Họ và tên: Quách Xuân Phúc

MSV: B20DCCN513

5.1

a)

* 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 lệnh gọi lại Lịch sử. 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())
```

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

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

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

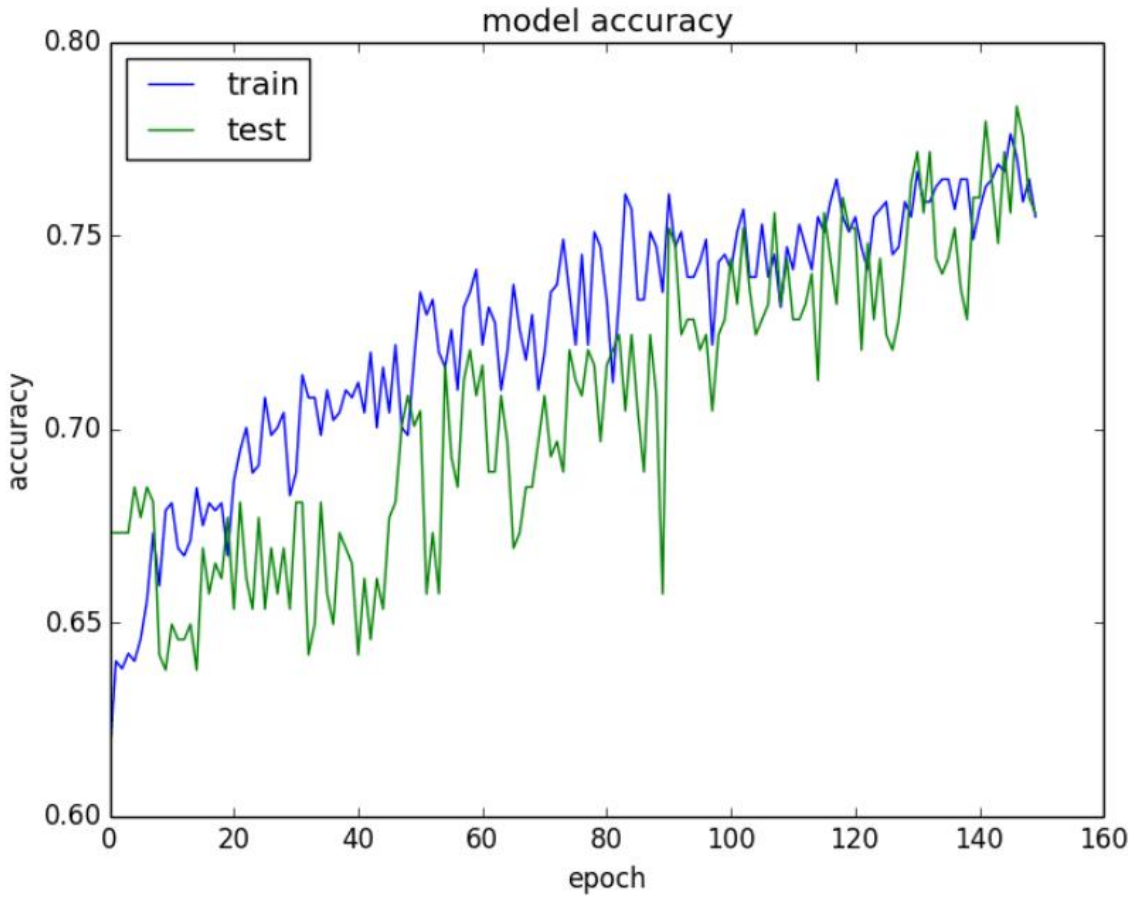
Ví dụ thu thập lịch sử được trả về từ quá trình đào tạo mô hình và tạo hai biểu đồ:

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

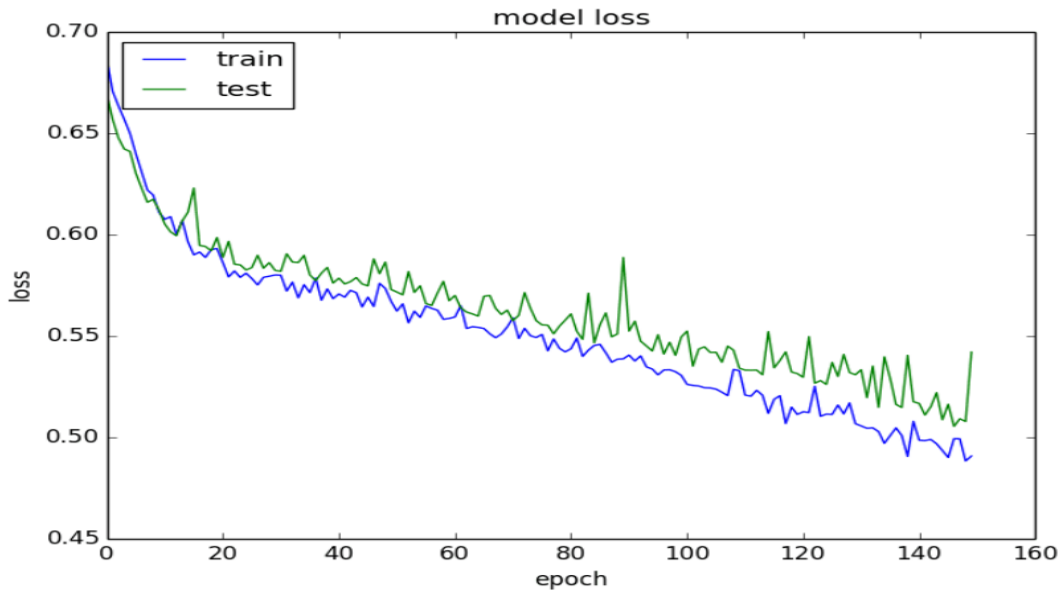
```
# Visualize training history
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
import numpy as np
# load pima indians dataset
dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter=",")
# split into input (X) and output (Y) 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())
# 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()
```

