/2019: Đã cập nhật cho API Keras 2.3.0.

Cập nhật tháng 7/2022: Đã cập nhật cho API TensorFlow 2.x.

Lịch sử đào tạo mô hình truy cập trong Keras

Keras cung cấp khả năng đăng ký lệnh gọi lại khi đào tạo mô hình học sâu.
Một trong những lệnh gọi lại mặc định được đăng ký khi đào tạo tất cả các mô hình học sâu là "History callback". Nó ghi lại số liệu đào tạo cho từng kỷ nguyên. Điều này bao gồm sự mất mát và độ chính xác (đối với các vấn đề phân loại) cũng như sự mất mát và độ chính xác đối với tập dữ liệu xác thực nếu được đặt.
Đối tượng lịch sử được trả về từ các lệnh gọi tới hàm fit() dùng để huấn luyện mô hình. Số liệu được lưu trữ trong từ điển trong lịch sử thành viên của đối tượng được trả về.

Ví dụ: bạn có thể liệt kê các số liệu được thu thập trong một đối tượng lịch sử bằng cách sử dụng đoạn mã sau sau khi đào tạo mô hình:

```
# list all data in history
print(history.history.keys())
```

Ví dụ: đối với một mô hình được đào tạo về vấn đề phân loại với tập dữ liệu xác thực, điều này có thể tạo ra danh sách sau:

```
['accuracy', 'loss', 'val_accuracy', 'val_loss']
```

Bạn có thể sử dụng dữ liệu được thu thập trong đối tượng lịch sử để tạo các plot.

Các sơ đồ có thể cung cấp dấu hiệu về những điều hữu ích trong quá trình đào tạo mô hình, chẳng hạn như:

- Tốc độ hội tụ của nó qua các kỷ nguyên (độ dốc)
- Liệu mô hình có thể đã hội tụ hay không (bình nguyên của đường)
- Chế độ có thể học quá mức dữ liệu huấn luyện hay không (biến đổi cho dòng xác nhận)
- Và hơn thế nữa

Trực quan hóa lịch sử đào tạo mô hình trong Keras

Bạn có thể tạo các ô từ dữ liệu lịch sử đã thu thập.

Trong ví dụ dưới đây, một mạng nhỏ để mô hình hóa vấn đề phân loại nhị phân bệnh tiểu đường ở người da đỏ Pima được tạo ra. Đây là một tập dữ liệu nhỏ có sẵn từ Kho lưu trữ máy học của UCI. Bạn có thể tải xuống tập dữ liệu và lưu dưới dạng pima-indians-diabetes.csv trong thư mục làm việc hiện tại của bạn.

Ví dụ thu thập lịch sử được trả về từ quá trình đào tạo mô hình và tạo hai biểu đồ:

- Biểu đồ về độ chính xác của tập dữ liệu huấn luyện và xác nhận qua các giai đoạn huấn luyện
- Biểu đồ tổn thất trên các tập dữ liệu huấn luyện và xác nhận qua các giai đoạn huấn luyện

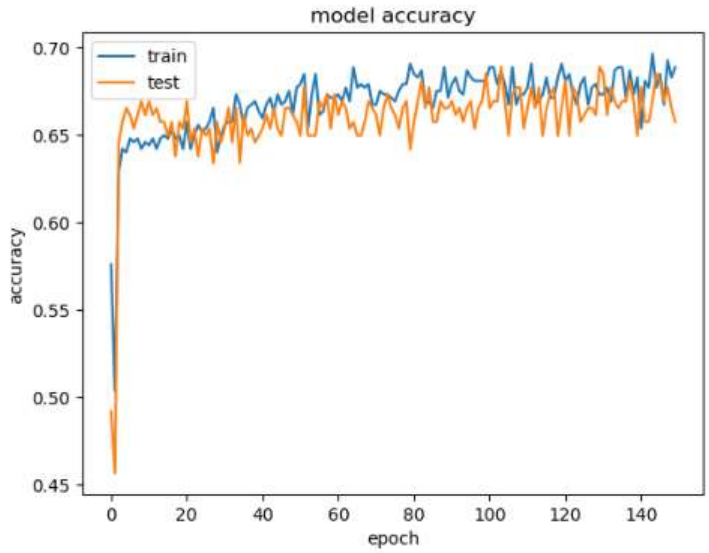
Lịch sử của tập dữ liệu xác thực được gắn nhãn thử nghiệm theo quy ước vì đây thực sự là tập dữ liệu thử nghiệm cho mô hình.

