Nguyễn Tiến Anh - B20DCCN005

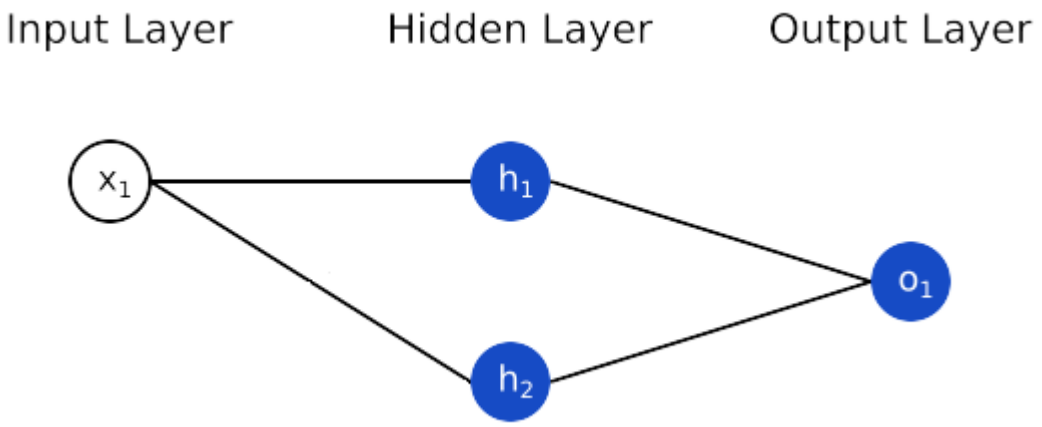
Bài 5: Natural Language Processing

5.1

a. Hiểu biết về keras

- Keras là thư viện mạnh mẽ trong Python cung cấp giao diện cho việc tạo mô hình học máy và rộng hơn kĩ thuật Tensorflow và Theano backends
- Keras là giao diện lập trình ứng dụng (API) neuron network dành cho Python được tích hợp chặt chẽ với TensorFlow, được sử dụng để xây dựng các mô hình học máy. Các mô hình của Keras cung cấp một cách đơn giản, thân thiện với người dùng để xác định neuron network
- Gồm các phần tiêu biểu:
  - **inputs**: Đầu vào của mô hình, có thể là một object keras.Input hoặc là sự kết hợp của nhiều object keras.Input trong một dict, list, tuple.
  - **outputs**: Đầu ra của mô hình, có thể là một object keras.Layer hoặc là sự kết hợp của nhiều object keras.Layer trong một dict, list, tuple.
  - **name**: Tên của mô hình, có thể là một string hoặc là None.
- Sự khác biệt giữa TensorFlow và Keras:
  - TensorFlow là một thư viện mã nguồn mở được sử dụng để thực hiện các tính toán số học sử dụng các biểu đồ luồng dữ liệu. Nó được sử dụng để xây dựng và huấn luyện các mô hình học máy và các mô hình học sâu. Keras là một API mạnh mẽ được sử dụng để giải quyết các vấn đề về học máy. Nó cung cấp một cách đơn giản để xây dựng các mô hình học máy và các mô hình học sâu.
  - Keras là một API cao cấp được sử dụng để xây dựng và huấn luyện các mô hình học máy. Nó cung cấp một cách đơn giản để xây dựng các mô hình học máy và các mô hình học sâu. TensorFlow là một thư viện mã nguồn mở được sử dụng để thực hiện các tính toán số học sử dụng các biểu đồ luồng dữ liệu. Nó được sử dụng để xây dựng và huấn luyện các mô hình học máy và các mô hình học sâu.
- Khi nào sử dụng TensorFlow và Keras:
  - TensorFlow cung cấp một nền tảng học máy toàn diện, cung cấp cả khả năng cấp cao và cấp thấp để xây dựng và triển khai các mô hình học máy. Nó được sử dụng tốt nhất khi có nhu cầu:
    - Nghiên cứu học sâu
    - Neural network phức tạp
    - Làm việc với dữ liệu lớn