Các lô đất được cung cấp dưới đây. 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.

Từ biểu đồ về độ chính xác, bạn có thể thấy rằng mô hình có thể được đào tạo thêm một chút vì xu hướng về độ chính xác trên cả hai tập dữ liệu vẫn đang tăng lên trong vài kỷ nguyên gần đây. Bạn cũng có thể thấy rằng mô hình vẫn chưa học quá nhiều về tập dữ liệu huấn luyện, thể hiện kỹ năng tương đương trên cả hai tập dữ liệu.



Từ biểu đồ tổn thất, bạn có thể thấy rằng mô hình có hiệu suất tương đương trên cả tập dữ liệu huấn luyện và tập dữ liệu xác thực (kiểm tra được gắn nhãn). Nếu những âm mưu song song này bắt đầu khởi hành một cách nhất quán, đó có thể là dấu hiệu để ngừng đào tạo ở thời điểm sớm hơn.



* Tổng kết:

Hiển thị quá trình học là một phần quan trọng của việc đào tạo mô hình, giúp bạn theo dõi và đánh giá hiệu suất của mô hình trong suốt quá trình đào tạo. Dưới đây là một số cách thông thường để hiển thị quá trình học trong Keras:

1.Sử dụng Callbacks: Keras cung cấp một loạt các "callbacks" mà bạn có thể sử dụng để tùy chỉnh việc hiển thị quá trình học. Một số callbacks phổ biến bao gồm:

- ModelCheckpoint: Lưu mô hình tốt nhất dựa trên tiêu chí quyết định.
- EarlyStopping: Dừng đào tạo nếu không có cải thiện nào được quan sát trong quá trình học.
- TensorBoard: Tạo log để có thể sử dụng TensorBoard để theo dõi tiến trình học.

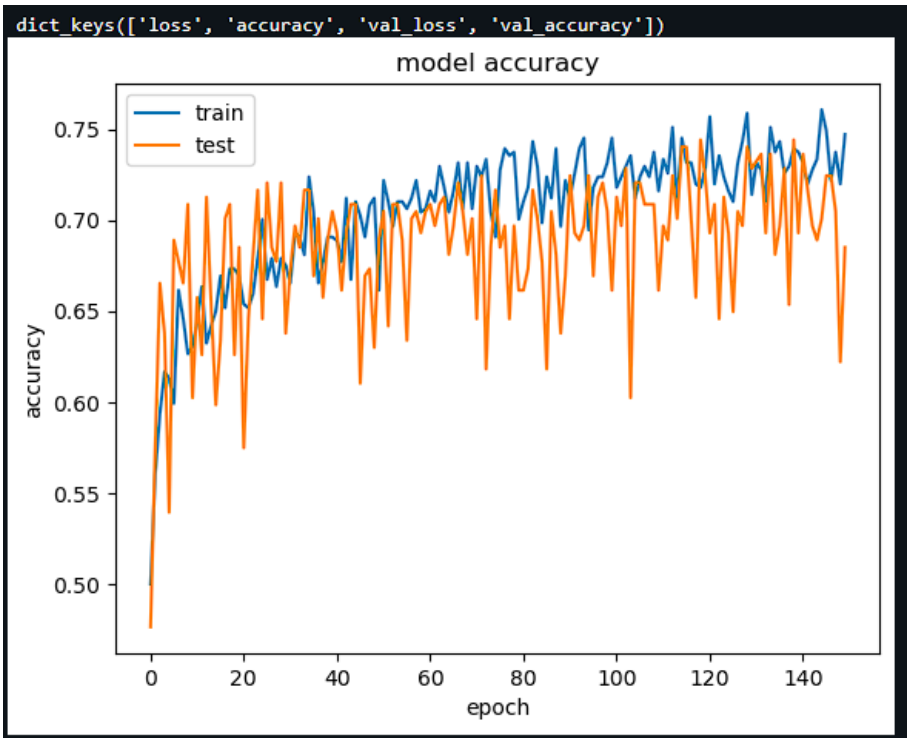
2. Sử dụng fit() method: Khi bạn đào tạo một mô hình bằng cách sử dụng model.fit(), bạn có thể truyền các tham số để kiểm soát việc hiển thị quá trình học.

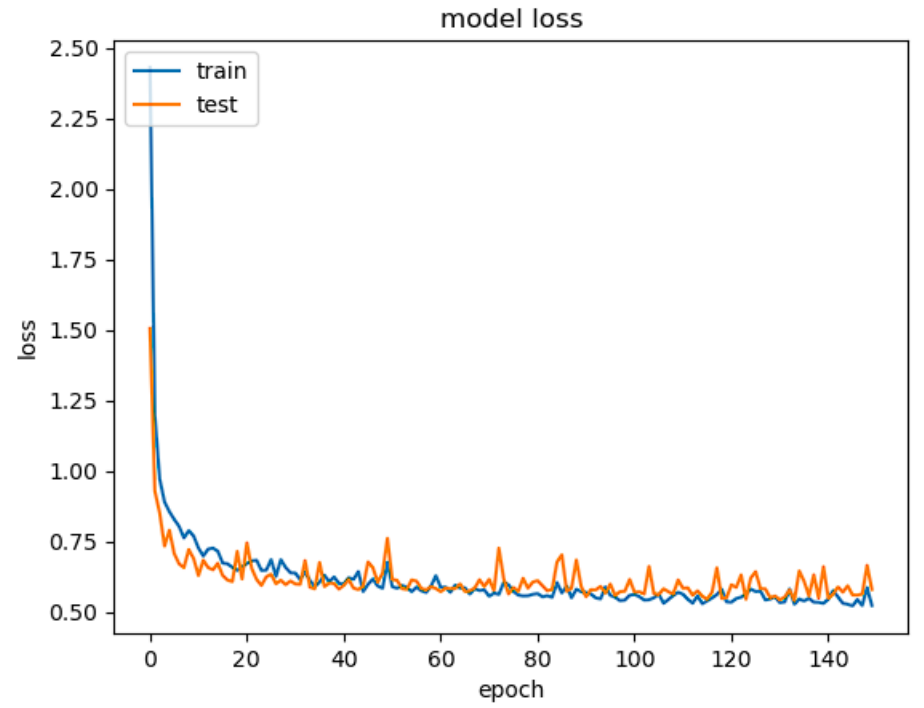
3. Sử dụng TensorBoard: TensorBoard là một công cụ mạnh mẽ được cung cấp bởi TensorFlow (cũng được tích hợp với Keras) để theo dõi quá trình học. Bằng cách sử dụng TensorBoard, bạn có thể xem biểu đồ về hàm mất mát, độ chính xác, và các thông số khác trong thời gian thực.

b)