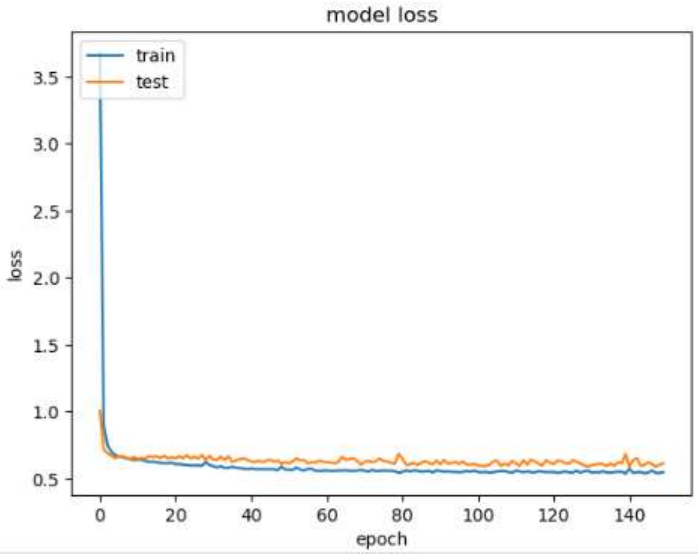
b. Chạy code

5.1

```
# Import các thư viện cần thiết
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
import numpy as np

# Load dữ liệu từ tập tin "pima-indians-diabetes.csv" và chia thành input (X) và output (Y)
dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter=",")
X = dataset[:, 0:8]  # Sử dụng cột từ 0 đến 7 làm đặc trưng đầu vào
Y = dataset[:, 8]    # Sử dụng cột 8 làm đầu ra (output)

# Tạo mô hình Sequential, một kiến trúc mạng thần kinh tuần tự
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))  # Lớp ẩn với 12 đơn vị và hàm kích hoạt ReLU
model.add(Dense(8, activation='relu'))              # Lớp ẩn với 8 đơn vị và hàm kích hoạt ReLU
model.add(Dense(1, activation='sigmoid'))           # Lớp đầu ra với hàm kích hoạt Sigmoid

# Compile mô hình: Chọn hàm mất mát, optimizer và các metric để theo dõi
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Đào tạo mô hình với dữ liệu huấn luyện
history = model.fit(X, Y, validation_split=0.33, epochs=150, batch_size=10, verbose=0)

# In ra tất cả các keys trong history (lịch sử đào tạo)
print(history.history.keys())

# Vẽ biểu đồ cho độ chính xác (accuracy) trên tập huấn luyện và tập kiểm tra
plt.plot(history.history['accuracy'])           # Độ chính xác trên tập huấn luyện
plt.plot(history.history['val_accuracy'])       # Độ chính xác trên tập kiểm tra (validation set)
plt.title('model accuracy')                     # Đặt tiêu đề biểu đồ
plt.ylabel('accuracy')                          # Đặt nhãn trục y
plt.xlabel('epoch')                             # Đặt nhãn trục x
plt.legend(['train', 'test'], loc='upper left') # Hiển thị chú thích (legend)
plt.show()                                      # Hiển thị biểu đồ

# Vẽ biểu đồ cho hàm mất mát (loss) trên tập huấn luyện và tập kiểm tra
plt.plot(history.history['loss'])               # Hàm mất mát trên tập huấn luyện
plt.plot(history.history['val_loss'])           # Hàm mất mát trên tập kiểm tra (validation set)
plt.title('model loss')                         # Đặt tiêu đề biểu đồ
plt.ylabel('loss')                              # Đặt nhãn trục y
plt.xlabel('epoch')                             # Đặt nhãn trục x
plt.legend(['train', 'test'], loc='upper left') # Hiển thị chú thích (legend)
plt.show()                                      # Hiển thị biểu đồ
```

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





❖ Giải thích :

- Import các thư viện và tải dữ liệu: Đầu tiên, các thư viện như Keras, Matplotlib và NumPy được import. Sau đó, dữ liệu từ tệp CSV "pima-indians-diabetes.csv" được nạp vào biến dataset. Dữ liệu được chia thành các biến đầu vào X và đầu ra Y.
- Xây dựng mô hình: Một mô hình Sequential được tạo. Mô hình này bao gồm một lớp input với 8 đặc trưng, một lớp ẩn với 12 đơn vị và hàm kích hoạt ReLU, một lớp ẩn khác với 8 đơn vị và hàm kích hoạt ReLU, và cuối cùng là một lớp đầu ra với hàm kích hoạt Sigmoid.

