1. Giải thích các khái niệm sau đây và cho ví dụ code (> 3 trang):

❖ Sample

- Sample là 1 hàng dữ liệu, nó chứa đầu vào để đưa vào thuật toán và đầu ra được sử dụng để so sánh với dự đoán và tính độ chênh lệch.
- Sample có thể được coi là một ví dụ, một quan sát, một vector đầu vào hoặc một vector đặc trưng.
- Vd code:

```
# Một ví dụ dữ liệu hình ảnh
sample_image = [0.1, 0.2, 0.5, 0.9, 0.3, 0.6]

# Một ví dụ dữ liệu văn bản
sample_text = "This is a sample sentence."
```

❖ Iteration

- Iteration là một bước trong quá trình huấn luyện mô hình machine learning. Trong mỗi iteration, mô hình sẽ được đưa một lượng nhất định dữ liệu để tính toán và cập nhật trọng số.
- Vd code:

```
for epoch in range(num_epochs):
    for iteration in range(num_iterations_per_epoch):
        # Tính toán và cập nhật trọng số ở đây
```

❖ Epoch

- Số lượng epoch là một siêu tham số (hyperparameter) xác định số lần thuật toán sẽ thực hiện qua toàn bộ tập dữ liệu huấn luyện.
- Một epoch là một lần duyệt qua toàn bộ tập dữ liệu huấn luyện. Một epoch bao gồm một hoặc nhiều batch.
- Số lượng epoch thông thường lớn, thường trăm hoặc nghìn, cho phép thuật toán chạy cho đến khi độ chênh lệch được giảm xuống thấp nhất.
- Việc tạo biểu đồ cho thấy epoch theo chiều x và độ chênh lệch mô hình theo chiều y rất phổ biến. Biểu đồ trên giúp chuẩn đoạn mô hình có bị

over/under learned hoặc fit với dữ liệu huấn luyện.

- Vd code:

```
num_epochs = 100
for epoch in range(num_epochs):
    # Huấn luyện mô hình trong mỗi epoch
```

❖ Batch

- Batch là một tập hợp các mẫu dữ liệu được sử dụng để tính gradient và cập nhật trọng số của mạng nơ-ron trong mỗi iteration. Batch size là kích thước của tập hợp này.
- Hãy nghĩ về batch như một vòng lặp duyệt qua một hoặc nhiều mẫu dữ liệu và thực hiện dự đoán. Ở cuối batch, các dự đoán được so sánh với giá trị dự kiến và một sai số được tính toán. Từ sai số này, thuật toán cập nhật được sử dụng để cải thiện mô hình.
- Một tập dữ liệu có thể được chia thành 1 hoặc nhiều batches:
  + Batch Gradient Descent. Batch Size = Size of Training Set.
  + Stochastic Gradient Descent. Batch Size = 1.
  + Mini-Batch Gradient Descent. 1 < Batch Size < Size of Training Set

```
batch_size = 5
for epoch in range(num_epochs):
    for iteration in range(num_iterations_per_epoch):
        # Lấy một batch từ tập dữ liệu
        batch_data = get_batch_data(iteration, batch_size)
        # Tính toán và cập nhật trọng số cho batch_data
```

❖ Update weights khi nào?

- Weight là một tham số số học được sử dụng để đánh giá tầm quan trọng của mỗi feature trong mô hình.
- Trọng số của mô hình được cập nhật sau mỗi batch hoặc sau mỗi iteration, tùy thuộc vào thuật toán tối ưu hóa được sử dụng. Trọng số được cập nhật để điều chỉnh mô hình sao cho hàm mất mát (loss function) đạt giá trị tối thiểu.

❖ Ví dụ có data với số lượng 200 mẫu và chọn kích cỡ batch là 5 và chạy với 100 epoch. Hỏi có bao nhiêu iteration, bao nhiêu lần cập nhật trọng số?