```
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())

# 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()
```

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





* Giải thích:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

- Nhập các lớp và mô hình liên quan từ thư viện Keras, một giao diện cao cấp cho TensorFlow để xây dựng và huấn luyện mạng nơ-ron.

```
import matplotlib.pyplot as plt
```

- Nhập thư viện để vẽ biểu đồ đánh giá kết quả của mô hình.

```
import numpy as np
```

- Nhập thư viện numpy để xử lý dữ liệu số học và mảng.

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

- Nạp dữ liệu từ tập tin "pima-indians-diabetes.csv" vào một mảng numpy.

```
# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]
```

- Dữ liệu được chia thành hai phần X chứa các đặc trưng đầu vào (các cột từ 0 đến 7) và Y chứa nhãn đầu ra (cột thứ 8).

```
# create model
model = Sequential()
```

- Tạo một mô hình mạng nơ-ron tuần tự bằng cách sử dụng Sequential API của Keras.

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

- Thêm một lớp Dense với 12 đơn vị nơ-ron, là lớp đầu tiên của mô hình. Hàm kích hoạt được sử dụng là 'relu'.

```
model.add(Dense(8, activation='relu'))
```

- Thêm một lớp Dense với 8 đơn vị nơ-ron và hàm kích hoạt 'relu'.

```
model.add(Dense(1, activation='sigmoid'))
```

- Thêm một lớp Dense với 1 đơn vị nơ-ron (được sử dụng cho bài toán phân loại nhị phân) và hàm kích hoạt 'sigmoid'.

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

- Đây là dòng code để biên dịch mô hình. Hàm mất mát được sử dụng là 'binary_crossentropy', trình tối ưu hóa là 'adam', và độ đo đánh giá là 'accuracy'.

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

- Đây là dòng code để huấn luyện mô hình trên dữ liệu đào tạo. Mô hình sẽ được đào tạo trong 150 epoch, với kích thước mẫu (batch_size) là 10 và sẽ sử dụng 33% dữ liệu cho kiểm tra (validation_split=0.33). Biến history lưu trữ các thông tin về quá trình đào tạo.

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

- In ra các khóa trong dictionary history.history, chứa thông tin về mất mát và độ chính xác trong quá trình đào tạo và kiểm tra.

```
# 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()
```

- Vẽ biểu đồ độ chính xác của mô hình trên tập đào tạo và tập kiểm tra qua các epoch.