        - Cần tốc độ xử lý nhanh
    - Keras hoàn hảo cho những người không có nền tảng vững chắc về Deep Learning nhưng vẫn muốn làm việc với mạng lưới thần kinh. Sử dụng Keras, bạn có thể xây dựng mô hình mạng thần kinh một cách nhanh chóng và dễ dàng bằng cách sử dụng mã tối thiểu, cho phép tạo nguyên mẫu nhanh chóng.
    - Keras ít xảy ra lỗi hơn TensorFlow và các mô hình có nhiều khả năng chính xác hơn với Keras so với TensorFlow. Điều này là do Keras hoạt động trong các giới hạn của khuôn khổ của nó, bao gồm:
        - Tốc độ tính toán: Keras hy sinh tốc độ để thân thiện với người dùng.
        - Lỗi cấp độ thấp: đôi khi bạn sẽ nhận được thông báo lỗi phụ trợ TensorFlow mà Keras không được thiết kế để xử lý.
        - Hỗ trợ thuật toán – Keras không phù hợp để làm việc với một số thuật toán và mô hình học máy cơ bản nhất định như phân cụm và Phân tích thành phần chính (PCM).
        - Biểu đồ động – Keras không hỗ trợ tạo biểu đồ động.
    - Mô hình của Keras:
        - Mô hình là thực thể cốt lõi mà bạn sẽ làm việc khi sử dụng Keras. Các mô hình này được sử dụng để xác định neural network TensorFlow bằng cách chỉ định các thuộc tính, chức năng và lớp muốn xử lý.
        - Keras cung cấp một số API có thể sử dụng để xác định neural network bao gồm:
            - API tuần tự, cho phép tạo từng lớp mô hình cho hầu hết các vấn đề. Nó đơn giản (chỉ là một danh sách các lớp đơn giản), nhưng nó bị giới hạn ở các lớp lớp một đầu vào, một đầu ra.
            - API chức năng, là API đầy đủ tính năng hỗ trợ các kiến trúc mô hình tùy ý. Nó linh hoạt và phức tạp hơn API tuần tự.
            - Phân lớp mô hình, cho phép bạn triển khai mọi thứ từ đầu. Thích hợp cho nghiên cứu và các trường hợp sử dụng có độ phức tạp cao nhưng hiếm khi được sử dụng trong thực tế.
- Các xây dựng neural network bằng API tuần tự của Keras:
    - API tuần tự là một khung để tạo các mô hình dựa trên các phiên bản của lớp tuần tự. Mô hình có một biến đầu vào, một lớp ẩn có hai nơ-ron và một lớp đầu ra có một đầu ra nhị phân. Các lớp bổ sung có thể được tạo và thêm vào mô hình.




```
# Define the model:
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(2, input_dim=1, activation="relu"))
model.add(Dense(1, activation="sigmoid"))
# Print a summary of the created model:
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(2, input_dim=1, activation="relu"))
model.add(Dense(1, activation="sigmoid"))
print(model.summary())
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_16 (Dense)            (None, 2)                 4

 dense_17 (Dense)            (None, 1)                 3

=================================================================
Total params: 7 (28.00 Byte)
Trainable params: 7 (28.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

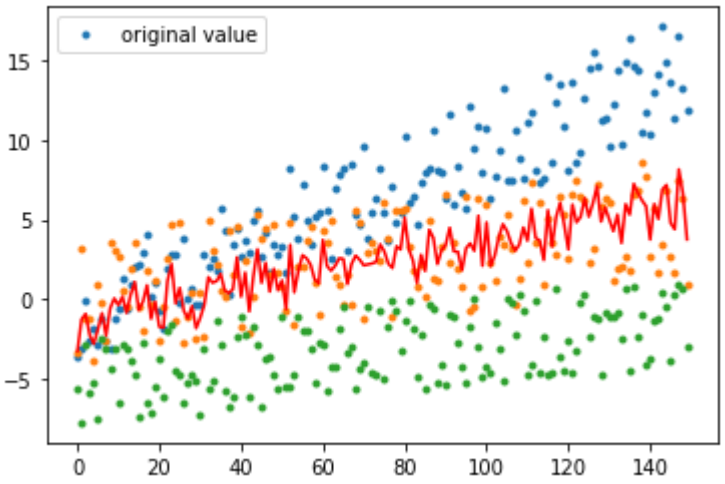
- Cách xây dựng neural network bằng API chức năng của Keras:
  - API chức năng của Keras cho phép:
    - Xây dựng các mô hình có nhiều đầu vào và đầu ra.
    - Xây dựng các mô hình có các lớp chia sẻ.
    - Xây dựng các mô hình không tuần tự (ví dụ: mô hình có các vòng lặp).
  - API chức năng được xây dựng bằng cách tạo ra các phiên của các lớp và kết nối chúng với nhau trực tiếp theo cặp. Điều này cho phép bạn xây dựng các mô hình có các lớp chia sẻ, các mô hình có nhiều đầu vào và đầu ra và các mô hình không tuần tự.

```
# Define the input layer:
from keras.layers import Input

visible = Input(shape=(2,))


# Connect the layers, then create a hidden layer as a Dense
# that receives input only from the input layer:
from keras.layers import Dense

visible = Input(shape=(2,))
hidden = Dense(2)(visible)
```

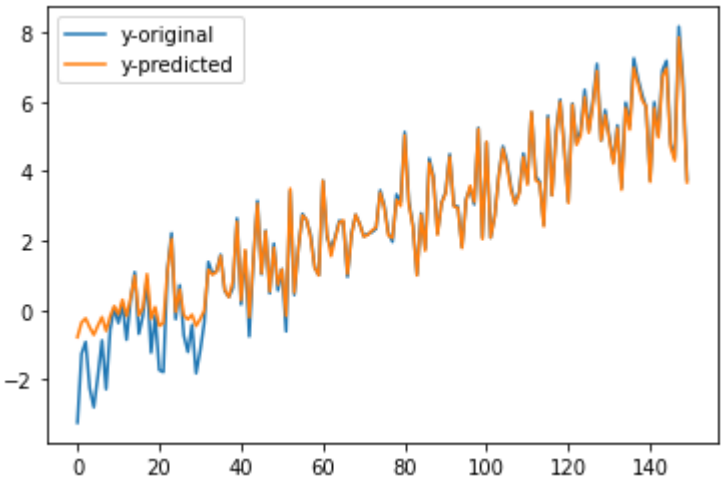
```
# Define a Functional API model:
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense

visible = Input(shape=(2,))
hidden = Dense(2)(visible)
model = Model(inputs=visible, outputs=hidden)
```

- Cách sử dụng Keras để đưa ra dự đoán

```
# PREPARE THE DATA
# Import libraries required in this example:
import random
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.metrics import mean_squared_error

# Generate a sample dataset from random data:
random.seed(123)


def CreateDataset(N):
    a, b, c, y = [], [], [], []
    for i in range(N):
        aa = i / 10 + random.uniform(-4, 3)
        bb = i / 30 + random.uniform(-4, 4)
        cc = i / 40 + random.uniform(-3, 3) - 5
        yy = (aa + bb + cc / 2) / 3
        a.append([aa])
        b.append([bb])
        c.append([cc])
        y.append([yy])
    return np.hstack([a, b, c]), np.array(y)


N = 150
x, y = CreateDataset(N)
x_ax = range(N)
plt.plot(x_ax, x, "o", label="original value", markersize=3)
plt.plot(x_ax, y, lw=1.5, color="red", label="y")
plt.legend(["original value"])
plt.show()
```

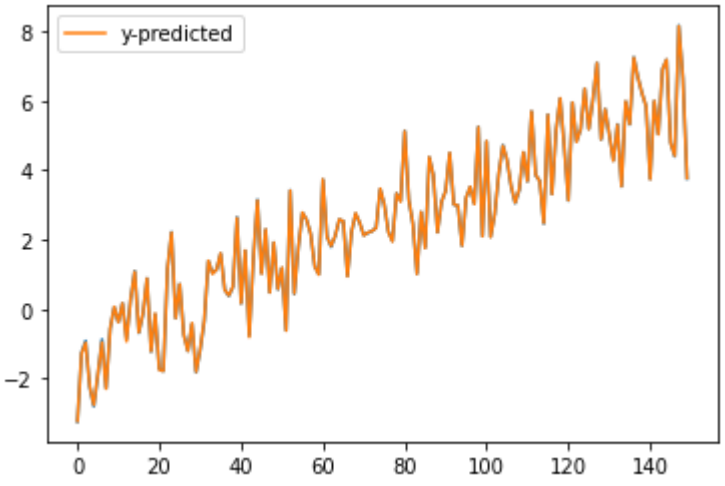