- Batch = 200, batch_size = 5, epoch = 100.
- Số lượng iteration = batch / batch_size = 40.
- Số lần cập nhất weight = iteration * epoch = 40 * 100 = 4000

❖ Stochastic Gradient Descent, or SGD là gì?

- Stochastic Gradient Descent, hoặc viết tắt là SGD, là một thuật toán tối ưu hóa được sử dụng để huấn luyện các thuật toán học máy, đặc biệt là các mạng nơ-ron nhân tạo được sử dụng trong học sâu.
- Thuật toán thực hiện tìm nhóm siêu tham số của mô hình mà thực hiện hiệu quả dựa trên các thouật toán kiểm tra như logarithmic loss hoặc mean squared error.
- Thuật toán tối ưu hóa được gọi là "gradient descent", trong đó "gradient" đề cập đến việc tính toán độ dốc của lỗi và "descent" đề cập đến việc di chuyển xuống theo con đường đó đến một mức tối thiểu của lỗi.
- Thuật toán này là một quá trình lặp lại. Điều này có nghĩa là quá trình tìm kiếm xảy ra qua nhiều bước riêng lẻ, mỗi bước hy vọng là cải thiện một chút tham số mô hình.
- Mỗi bước liên quan đến việc sử dụng mô hình với tập hợp hiện tại của các tham số trong mô hình để thực hiện dự đoán trên một số mẫu, so sánh dự đoán với các kết quả thực tế mong đợi, tính toán lỗi và sử dụng lỗi để cập nhật các tham số bên trong mô hình.
- Vd code:

```
from tensorflow.keras.optimizers import SGD

# Khởi tạo optimizer SGD
sgd_optimizer = SGD(learning_rate=0.01)

# Sử dụng SGD trong quá trình huấn luyện mô hình
model.compile(optimizer=sgd_optimizer, loss='mse')
```

2. Stochastic Gradient Descent, or SGD là gì? (>3 trang)

- Gradient Descent:

+ Gradient descent là một thuật toán tối ưu hóa được sử dụng để tìm giá trị của các tham số (hệ số) của một hàm số (f) sao cho giảm thiểu một hàm mất mát (cost) nào đó.
+ Gradient descent thường được sử dụng khi các tham số không thể tính toán bằng phép toán phân tích mà cần được tìm kiếm bằng một thuật toán tối ưu hóa.

– Quá trình Gradient Descent:
  + Quá trình bắt đầu với các giá trị ban đầu cho hệ số hoặc các hệ số của hàm. Các giá trị này có thể là 0.0 hoặc một giá trị ngẫu nhiên nhỏ.

coefficient = 0.0

+ Giá trị của các hệ số được đánh giá bằng cách đưa chúng vào hàm và tính toán giá trị của hàm.

cost = f(coefficient)

+ Sau đó, đạo hàm của hàm mất mát được tính toán. Đạo hàm là một khái niệm từ toán học và liên quan đến độ dốc của hàm tại một điểm cụthể. Chúng ta cần biết độ dốc để biết hướng di chuyển các giá trị hệ số để có được giá trị mất mát thấp hơn trong vòng lặp tiếp theo.

delta = derivative(cost)

+ Bây giờ chúng ta đã biết từ đạo hàm là hướng nào là hướng xuống, chúng ta có thể cập nhật các giá trị hệ số. Một tham số tốc độ học(alpha) phải được chỉ định để kiểm soát mức độ thay đổi của các hệ số trong mỗi lần cập nhật.

coefficient = coefficient – (alpha * delta)

+ Quá trình này được lặp lại cho đến khi giá trị mất mát của các hệ số (cost) là 0.0 hoặc gần đủ gần với 0 để coi đó là đủ tốt.
+ Thuật toán gradient descient rất đơn giản, nó chỉ yêu cầu đạo hàm của hàm mất mát hoặc hàm đang tối ưu.