```
# 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()
```

- Vẽ biểu đồ mất mát của mô hình trên tập đào tạo và tập kiểm tra qua các epoch.

5.2

a)

Xử lý ngôn ngữ tự nhiên (NLP) là một lĩnh vực trong trí tuệ nhân tạo (AI) mà mục tiêu chính là hiểu và xử lý ngôn ngữ con người bằng cách sử dụng máy tính. Một phần quan trọng của NLP là biểu diễn từ và văn bản dưới dạng số học để máy tính có thể hiểu được.

Trong NLP, Word Embeddings (nhúng từ) là một khái niệm quan trọng và mạnh mẽ để biểu diễn từ và văn bản bằng các vectơ số học có chiều thấp. Dưới đây là một số khía cạnh quan trọng về Word Embeddings trong NLP:

1. Mục đích của Word Embeddings:

- Biểu diễn từ và văn bản bằng vectơ số học: Word embeddings giúp chuyển từ và văn bản thành dạng số học, giúp máy tính hiểu được chúng và thực hiện các nhiệm vụ phức tạp như phân loại văn bản, dự đoán từ tiếp theo trong một chuỗi văn bản, dịch máy, và nhiều ứng dụng khác.

- Giảm chiều dữ liệu: Word embeddings giảm chiều dữ liệu từ không gian từ vựng lớn xuống các vectơ số chiều thấp. Điều này giúp cải thiện hiệu suất mô hình và giảm độ phức tạp tính toán.

2. Cách tạo Word Embeddings:

- Word2Vec: Word2Vec là một trong những phương pháp phổ biến nhất để tạo word embeddings. Nó dựa trên ý tưởng rằng các từ xuất hiện cùng nhau trong các ngữ cảnh tương tự sẽ có những vectơ nhúng tương tự. Có hai phương pháp chính trong Word2Vec là Skip-gram và Continuous Bag of Words (CBOW).