- Compile mô hình: Mô hình được biên soạn bằng cách chọn hàm mất mát (binary_crossentropy), optimizer (adam), và các metric để theo dõi (accuracy).
- Đào tạo mô hình: Mô hình được đào tạo trên dữ liệu X và Y với việc sử dụng 33% dữ liệu cho kiểm tra, 150 epochs (vòng lặp), và batch size là 10. Biến history lưu trữ lịch sử quá trình đào tạo.
- In ra các keys trong history: In ra các thông số được lưu trong lịch sử đào tạo, bao gồm accuracy và loss trên tập huấn luyện và tập kiểm tra.
- Vẽ biểu đồ accuracy và loss: Sử dụng Matplotlib, biểu đồ cho độ chính xác trên tập huấn luyện và tập kiểm tra cùng với biểu đồ cho hàm mất mát trên tập huấn luyện và tập kiểm tra được vẽ và hiển thị. Biểu đồ này giúp bạn theo dõi hiệu suất của mô hình qua từng epoch.

Câu 5.2: Xử lý ngôn ngữ tự nhiên

a) Hiểu biết về xử lý ngôn ngữ tự nhiên

Word Embeddings in NLP

Nhúng từ là gì?

Đó là một cách tiếp cận để biểu diễn các từ và tài liệu. Nhúng từ hoặc Vector từ là đầu vào vectơ số đại diện cho một từ trong không gian có chiều thấp hơn. Nó cho phép các từ có nghĩa tương tự có cách biểu diễn tương tự. Họ cũng có thể gần đúng ý nghĩa. Một vectơ từ có 50 giá trị có thể biểu thị 50 đặc điểm riêng biệt.

Đặc điểm: Bất cứ điều gì liên quan đến các từ với nhau. Ví dụ: Tuổi, Thể thao, Thể lực, Việc làm, v.v. Mỗi vectơ từ có các giá trị tương ứng với các đặc điểm này.

Mục tiêu của việc nhúng từ

- Để giảm kích thước
- Dùng một từ để dự đoán những từ xung quanh nó
- Ngữ nghĩa giữa các từ phải được nắm bắt

Tính năng nhúng từ được sử dụng như thế nào?

- Chúng được sử dụng làm đầu vào cho các mô hình học máy.
  Lấy các từ —-> Cho biểu diễn số của chúng —-> Sử dụng trong huấn luyện hoặc suy luận.
- Để thể hiện hoặc trực quan hóa bất kỳ mô hình sử dụng cơ bản nào trong kho dữ liệu để huấn luyện chúng.

Triển khai tính năng nhúng từ:

Nhúng từ là một phương pháp trích xuất các tính năng ra khỏi văn bản để chúng ta có thể nhập các tính năng đó vào mô hình machine learning để làm việc với dữ liệu văn bản. Họ cố gắng bảo tồn thông tin cú pháp và ngữ nghĩa. Các phương pháp như Mục tiêu của từ(BOW), CountVectorizer và TFIDF dựa vào số từ trong câu nhưng không lưu bất kỳ thông tin cú pháp hoặc ngữ nghĩa nào. Trong các thuật toán này, kích thước của vectơ là số phần tử trong từ vựng. Chúng ta có thể nhận được ma trận thưa nếu hầu hết các phần tử bằng 0. Các vectơ đầu vào lớn sẽ có nghĩa là số lượng trọng số khổng lồ sẽ dẫn đến yêu cầu tính toán cao cho việc đào tạo. Tính năng nhúng từ đưa ra giải pháp cho những vấn đề này.

Hãy lấy một ví dụ để hiểu cách tạo vector từ bằng cách lấy các biểu tượng cảm xúc được sử dụng thường xuyên nhất trong các điều kiện nhất định và chuyển đổi từng biểu tượng cảm xúc thành một vectơ và các điều kiện sẽ là đặc điểm của chúng ta.