– Batch Gradient Descent trong học máy:
  + Mục tiêu của tất cả các thuật toán học máy có giám sát là ước tính một hàm mục tiêu (f) tốt nhất mà ánh xạ dữ liệu đầu vào (X) vào các biến đầu ra (Y). Điều này áp dụng cho tất cả các vấn đề phân loại và hồi quy.

+ Một số thuật toán học máy có các hệ số mô tả ước tính của thuật toán cho hàm mục tiêu (f). Các thuật toán khác nhau có biểu diễn khác nhau và các hệ số khác nhau, nhưng hầu hết cần một quá trình tối ưu hóa để tìm ra bộ hệ số dẫn đến ước tính tốt nhất cho hàm mục tiêu.
+ Việc đánh giá mức độ phù hợp của một mô hình học máy ước tính hàm mục tiêu có thể được tính toán bằng nhiều cách khác nhau.
+ Hàm mất mát thực hiện việc đánh giá các hệ số trong mô hình học máy bằng cách tính toán dự đoán cho mô hình đối với mỗi trường hợp huấn luyện trong tập dữ liệu và so sánh các dự đoán với các giá trị đầu ra thực tế và tính toán tổng hoặc sai số trung bình.
+ Từ hàm mất mát, có thể tính toán đạo hàm cho mỗi hệ số để có thể cập nhật bằng cách sử dụng công thức cập nhật trên.
+ Giá trị mất mát được tính cho một thuật toán học máy trên toàn bộ tập dữ liệu huấn luyện cho mỗi lần lặp của thuật toán gradient descent. Một lần lặp của thuật toán được gọi là một batch và hình thức gradient descent này được gọi là batch gradient descent.
+ Batch gradient descent là hình thức gradient descent phổ biến nhất được mô tả trong lĩnh vực học máy.

– Stochastic Gradient Descent trong học máy:
+ Gradient Descent có thể chậm trên các tập dữ liệu rất lớn.
+ Mỗi lần lặp của thuật toán gradient descent đòi hỏi phải thực hiện dự đoán cho mỗi trường hợp trong tập dữ liệu huấn luyện, nó có thể mất rất nhiều thời gian khi có hàng triệu trường hợp.
+ Đối với các trường hợp có lượng lớn dữ liệu, có thể sử dụng biến thể của gradient descent gọi là stochastic gradient descent (SGD).
+ Trong biến thể này, quá trình gradient descent được thực hiện như mô tả ở trên nhưng việc cập nhật các hệ số được thực hiện cho từng trường hợp huấn luyện, chứ không phải ở cuối batch.
+ Bước đầu tiên của quá trình cần ngẫu nhiên thứ tự của tập dữ liệu huấn luyện. Điều này nhằm trộn lẫn thứ tự của các cập nhật cho các hệ số. Bởi vì các hệ số được cập nhật sau mỗi trường hợp huấn luyện, các cập nhật này sẽ không đồng đều và có thể nhảy đi khắp nơi, và hàm mất mát tương ứng cũng sẽ nhảy đi khắp nơi. Bằng cách trộn lẫn thứ tự của các cập nhật cho các hệ số, nó khai thác được sự ngẫu nhiên này và tránh bị sa lầy hoặc bị kẹt.

+ Quy trình cập nhật cho các hệ số giống như đã mô tả ở trên, ngoại trừ việc cost không được tổng hợp trên toàn bộ các mẫu huấn luyện, mà thay vào đó được tính cho một mẫu huấn luyện.
+ Quá trình học có thể nhanh hơn rất nhiều với stochastic gradient descent đối với các tập dữ liệu huấn luyện rất lớn và thường bạn chỉ cần một số lần lặp nhỏ qua tập dữ liệu để đạt được một bộ hệ số tốt.