- GloVe (Global Vectors for Word Representation): GloVe là một phương pháp khác để tạo word embeddings dựa trên việc xác định tần suất xuất hiện của các từ và cách chúng tương tác với nhau trong toàn bộ tập dữ liệu.

3. Các đặc điểm của Word Embeddings:

- Tính tương tự ngữ nghĩa: Các vectơ word embeddings được thiết kế sao cho các từ có ý nghĩa tương tự sẽ có các vectơ gần nhau trong không gian số học.

- Tính đại diện: Mỗi từ được biểu diễn bằng một vectơ số cố định có thể sử dụng trong các mô hình NLP khác nhau.

- Khả năng tự động học: Word embeddings có thể được học tự động từ dữ liệu lớn, không cần phải thiết lập thủ công các quy tắc ngữ nghĩa.

4. Ứng dụng của Word Embeddings:

- Phân loại văn bản: Word embeddings thường được sử dụng trong các mô hình phân loại văn bản để cải thiện hiệu suất.

- Dự đoán từ tiếp theo: Trong mô hình ngôn ngữ tự nhiên, word embeddings có thể được sử dụng để dự đoán từ tiếp theo trong một chuỗi văn bản.

- Dịch máy: Word embeddings cũng được sử dụng trong các mô hình dịch máy để cải thiện chất lượng dịch.

- Phân tích ý kiến: Trong phân tích ý kiến, word embeddings có thể giúp xác định cảm xúc và ý kiến trong văn bản.

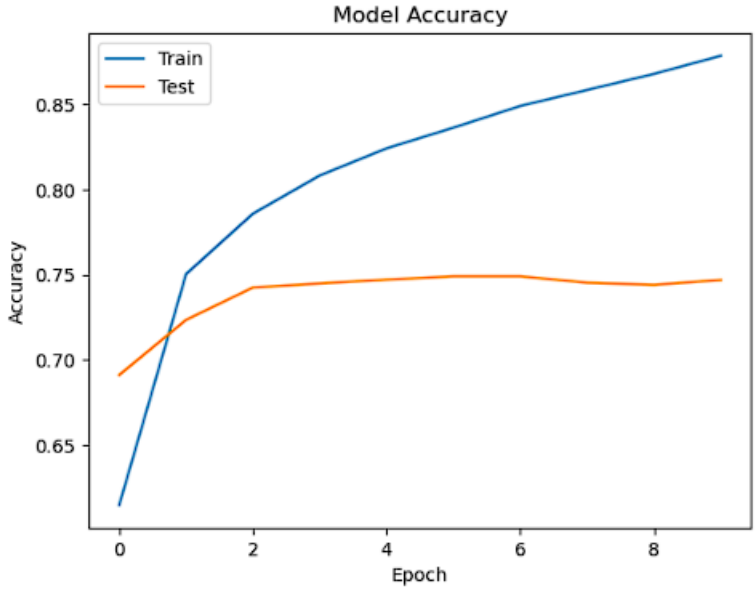
5. Pre-trained Word Embeddings: Có sẵn các bộ word embeddings được huấn luyện trước trên các tập dữ liệu lớn như Word2Vec, GloVe và BERT embeddings. Các pre-trained embeddings có thể được sử dụng trong các ứng dụng NLP để tiết kiệm thời gian và tài nguyên.

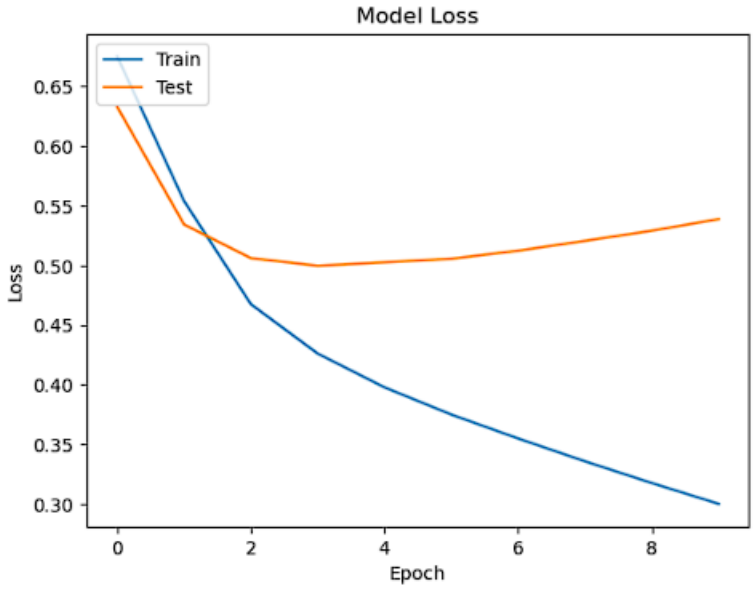
Word embeddings là một công cụ quan trọng trong NLP, giúp máy tính hiểu và xử lý ngôn ngữ con người một cách hiệu quả và mạnh mẽ.

b)