Happy	????	????	????
Sad	????	????	????
Excited	????	????	????
Sick	????	????	????

```
The emoji vectors for the emojis will be:
     [happy,sad,excited,sick]
???? =[1,0,1,0]
???? =[0,1,0,1]
???? =[0,0,1,1]
.....
```

Theo cách tương tự, chúng ta có thể tạo vectơ từ cho các từ khác nhau dựa trên các đặc điểm nhất định. Những từ có vectơ tương tự rất có thể có cùng ý nghĩa hoặc được sử dụng để truyền đạt cùng một tình cảm.
Trong bài viết này, chúng ta sẽ thảo luận về hai cách tiếp cận khác nhau để có được Word Embeddings:

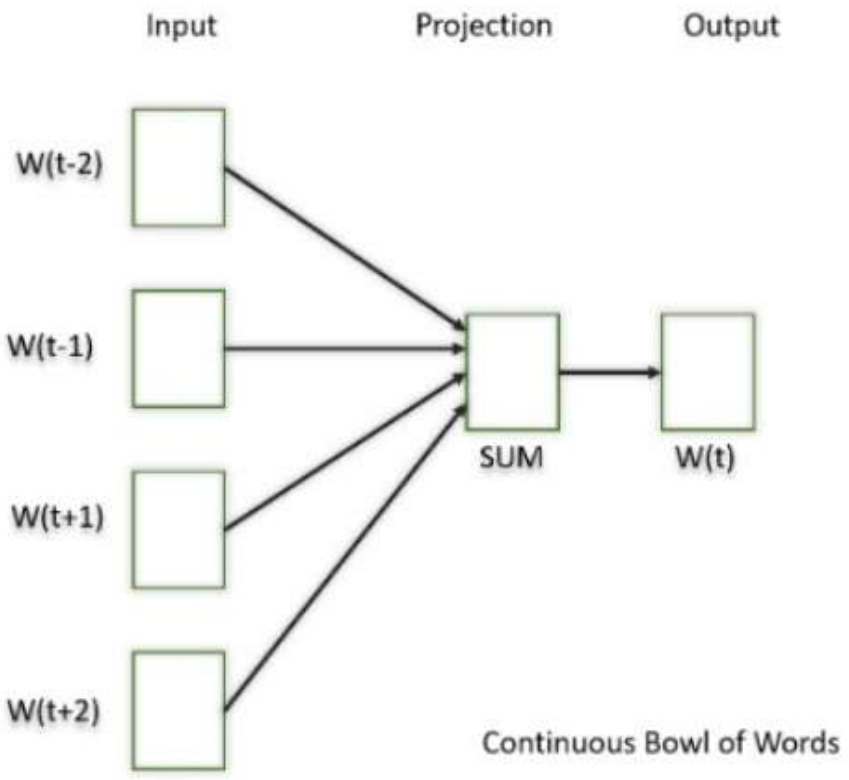
1) Word2Vec:

Trong Word2Vec mỗi từ được gán một vectơ. Chúng ta bắt đầu với một vectơ ngẫu nhiên hoặc **một vectơ nóng** .

Vectơ một nóng: 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:

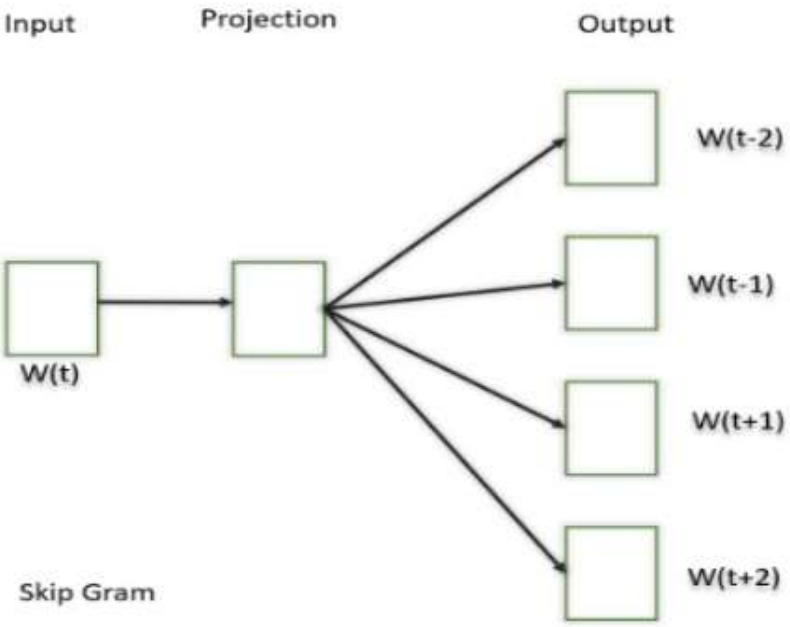
1.1) Túi từ liên tục (CBOW)

Trong mô hình này, điều chúng tôi làm là cố gắng khớp các từ lân cận trong cửa sổ với từ trung tâm.