– Một số tips tối đa thuật toán Gradient Descent:
+ Vẽ biểu đồ cost theo thời gian: thu thập và vẽ giá trị cost ở mỗi lần lặp. Thuật toán chạy tốt khi giá trị cost giảm sau mỗi lần lặp. Nếu không giảm, hãy thử giảm learning_rate.
+ Learning_rate: là 1 số có giá trị nhỏ như 0.1, 0.001, … Hãy thử nhiều số khác nhau để tìm ra tỉ lệ học tốt nhất.
+ Chuẩn hoá dữ liệu đầu vào: thuật toán sẽ đạt cost thấp nhất nếu hình dạng hàm không bị lỗi. Điều trên có thể thực hiện bằng việc đưa tất cả các biến đầu vào về cùng 1 khoảng.
+ Few Passes: Stochastic gradient descent thường không cần nhiều hơn từ 1 đến 10 lần lặp qua tập dữ liệu huấn luyện để hội tụ với các hệ số tốt hoặc tốt đủ.
+ Plot Mean Cost: Cập nhật cho mỗi mẫu dữ liệu trong tập huấn luyện có thể dẫn đến biểu đồ giá trị lỗi nhiễu qua thời gian khi sử dụng stochastic gradient descent. Tính trung bình giá trị qua 10, 100 hoặc 1000 cập nhật có thể giúp bạn có cái nhìn tốt hơn về xu hướng học của thuật toán.

3. Sinh viên chạy ví dụ sau đây và giải thích ý nghĩa flatten

```
//Flatten is used to flatten the input. For example, if flatten is applied to layer having input

//shape as (batch_size, 2,2), then the output shape of the layer will be (batch_size,4)

model = Sequential()

layer_1 = Dense(16, input_shape=(8,8))

model.add(layer_1)

layer_2 = Flatten()

model.add(layer_2)
```

layer_2.input_shape #(None, 8, 16)

layer_2.output_shape #(None, 128)

- Flatten được sử dụng để làm phẳng đầu vào. Ví dụ: nếu làm phẳng được áp dụng cho lớp có đầu vào hình dạng là (batch_size, 2,2), thì hình dạng đầu ra của lớp sẽ là(batch_size, 4)
- VD: như 1 ảnh có kích cỡ 5x5 thì flatten ra sẽ thành 25 giá trị
- Ý nghĩa của flatten: biến đổi cấu trúc dữ liệu đầu vào để phù hợp với kiến trúc của mạng nơ-ron. Chủ yếu thực hiện việc chuyển đổi từ dữ liệu dạng ma trận (nhiều chiều) thành dạng vectơ (1 chiều)

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

model = Sequential()
layer_1 = Dense(16, input_shape=(8,8))
model.add(layer_1)
layer_2 = Flatten()
model.add(layer_2)
layer_2.input_shape
model.summary()
```

```
chấm chấm:
---------------
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 8, 16)             144

 flatten_4 (Flatten)         (None, 128)               0

=================================================================
Total params: 144 (576.00 Byte)
Trainable params: 144 (576.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