```
# Define and build a Sequential model, and print a summary:
def BuildModel():
    model = Sequential()
    model.add(Dense(128, input_dim=3, activation="relu"))
    model.add(Dense(32, activation="relu"))
    model.add(Dense(8, activation="relu"))
    model.add(Dense(1, activation="linear"))
    model.compile(loss="mean_squared_error", optimizer="adam")
    return model
    BuildModel().summary()


# Fit the Sequential model with Scikit-learn Regressor API for Keras:
regressor = KerasRegressor(build_fn=BuildModel, nb_epoch=100, batch_size=3)
regressor.fit(x, y)
y_pred = regressor.predict(x)
mse_krr = mean_squared_error(y, y_pred)
print(mse_krr)
plt.plot(y, label="y-original")
plt.plot(y_pred, label="y-predicted")
plt.legend()
plt.show()


50/50 [==============================] - 1s 2ms/step - loss: 0.8126
0.21117729273316801
```




```
# Fit the model without the KerasRegressor wrapper:
model = BuildModel()
model.fit(x, y, verbose=False, shuffle=False, epochs=100)
y_krm = model.predict(x)
mse_krm = mean_squared_error(y, y_krm)
print(mse_krm)
plt.plot(y)
plt.plot(y_krm, label="y-predicted")
plt.legend()
plt.show()
```

```
0.00029347680015544194
```