1.2) Bỏ qua gram

Trong mô hình này, chúng tôi cố gắng làm cho từ trung tâm gần hơn với các từ lân cận. Nó hoàn toàn trái ngược với mô hình CBOW. Nó cho thấy rằng phương pháp này tạo ra các phần nhúng có ý nghĩa hơn.



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.

2) GloVe:

Đây là một phương pháp khác để tạo từ nhúng. Trong phương pháp này, chúng tôi lấy kho văn bản và lặp qua nó và nhận được sự xuất hiện đồng thời của mỗi từ với các từ khác trong 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:
It is a nice evening.
Good Evening!
Is it a nice evening?
```

	it	is	a	nice	evening	good
it	0					
is	1+1	0				
a	1/2+1	1+1/2	0			
nice	1/3+1/2	1/2+1/3	1+1	0		
evening	1/4+1/3	1/3+1/4	1/2+1/2	1+1	0	
good	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 Word:

- 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:

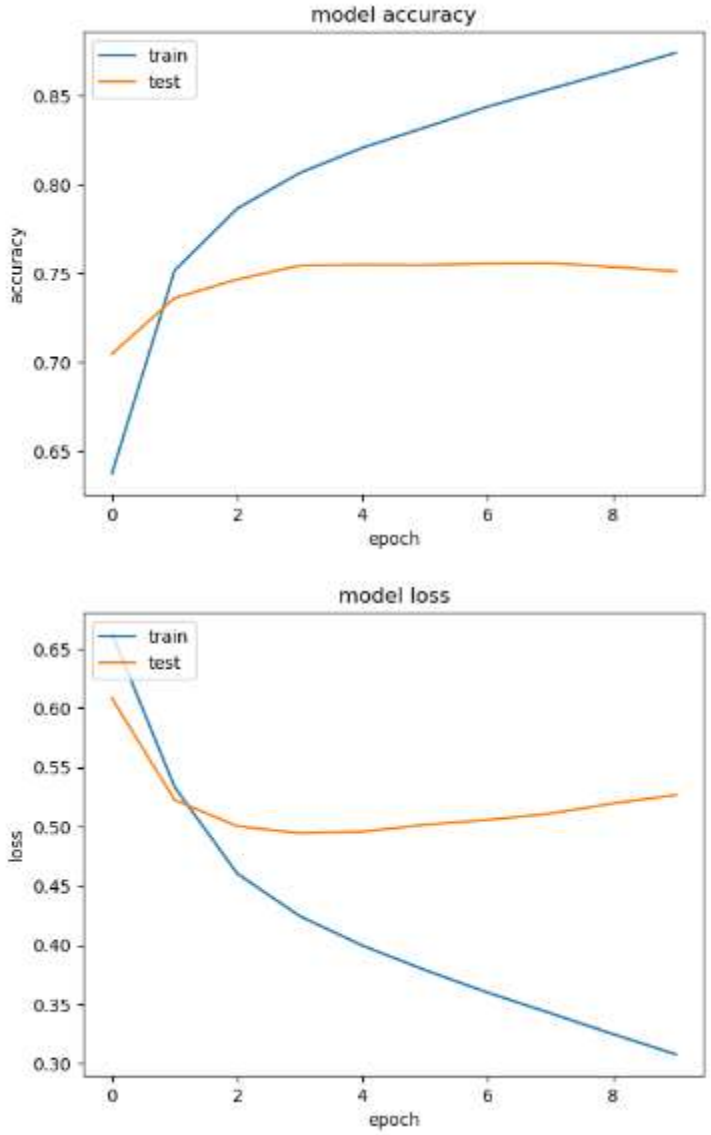
- Nó có thể tốn nhiều bộ nhớ
- Nó phụ thuộc vào ngữ liệu. Bất kỳ sự thiên vị cơ bản nào cũng sẽ có ảnh hưởng đến mô hình của bạn

b) Chạy code

5.2