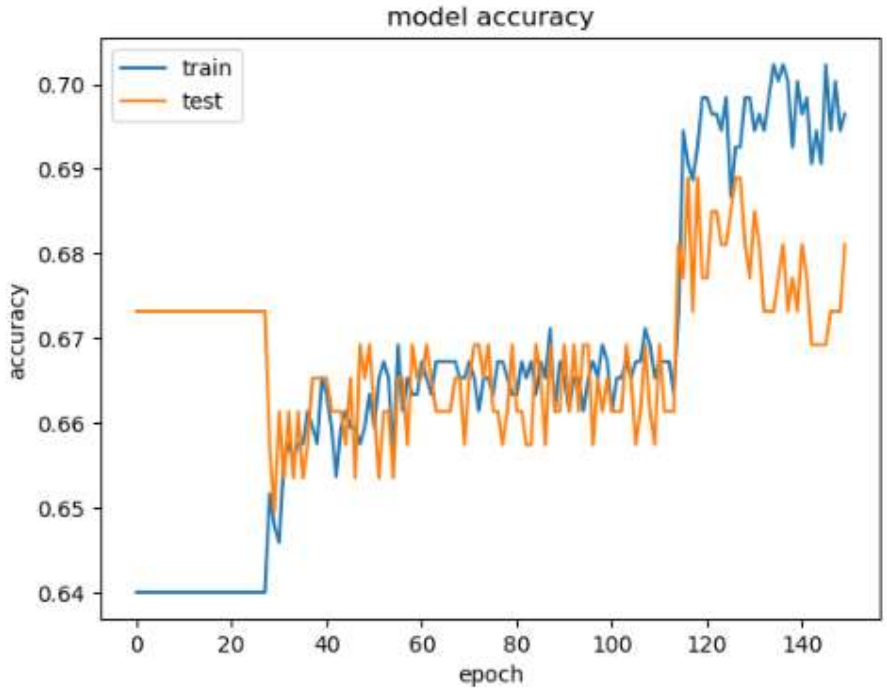
- model = Sequential()
  + Đầu tiên, tạo một đối tượng Sequential để xây dựng mô hình mạng nơ-ron. layer_1 = Dense(16, input_shape=(8,8))
  + Thêm một lớp Dense với 16 đơn vị nơ-ron và input_shape=(8, 8). Lớp này là lớp đầu tiên của mạng và sẽ có 16 nơ-ron đầu ra. input_shape=(8, 8) chỉ ra rằng đầu vào của lớp này là một ma trận có kích thước 8x8.
- Thêm một lớp Flatten. Lớp này không có nơ-ron nào, nhưng nó có một chức năng quan trọng. Lớp Flatten được sử dụng để biến đổi dữ liệu từ một ma trận nhiều chiều thành một vectơ 1 chiều. Trong trường hợp này, sau khi lớp Flatten được áp dụng, kết quả của lớp Dense trước đó (với 16 nơ-ron) sẽ được biến đổi từ một ma

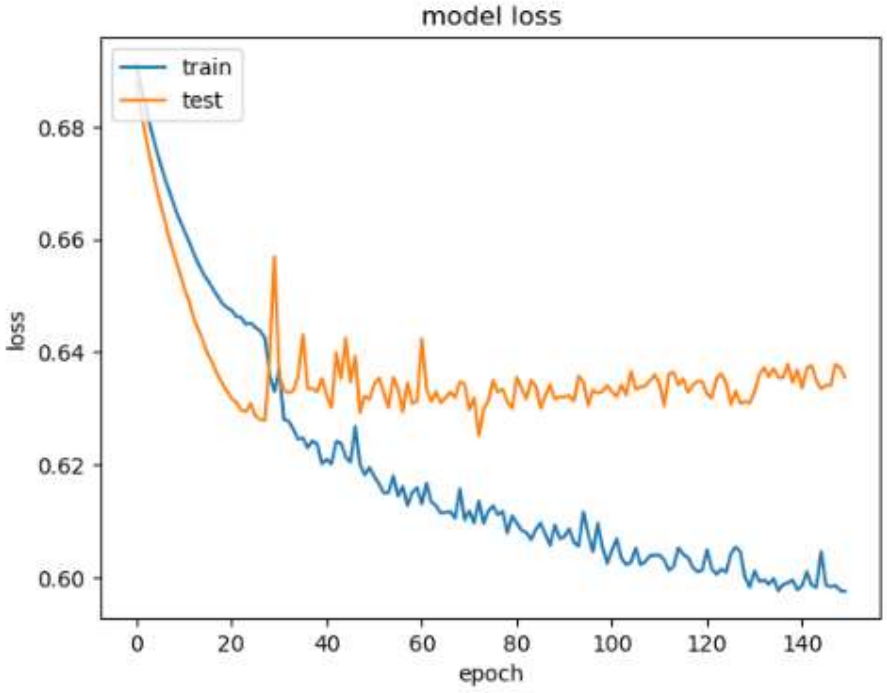
trận có kích thước (None, 8, 16) thành một vectơ có kích thước (None, 128). Điều này có nghĩa là tất cả các phần tử trong ma trận 8x16 sẽ được ghép nối lại để tạo thành một vectơ 1 chiều với 128 phần tử.