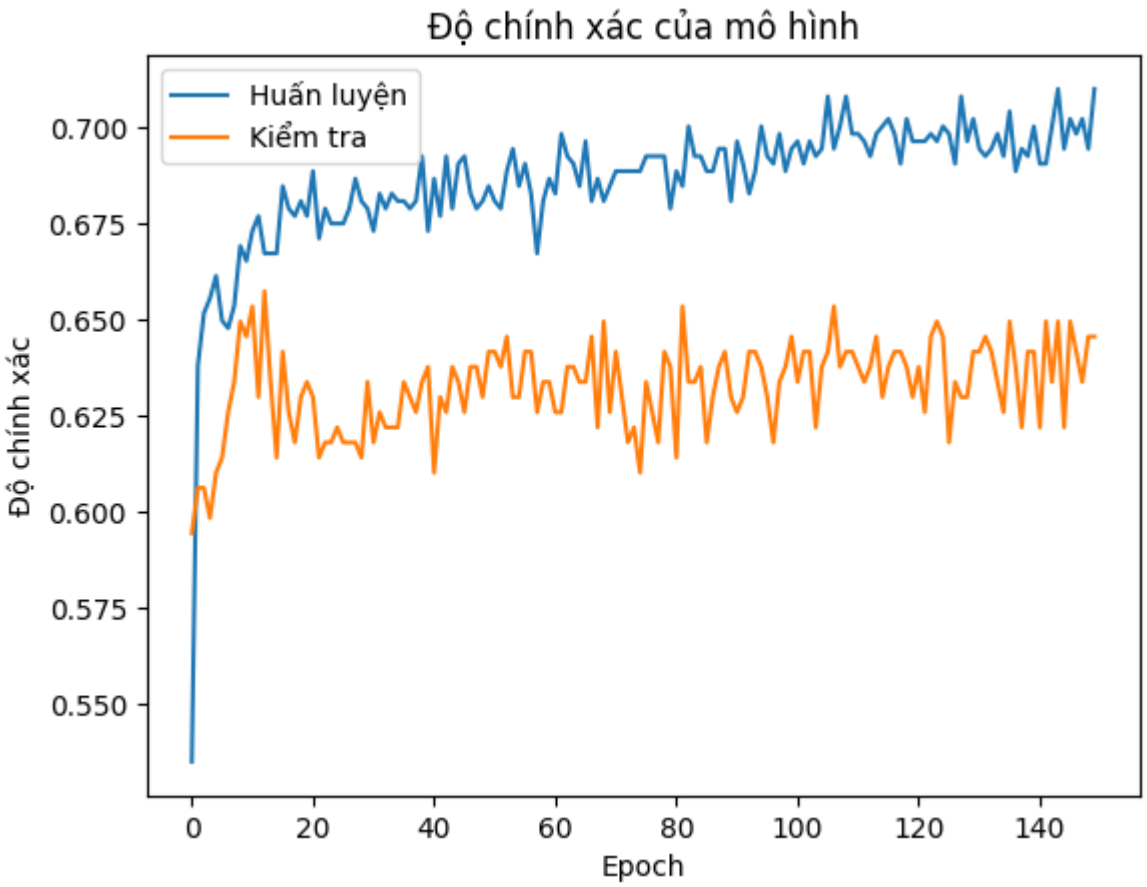


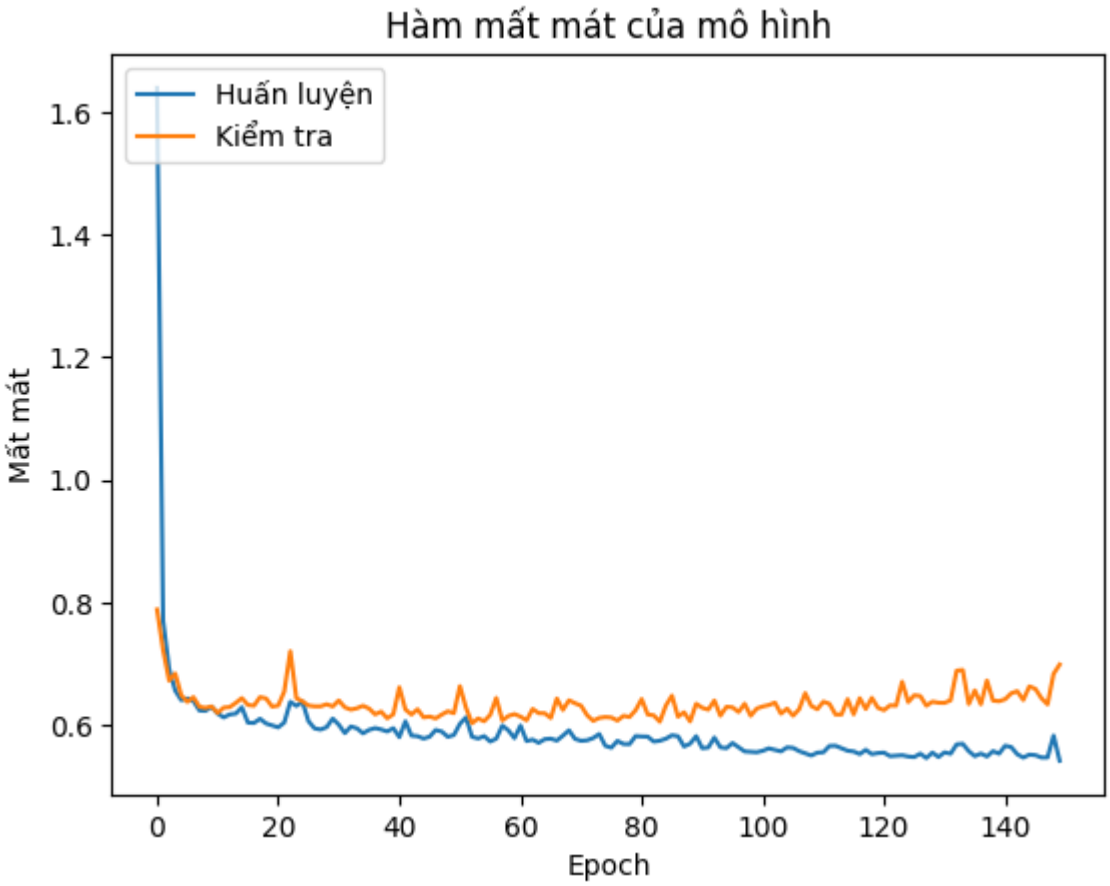
b. Chạy Code

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

# Tải tập dữ liệu Pima Indians từ file csv
dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter=",")

# Chia tập dữ liệu thành biến đầu vào (X) và biến đầu ra (Y)
X = dataset[:, 0:8]
Y = dataset[:, 8]

# 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"))

# Biên soạn mô hình
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])

# Huấn luyện mô hình
history = model.fit(X, Y, validation_split=0.33, epochs=150, batch_size=10, verbose=0)

# Hiển thị các thông tin trong lịch sử huấn luyện
print(history.history.keys())

# Vẽ đồ thị cho độ chính xác của mô hình
plt.plot(history.history["accuracy"])
plt.plot(history.history["val_accuracy"])
plt.title("Độ chính xác của mô hình")
plt.ylabel("Độ chính xác")
plt.xlabel("Epoch")
plt.legend(["Huấn luyện", "Kiểm tra"], loc="upper left")
plt.show()

# Vẽ đồ thị cho hàm mất mát của mô hình
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("Hàm mất mát của mô hình")
plt.ylabel("Mất mát")
plt.xlabel("Epoch")
```

```
plt.legend(["Huấn luyện", "Kiểm tra"], loc="upper left")
plt.show()
```