```
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

# Định nghĩa số lượng từ vựng tối đa và độ dài tối đa của mỗi đánh giá phim
max_features = 10000
maxlen = 20

# Tải dữ liệu IMDB và chia thành tập huấn luyện và tập kiểm tra
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# Chuẩn bị dữ liệu bằng cách đảm bảo rằng mỗi đánh giá phim có cùng độ dài
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)

# Xây dựng mô hình Sequential
model = Sequential()

# Thêm lớp Embedding để biểu diễn từ
model.add(Embedding(10000, 8, input_length=maxlen))

# Thêm lớp Flatten để chuyển đổi dữ liệu thành vector 1D
model.add(Flatten())

# Thêm lớp Dense với hàm kích hoạt sigmoid cho phân loại nhị phân
model.add(Dense(1, activation='sigmoid'))

# Biên soạn mô hình với optimizer, hàm mất mát và metric
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

# In thông tin về kiến trúc của mô hình
model.summary()

# Đào tạo mô hình trên dữ liệu huấn luyện
history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

# Vẽ biểu đồ độ chính xác trên tập huấn luyện và tập kiểm tra qua từng epoch
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()

# Vẽ biểu đồ hàm mất mát trên tập huấn luyện và tập kiểm tra qua từng epoch
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()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 20, 8)             80000

 flatten_1 (Flatten)         (None, 160)               0

 dense_7 (Dense)             (None, 1)                 161

=================================================================
Total params: 80161 (313.13 KB)
Trainable params: 80161 (313.13 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
625/625 [==============================] - 2s 2ms/step - loss: 0.6630 - acc: 0.6377 - val_loss: 0.6084 - val_acc: 0.7048
Epoch 2/10
625/625 [==============================] - 1s 2ms/step - loss: 0.5335 - acc: 0.7516 - val_loss: 0.5225 - val_acc: 0.7362
Epoch 3/10
625/625 [==============================] - 2s 3ms/step - loss: 0.4602 - acc: 0.7865 - val_loss: 0.5004 - val_acc: 0.7466
Epoch 4/10
625/625 [==============================] - 2s 3ms/step - loss: 0.4243 - acc: 0.8065 - val_loss: 0.4944 - val_acc: 0.7544
Epoch 5/10
625/625 [==============================] - 2s 3ms/step - loss: 0.3994 - acc: 0.8206 - val_loss: 0.4957 - val_acc: 0.7548
Epoch 6/10
625/625 [==============================] - 2s 3ms/step - loss: 0.3787 - acc: 0.8320 - val_loss: 0.5016 - val_acc: 0.7546
Epoch 7/10
625/625 [==============================] - 2s 3ms/step - loss: 0.3597 - acc: 0.8437 - val_loss: 0.5056 - val_acc: 0.7556
Epoch 8/10
625/625 [==============================] - 2s 2ms/step - loss: 0.3423 - acc: 0.8536 - val_loss: 0.5110 - val_acc: 0.7558
Epoch 9/10
625/625 [==============================] - 2s 2ms/step - loss: 0.3246 - acc: 0.8636 - val_loss: 0.5196 - val_acc: 0.7536
Epoch 10/10
625/625 [==============================] - 2s 2ms/step - loss: 0.3075 - acc: 0.8740 - val_loss: 0.5266 - val_acc: 0.7512
```



❖ Giải thích:

- Import các thư viện cần thiết và định nghĩa max_features (số từ vựng tối đa) và maxlen (độ dài tối đa của đánh giá phim).
- Tải dữ liệu IMDB và chia thành tập huấn luyện và tập kiểm tra.
- Chuẩn bị dữ liệu bằng cách đảm bảo rằng mỗi đánh giá phim có cùng độ dài.
- Xây dựng mô hình Sequential với lớp Embedding để biểu diễn từ, lớp Flatten để chuyển đổi dữ liệu thành vector 1D, và lớp Dense với hàm kích hoạt sigmoid cho phân loại nhị phân.
- Biên soạn mô hình với optimizer, hàm mất mát và metric, và in thông tin về kiến trúc của mô hình.
- Đào tạo mô hình trên dữ liệu huấn luyện.

- Vẽ biểu đồ độ chính xác trên tập huấn luyện và tập kiểm tra qua từng epoch.
- Vẽ biểu đồ hàm mất mát trên tập huấn luyện và tập kiểm tra qua từng epoch.