4. Sinh viên áp dụng kiểu khai báo layer_1, model.add(layer_1) để viết lại khai báo cấu trúc mạng neuron trong các câu của Bài tập 5

```
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()

# Định nghĩa các lớp layers
layer_1 = Dense(5, input_dim=8, activation='relu')
layer_2 = Dense(4, activation='relu')
layer_3 = Dense(2, activation='relu')
output_layer = Dense(1, activation='sigmoid')

# Thêm các lớp layers vào mô hình sử dụng model.add(layer)
model.add(layer_1)
model.add(layer_2)
model.add(layer_3)
model.add(output_layer)

# 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'])
```



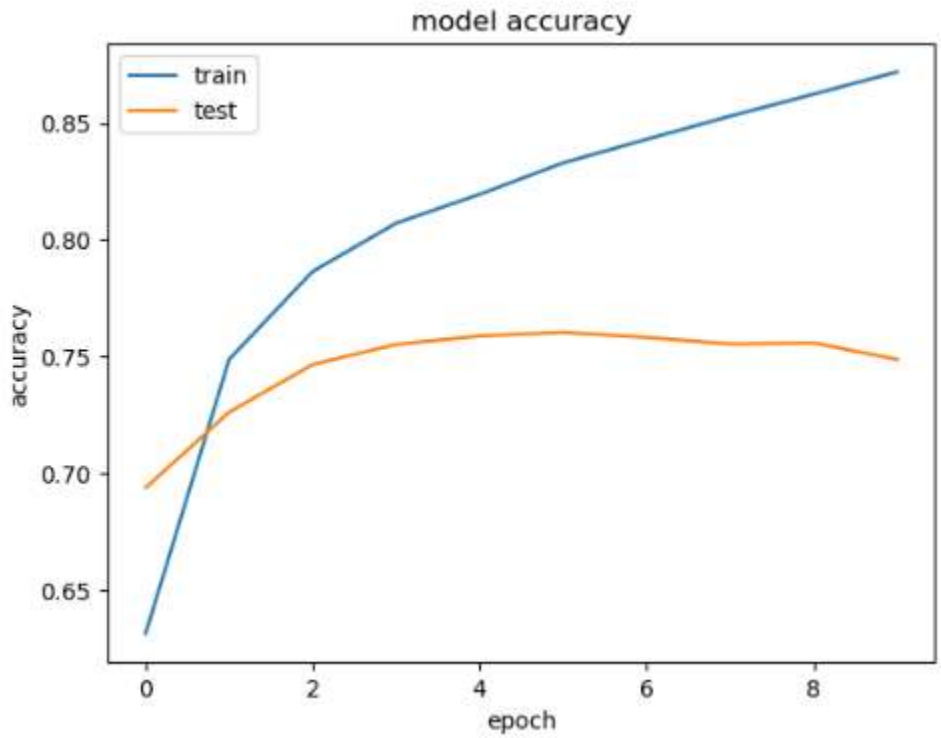


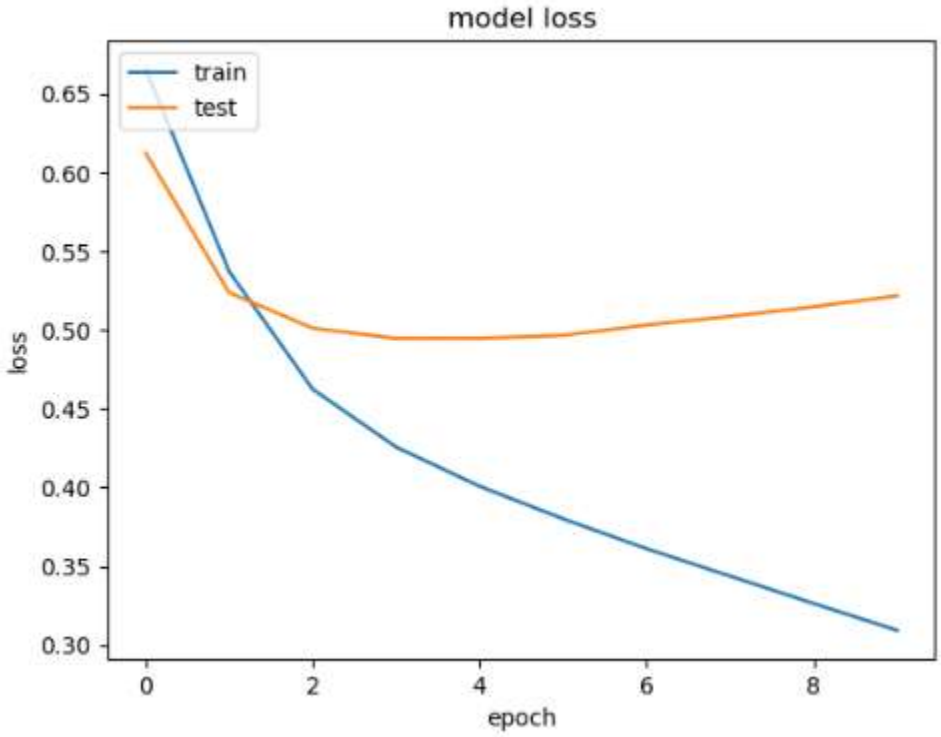
Với câu 5.2

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

max_features = 10000
maxlen = 20

(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)

model = Sequential()

# Định nghĩa các lớp layers
layer_1 = Embedding(10000, 8, input_length=maxlen)
layer_2 = Flatten()
output_layer = Dense(1, activation='sigmoid')

# Thêm các lớp layers vào mô hình sử dụng model.add(layer)
model.add(layer_1)
model.add(layer_2)
model.add(output_layer)

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()

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

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()

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_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 20, 8)             80000