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





Đoạn mã Python này liên quan đến việc sử dụng thư viện TensorFlow và Keras để xây dựng và huấn luyện một mạng neural để giải quyết một bài toán phân loại nhị phân (binary classification). Dưới đây là giải thích từng phần của đoạn mã:

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
```

   - `tensorflow.keras.models.Sequential` : Sử dụng để xây dựng mô hình mạng neural theo kiểu tuần tự.
   - `tensorflow.keras.layers.Dense` : Được sử dụng để thêm các lớp neural (dense layers) vào mô hình.
   - `matplotlib.pyplot as plt` : Được sử dụng để vẽ đồ thị.
   - `numpy as np` : Được sử dụng để làm việc với mảng và dữ liệu số học.

2. **Tải dữ liệu từ file CSV**:

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

- Dữ liệu được tải từ file CSV có tên "pima-indians-diabetes.csv" và được lưu vào biến `dataset`.

3. **Chia dữ liệu thành biến đầu vào và đầu ra**:

```
X = dataset[:, 0:8]
Y = dataset[:, 8]
```

- Dữ liệu đầu vào (features) là các cột từ 0 đến 7 của `dataset`.
- Dữ liệu đầu ra (labels) là cột thứ 8 của `dataset`.

4. **Xây dựng mô hình mạng neural**:

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

- Mô hình là một mạng neural tuần tự gồm 3 lớp.
- Lớp đầu tiên có 12 nơ-ron, hàm kích hoạt là ReLU và là lớp đầu vào với 8 đặc trưng.
- Lớp thứ hai có 8 nơ-ron và hàm kích hoạt là ReLU.
- Lớp cuối cùng có 1 nơ-ron và hàm kích hoạt là sigmoid, thích hợp cho bài toán phân loại nhị phân.

5. **Biên soạn mô hình**:

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

- Mô hình được biên soạn với hàm mất mát là 'binary_crossentropy' (sử dụng cho bài toán phân loại nhị phân), trình tối ưu hóa 'adam', và sử dụng độ đo 'accuracy' để đánh giá hiệu suất mô hình.

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 đầu vào `X` và đầu ra `Y`.
- `validation_split=0.33` chia tập dữ liệu thành tập huấn luyện và tập kiểm tra với tỷ lệ 33% cho tập kiểm tra.
- Mô hình được huấn luyện trong 150 epochs với batch size là 10.

7. **Vẽ đồ thị độ chính xác và hàm mất mát của mô hình**:
   - Đoạn mã sử dụng thư viện `matplotlib.pyplot` để vẽ đồ thị độ chính xác (accuracy) và đồ thị hàm mất mát (loss) của mô hình trên cả tập huấn luyện và tập kiểm tra. Điều này giúp bạn theo dõi hiệu suất của mô hình và kiểm tra xem liệu nó có bị overfitting hay không.

5.2

a. Hiểu biết về NLP

Xử lý ngôn ngữ tự nhiên (NLP - Natural Language Processing) là một lĩnh vực trong trí tuệ nhân tạo (AI) tập trung vào việc xử lý và hiểu các ngôn ngữ tự nhiên được sử dụng bởi con người. NLP cho phép máy tính tương tác và hiểu được ngôn ngữ con người thông qua việc xử lý, phân tích và tạo ra ngôn ngữ tự nhiên.

NLP đã tồn tại hơn 50 năm và bắt nguồn từ lĩnh vực ngôn ngữ học. Nó có nhiều ứng dụng trong thế giới thực trong một số lĩnh vực, bao gồm nghiên cứu y tế, công cụ tìm kiếm và kinh doanh thông minh.

NLP cho phép máy tính hiểu ngôn ngữ tự nhiên như con người. Cho dù ngôn ngữ được nói hay viết, quá trình xử lý ngôn ngữ tự nhiên sử dụng trí thông minh nhân tạo để nhận đầu vào trong thế giới thực, xử lý và hiểu ý nghĩa của nó theo cách mà máy tính có thể hiểu được.

Giống như con người có các cảm biến khác nhau. Chẳng hạn như tai để nghe và mắt để nhìn. Máy tính có chương trình để đọc và micrô để thu âm thanh. Và giống như con người có bộ não để xử lý đầu vào đó, máy tính có một chương trình để xử lý đầu vào tương ứng của chúng. Tại một số thời điểm trong quá trình xử lý, đầu vào được chuyển đổi thành mã mà máy tính có thể hiểu được.
Có hai giai đoạn chính để xử lý ngôn ngữ tự nhiên: Tiền xử lý dữ liệu và phát triển thuật toán.