```python
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense
import matplotlib.pyplot as plt

# Thiết lập các thông số
max_features = 10000
maxlen = 20

# Tải dữ liệu IMDb
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# Tiền xử lý dữ liệu
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)

# Xây dựng mô hình
model = Sequential()
model.add(Embedding(10000, 8, input_length=maxlen))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

# Biên dịch mô hình
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

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

# Huấn luyện mô hình và lưu thông tin đào tạo vào biến history
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)

# Vẽ biểu đồ độ chính xác
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()

# Vẽ biểu đồ 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()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 11s 1us/step
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 20, 8)             80000

 flatten (Flatten)           (None, 160)               0

 dense_3 (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 [==============================] - 3s 3ms/step - loss: 0.6751 - accuracy: 0.6145 - val_loss: 0.6329 - val_accuracy: 0.6910
Epoch 2/10
625/625 [==============================] - 2s 3ms/step - loss: 0.5540 - accuracy: 0.7503 - val_loss: 0.5343 - val_accuracy: 0.7234
Epoch 3/10
625/625 [==============================] - 2s 3ms/step - loss: 0.4675 - accuracy: 0.7858 - val_loss: 0.5061 - val_accuracy: 0.7424
Epoch 4/10
625/625 [==============================] - 2s 4ms/step - loss: 0.4261 - accuracy: 0.8083 - val_loss: 0.4997 - val_accuracy: 0.7448
Epoch 5/10
625/625 [==============================] - 2s 3ms/step - loss: 0.3978 - accuracy: 0.8242 - val_loss: 0.5026 - val_accuracy: 0.7470
Epoch 6/10
625/625 [==============================] - 3s 4ms/step - loss: 0.3750 - accuracy: 0.8364 - val_loss: 0.5056 - val_accuracy: 0.7490
Epoch 7/10
625/625 [==============================] - 2s 4ms/step - loss: 0.3550 - accuracy: 0.8491 - val_loss: 0.5123 - val_accuracy: 0.7490
Epoch 8/10
625/625 [==============================] - 2s 3ms/step - loss: 0.3358 - accuracy: 0.8585 - val_loss: 0.5206 - val_accuracy: 0.7452
Epoch 9/10
625/625 [==============================] - 2s 2ms/step - loss: 0.3176 - accuracy: 0.8680 - val_loss: 0.5291 - val_accuracy: 0.7440
Epoch 10/10
625/625 [==============================] - 1s 2ms/step - loss: 0.3002 - accuracy: 0.8786 - val_loss: 0.5388 - val_accuracy: 0.7468
```





* Giải thích:

```
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense
import matplotlib.pyplot as plt
```

- Nhập các thư viện và module cần thiết. imdb là một tập dữ liệu được sử dụng để phân loại đánh giá phim, sequence là module để tiền xử lý dữ liệu chuỗi, Sequential là mô hình mạng nơ-ron tuần tự, Embedding là lớp nhúng từ, Flatten là lớp để làm phẳng dữ liệu, và Dense là lớp kết nối đầy đủ trong mạng nơ-ron.

```
# Thiết lập các thông số
max_features = 10000
maxlen = 20
```

- Định nghĩa các thông số cho tập dữ liệu. max_features là số lượng từ tối đa để xem xét trong từ điển, và maxlen là độ dài tối đa của các mẫu dữ liệu đầu vào.

```
# Tải dữ liệu IMDb
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
```

- Sử dụng hàm imdb.load_data để tải tập dữ liệu IMDb. num_words được sử dụng để giới hạn số từ tối đa trong từ điển của tập dữ liệu theo max_features. Dữ liệu được chia thành hai tập: tập huấn luyện và tập kiểm tra.

```
# Tiền xử lý dữ liệu
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
```

- Sử dụng sequence.pad_sequences để tiền xử lý dữ liệu. Các chuỗi văn bản được cắt hoặc bổ sung để có độ dài maxlen.

```
# Xây dựng mô hình
model = Sequential()
model.add(Embedding(10000, 8, input_length=maxlen))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
```

- Xây dựng một mô hình mạng nơ-ron tuần tự. Mô hình bao gồm:

- Lớp nhúng từ với kích thước từ điển là 10000, kích thước nhúng là 8, và độ dài đầu vào là maxlen.
- Lớp làm phẳng dữ liệu.
- Lớp kết nối đầy đủ với hàm kích hoạt sigmoid.

```
# Biên dịch mô hình
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
```

- Biên dịch mô hình với trình tối ưu hóa 'rmsprop', hàm mất mát 'binary_crossentropy' (phù hợp cho bài toán phân loại nhị phân), và độ đo đánh giá là 'accuracy'.

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

- Hiển thị tóm tắt thông tin của mô hình, bao gồm các lớp, kích thước đầu vào/đầu ra và số lượng tham số.

```
# Huấn luyện mô hình và lưu thông tin đào tạo vào biến history
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

- Huấn luyện mô hình trên tập dữ liệu huấn luyện với 10 epoch, kích thước mẫu (batch_size) là 32, và sử dụng 20% dữ liệu cho kiểm tra. Thông tin đào tạo được lưu vào biến history.

```
# Vẽ biểu đồ độ chính xác
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()
```

- Vẽ biểu đồ độ chính xác của mô hình trên tập huấn luyện và tập kiểm tra qua các epoch.

```
# Vẽ biểu đồ 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()
```

- Vẽ biểu đồ mất mát của mô hình trên tập huấn luyện và tập kiểm tra qua các epoch.