Tiền xử lý dữ liệu

Tiền xử lý dữ liệu liên quan đến việc chuẩn bị và “làm sạch” dữ liệu văn bản để máy có thể phân tích dữ liệu đó. Quá trình đặt dữ liệu ở dạng có thể sử dụng được và làm nổi bật các tính năng trong văn bản mà thuật toán có thể dùng được. Có một số cách có thể được thực hiện, bao gồm:

- Token hóa: Đây là lúc văn bản được chia thành các đơn vị nhỏ hơn để làm việc.
- Xóa: Đây là khi các từ phổ biến bị xóa khỏi văn bản để các từ duy nhất cung cấp nhiều thông tin nhất về văn bản vẫn còn.
- Từ vựng và từ gốc: Đây là khi các từ được rút gọn về dạng gốc để xử lý.
- Gắn thẻ một phần của bài phát biểu: Đây là khi các từ được đánh dấu dựa trên phần của bài phát biểu. Chẳng hạn như danh từ, động từ và tính từ.
  Phát triển thuật toán

Khi dữ liệu đã được xử lý trước, một thuật toán được phát triển để xử lý nó. Có nhiều thuật toán xử lý ngôn ngữ tự nhiên khác nhau, nhưng có hai loại chính thường được sử dụng:

Hệ thống dựa trên quy tắc: Hệ thống này sử dụng các quy tắc ngôn ngữ được thiết kế cẩn thận. Cách tiếp cận này đã được sử dụng từ rất sớm trong quá trình phát triển xử lý ngôn ngữ tự nhiên và vẫn được sử dụng.

Hệ thống dựa trên máy học: Các thuật toán học máy sử dụng các phương pháp thống kê. Chúng học cách thực hiện các tác vụ dựa trên dữ liệu đào tạo mà chúng được cung cấp và điều chỉnh phương pháp của chúng khi có nhiều dữ liệu hơn được xử lý. Sử dụng kết hợp học máy, học sâu và mạng lưới thần kinh, các thuật toán xử lý ngôn ngữ tự nhiên trau dồi các quy tắc của riêng chúng thông qua quá trình xử lý và học lặp đi lặp lại.

Các doanh nghiệp sử dụng số lượng lớn dữ liệu nặng về văn bản, không có cấu trúc và cần một cách để xử lý dữ liệu đó một cách hiệu quả. Rất nhiều thông tin được tạo trực tuyến và lưu trữ trong cơ sở dữ liệu là ngôn ngữ tự nhiên của con người. Và cho đến gần đây, các doanh nghiệp không thể phân tích dữ liệu này một cách hiệu quả. Đây là nơi xử lý ngôn ngữ tự nhiên hữu ích.

Xử lý ngôn ngữ tự nhiên nắm vai trò rất quan trọng trong việc phân tích đầy đủ dữ liệu văn bản và giọng nói một cách hiệu quả. Công nghệ này có thể xử lý những nét khác biệt trong phương ngữ, tiếng lóng và điểm bất thường về ngữ pháp thường thấy trong các cuộc hội thoại hàng ngày. Các công ty sử dụng công nghệ này cho một số tác vụ tự động, chẳng hạn như:

- Xử lý, phân tích và lưu trữ các tài liệu lớn
- Phân tích phản hồi của khách hàng hoặc bản ghi âm của tổng đài
- Chạy chatbot cho dịch vụ khách hàng tự động
- Trả lời các câu hỏi về người, sự vật, thời gian, địa điểm
- Phân loại và trích xuất văn bản

Các vấn đề chính trong NLP bao gồm:

1. Tokenization: Quá trình chia câu văn thành các thành phần nhỏ hơn như từ, cụm từ hoặc câu.
2. Morphological analysis: Phân tích cấu trúc từ và các thành phần nhỏ hơn của từ, chẳng hạn như hậu tố, tiền tố, nguyên âm và phụ âm.
3. Parsing: Phân tích cấu trúc ngữ pháp của câu để hiểu các mối quan hệ cú pháp giữa các từ và cấu trúc ngữ pháp.
4. Semantic analysis: Phân tích ý nghĩa của câu để hiểu ý nghĩa và ngữ cảnh của từ và câu.
5. Named Entity Recognition (NER): Nhận dạng và phân loại các thực thể được đặt tên trong văn bản như tên riêng, địa chỉ, ngày tháng, v.v.
6. Sentiment analysis: Phân tích cảm xúc và ý kiến trong văn bản để xác định xem nó có tính tích cực, tiêu cực hoặc trung lập.
7. Machine Translation: Dịch văn bản từ một ngôn ngữ này sang ngôn ngữ khác.
8. Question Answering: Trả lời các câu hỏi dựa trên thông tin trong văn bản.

9. Text generation: Tạo ra văn bản tự động, bao gồm viết văn bản, tóm tắt và sáng tác.
   Các phương pháp và công nghệ phổ biến trong NLP bao gồm: mô hình học sâu (deep learning), mạng nơ-ron nhân tạo (artificial neural networks), các thuật toán học máy (machine learning) như Naive Bayes, Support Vector Machines (SVM), Hidden Markov Models (HMM), và các nguồn dữ liệu ngôn ngữ tự nhiên để huấn luyện và xây dựng mô hình.

NLP có nhiều ứng dụng trong thực tế như hỗ trợ dịch thuật tự động, trợ lý ảo, phân tích ý kiến khách hàng, phân loại văn bản, gợi ý từ khóa trong công cụ tìm kiếm, và nhiều ứng dụng khác liên quan đến xử lý và hiểu ngôn ngữ tự nhiên.
Xử lý ngôn ngữ tự nhiên đã phát triển nhanh chóng nhờ sự tiến bộ trong lĩnh vực học sâu (deep learning) và mạng nơ-ron nhân tạo (artificial neural networks). Các mô hình học sâu như mạng nơ-ron biến đổi (transformer networks) đã đạt được những kết quả ấn tượng trong các nhiệm vụ NLP như dịch máy và sinh văn bản tự động.

Một trong những thành tựu đáng chú ý của NLP là mô hình ngôn ngữ BERT (Bidirectional Encoder Representations from Transformers), được đào tạo trên một lượng lớn dữ liệu ngôn ngữ tự nhiên. BERT đã cải thiện đáng kể hiệu suất của nhiều ứng dụng NLP như phân loại văn bản, phát hiện ngôn ngữ gốc, và tạo ra các biểu đồ ý nghĩa từ các câu văn.

Một ứng dụng quan trọng của NLP là trong lĩnh vực trợ lý ảo (virtual assistants) như Siri, Alexa hay Google Assistant. Các trợ lý ảo này sử dụng NLP để hiểu và đáp ứng các câu hỏi và yêu cầu từ người dùng. NLP giúp xác định ý định của người dùng và trích xuất thông tin cần thiết từ câu hỏi để cung cấp câu trả lời chính xác.

Xử lý ngôn ngữ tự nhiên cũng có ứng dụng quan trọng trong phân tích cảm xúc và ý kiến (sentiment analysis). Các công cụ phân tích cảm xúc sử dụng NLP để xác định cảm xúc tích cực, tiêu cực hoặc trung lập trong văn bản, giúp doanh nghiệp đánh giá đánh giá của khách hàng, phản hồi về sản phẩm hoặc dịch vụ.
Trong lĩnh vực tìm kiếm thông tin, NLP được sử dụng để tạo ra các gợi ý từ khóa (keyword suggestion) khi người dùng nhập câu truy vấn vào công cụ tìm kiếm. NLP giúp phân tích và hiểu câu truy vấn, và gợi ý những từ khóa phù hợp để cải thiện kết quả tìm kiếm.

Trích xuất thông tin (information extraction) là một nhiệm vụ quan trọng của NLP, nơi ngôn ngữ tự nhiên được sử dụng để tìm kiếm và trích xuất thông tin cụ thể từ văn bản. Ví dụ, hệ thống NLP có thể được sử dụng để tự động rút trích thông tin từ các hợp đồng hoặc báo cáo tài chính.

Trong lĩnh vực giáo dục, NLP có thể được sử dụng để xây dựng các công cụ học tập tự động. Ví dụ, hệ thống NLP có thể phân tích và đánh giá các bài luận của học sinh, cung cấp phản hồi và đề xuất cách cải thiện viết.

NLP cũng được sử dụng trong phân tích ngôn ngữtự nhiên của xã hội (social media) để phát hiện và phân tích xu hướng, ý kiến và sự lan truyền thông tin trên mạng xã hội. Điều này có thể hỗ trợ trong việc nắm bắt quan điểm của công chúng, phân tích tác động của các sự kiện xã hội và đánh giá tác động của chiến dịch truyền thông.

Một ứng dụng khác của NLP là phân loại và tổ chức thông tin trong các tài liệu và bài viết. Các thuật toán NLP có thể giúp xác định chủ đề, phân loại văn bản vào các danh mục, và tạo ra các tóm tắt ngắn gọn để giúp người dùng nắm bắt thông tin một cách nhanh chóng và hiệu quả.

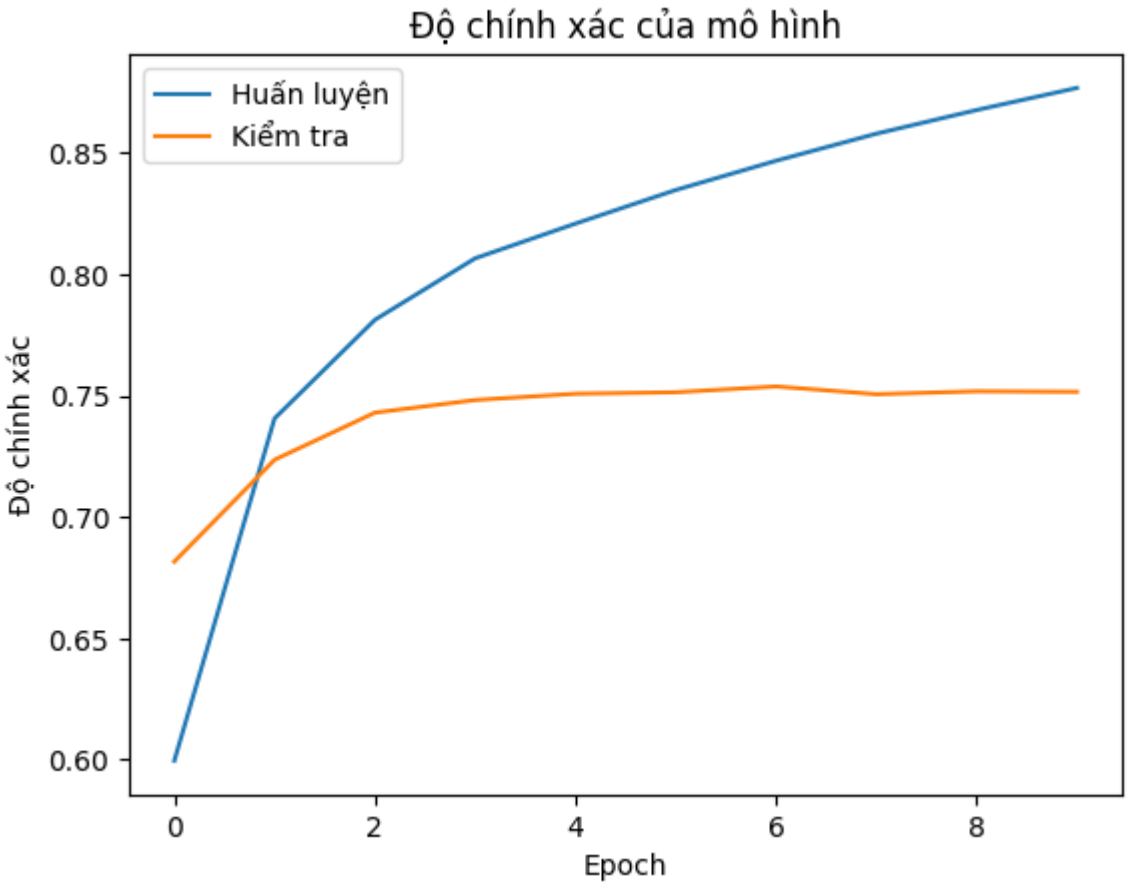
Trong lĩnh vực y tế, NLP có thể được sử dụng để phân tích và trích xuất thông tin từ hồ sơ bệnh án điện tử. Điều này có thể giúp cải thiện quản lý thông tin bệnh án, nghiên cứu y học và phân tích dữ liệu lớn trong lĩnh vực chăm sóc sức khỏe.

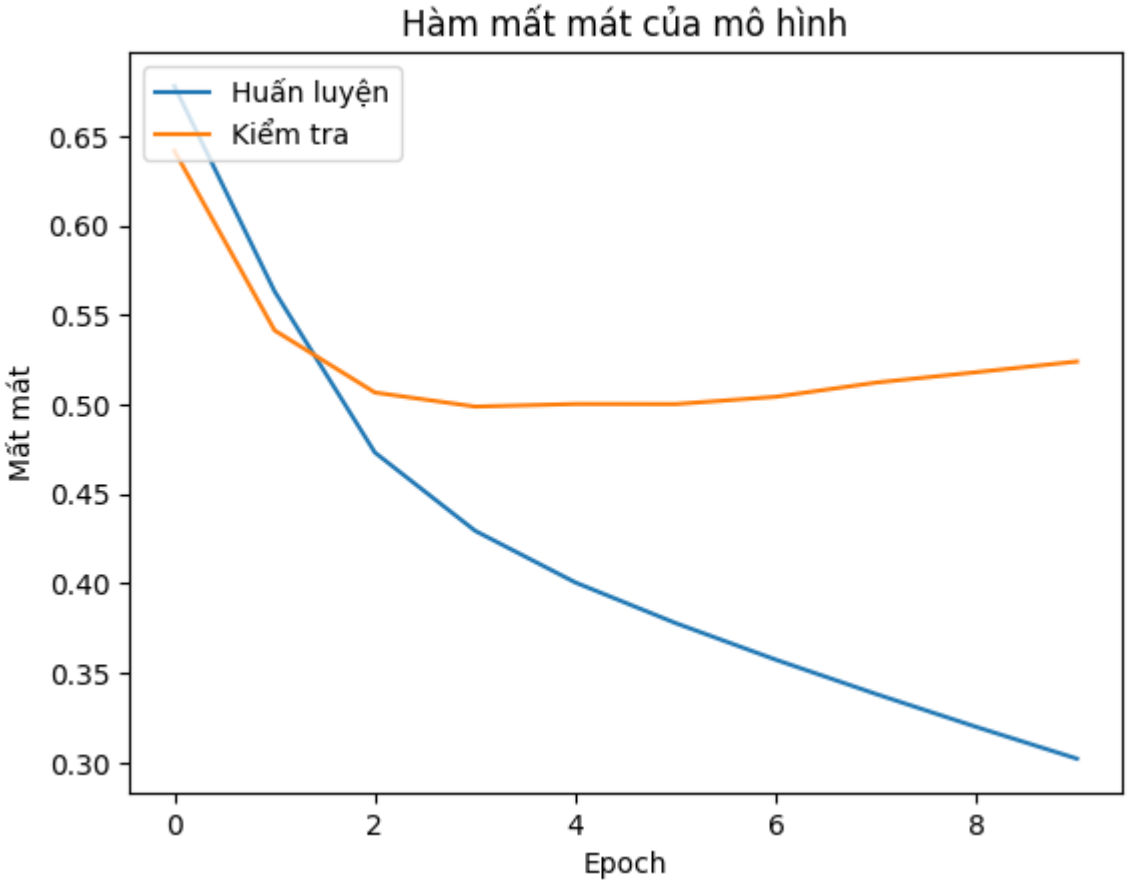
b. Chạy Code

```
# Import các thư viện cần thiết
from keras.datasets import imdb
from keras import preprocessing
from keras.layers import Embedding

# Import các lớp mô hình và lớp Dense từ Keras
from keras.models import Sequential
from keras.layers import Flatten, Dense

# Import thư viện matplotlib để vẽ đồ thị
import matplotlib.pyplot as plt
import numpy as np

# Số lượng từ tối đa trong từ điển (được sử dụng để giới hạn kích thước dữ liệu đầu vào)
max_features = 10000

# Độ dài tối đa của mỗi mẫu dữ liệu đầu vào
maxlen = 20

# Tải dữ liệu huấn luyện và kiểm tra từ tập dữ liệu IMDb với số lượng từ tối đa
# và độ dài tối đa
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# Chuyển đổi dữ liệu đầu vào thành các chuỗi có độ dài cố định (pad_sequences)
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)

# Tạo một mô hình mạng neural tuần tự
model = Sequential()

# Thêm lớp nhúng (embedding layer) với 10,000 từ, mỗi từ có vectơ biểu diễn 8 chiều
# và độ dài đầu vào là maxlen
model.add(Embedding(10000, 8, input_length=maxlen))

# Thêm lớp phẳng (flatten layer) để biến đổi dữ liệu từ ma trận thành vector
model.add(Flatten())

# Thêm lớp Dense với 1 đầu ra và hàm kích hoạt sigmoid (được sử dụng cho bài toán nhị phân)
model.add(Dense(1, activation="sigmoid"))

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

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

# Huấn luyện mô hình trên dữ liệu huấn luyện trong 10 epochs, với batch size là 32
# và tách ra 20% dữ liệu cho kiểm tra
history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

# Vẽ đồ thị độ chính xác trên dữ liệu huấn luyện và dữ liệu kiểm tra
plt.plot(history.history["acc"])
plt.plot(history.history["val_acc"])
plt.title("Độ chính xác của mô hình")
plt.ylabel("Độ chính xác")
plt.xlabel("Epoch")
plt.legend(["Huấn luyện", "Kiểm tra"], loc="upper left")
plt.show()

# Vẽ đồ thị hàm mất mát trên dữ liệu huấn luyện và dữ liệu kiểm tra
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("Hàm mất mát của mô hình")
plt.ylabel("Mất mát")
plt.xlabel("Epoch")
plt.legend(["Huấn luyện", "Kiểm tra"], loc="upper left")
plt.show()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 20, 8)             80000

 flatten_3 (Flatten)         (None, 160)               0

 dense_6 (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 4ms/step - loss: 0.6780 - acc: 0.59
Epoch 2/10
625/625 [==============================] - 2s 3ms/step - loss: 0.5633 - acc: 0.74
...
Epoch 8/10
625/625 [==============================] - 2s 3ms/step - loss: 0.3383 - acc: 0.85
Epoch 9/10
625/625 [==============================] - 2s 3ms/step - loss: 0.3199 - acc: 0.86
Epoch 10/10
625/625 [==============================] - 2s 3ms/step - loss: 0.3022 - acc: 0.87
```





Dưới đây là giải thích từng phần của mã nguồn Python:

1. **Import thư viện và module:** Đầu tiên, chúng ta import các thư viện và module cần thiết để xây dựng và huấn luyện mạng neural:

```
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
```

2. **Định nghĩa các siêu tham số:** Chúng ta định nghĩa một số siêu tham số quan trọng cho việc xử lý dữ liệu và mô hình hóa:

```
max_features = 10000  # Số lượng từ tối đa trong từ điển
maxlen = 20  # Độ dài tối đa của mỗi mẫu dữ liệu đầu vào
```

3. **Tải và xử lý dữ liệu IMDb:** Dữ liệu đánh giá phim từ tập dữ liệu IMDb được tải và chia thành tập huấn luyện và kiểm tra. Nó đã được xử lý để mỗi mẫu có cùng độ dài (maxlen) bằng cách thêm hoặc cắt bớt các từ:

```
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

4. **Xây dựng mô hình mạng neural:** Một mô hình tuần tự (Sequential model) được tạo và các lớp được thêm vào mô hình theo thứ tự:
   - Lớp nhúng (Embedding layer): Lớp này biểu diễn từ vựng dưới dạng các vectơ có kích thước 8 chiều. Đây là lớp đầu tiên trong mạng và nó giúp biểu diễn từng từ dưới dạng các số thực.
   - Lớp phẳng (Flatten layer): Lớp này chuyển từng mẫu dữ liệu từ ma trận thành vector để chuẩn bị cho lớp Dense tiếp theo.
   - Lớp Dense (Dense layer): Lớp này có 1 đầu ra và sử dụng hàm kích hoạt sigmoid để phân loại mỗi đánh giá là tích cực hoặc tiêu cực.

```
model = Sequential()
model.add(Embedding(10000, 8, input_length=maxlen))
model.add(Flatten())
model.add(Dense(1, activation="sigmoid"))
```

5. **Biên dịch mô hình:** Mô hình được biên dịch với các thông số quan trọng như bộ tối ưu hóa ("rmsprop"), hàm mất mát ("binary_crossentropy" - phù hợp cho bài toán phân loại nhị phân) và độ đo đánh giá ("acc" - độ chính xác).

```
model.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=["acc"])
```

6. **Huấn luyện mô hình:** Mô hình được huấn luyện trên dữ liệu huấn luyện trong 10 epochs, với batch size là 32 và sử dụng 20% dữ liệu cho kiểm tra.

```
history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

7. **Vẽ đồ thị độ chính xác và hàm mất mát:** Cuối cùng, chúng ta vẽ đồ thị để theo dõi độ chính xác và hàm mất mát của mô hình trên dữ liệu huấn luyện và kiểm tra qua các epochs.

```
plt.plot(history.history["acc"])
plt.plot(history.history["val_acc"])
plt.title("Độ chính xác của mô hình")
plt.ylabel("Độ chính xác")
plt.xlabel("Epoch")
plt.legend(["Huấn luyện", "Kiểm tra"], loc="upper left")
plt.show()

plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("Hàm mất mát của mô hình")
plt.ylabel("Mất mát")
plt.xlabel("Epoch")
plt.legend(["Huấn luyện", "Kiểm tra"], loc="upper left")
plt.show()
```

Mã này giúp xây dựng một mô hình mạng neural để phân loại đánh giá phim và theo dõi hiệu suất của mô hình qua các epochs.