 flatten_1 (Flatten)         (None, 160)               0

 dense_11 (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 [==============================] - 5s 6ms/step - loss: 0.6656 - acc: 0.6314 - val_loss: 0.6123 - val_acc: 0.6938
Epoch 2/10
625/625 [==============================] - 3s 5ms/step - loss: 0.5370 - acc: 0.7488 - val_loss: 0.5237 - val_acc: 0.7262
Epoch 3/10
625/625 [==============================] - 4s 7ms/step - loss: 0.4624 - acc: 0.7865 - val_loss: 0.5011 - val_acc: 0.7466
Epoch 4/10
625/625 [==============================] - 5s 8ms/step - loss: 0.4257 - acc: 0.8072 - val_loss: 0.4946 - val_acc: 0.7552
Epoch 5/10
625/625 [==============================] - 5s 7ms/step - loss: 0.4008 - acc: 0.8195 - val_loss: 0.4947 - val_acc: 0.7588
Epoch 6/10
625/625 [==============================] - 3s 5ms/step - loss: 0.3801 - acc: 0.8329 - val_loss: 0.4966 - val_acc: 0.7604
Epoch 7/10
625/625 [==============================] - 3s 4ms/step - loss: 0.3611 - acc: 0.8431 - val_loss: 0.5031 - val_acc: 0.7582
Epoch 8/10
625/625 [==============================] - 3s 4ms/step - loss: 0.3438 - acc: 0.8529 - val_loss: 0.5086 - val_acc: 0.7554
Epoch 9/10
625/625 [==============================] - 3s 5ms/step - loss: 0.3264 - acc: 0.8624 - val_loss: 0.5147 - val_acc: 0.7558
Epoch 10/10
625/625 [==============================] - 3s 5ms/step - loss: 0.3093 - acc: 0.8719 - val_loss: 0.5217 - val_acc: 0.7488
```



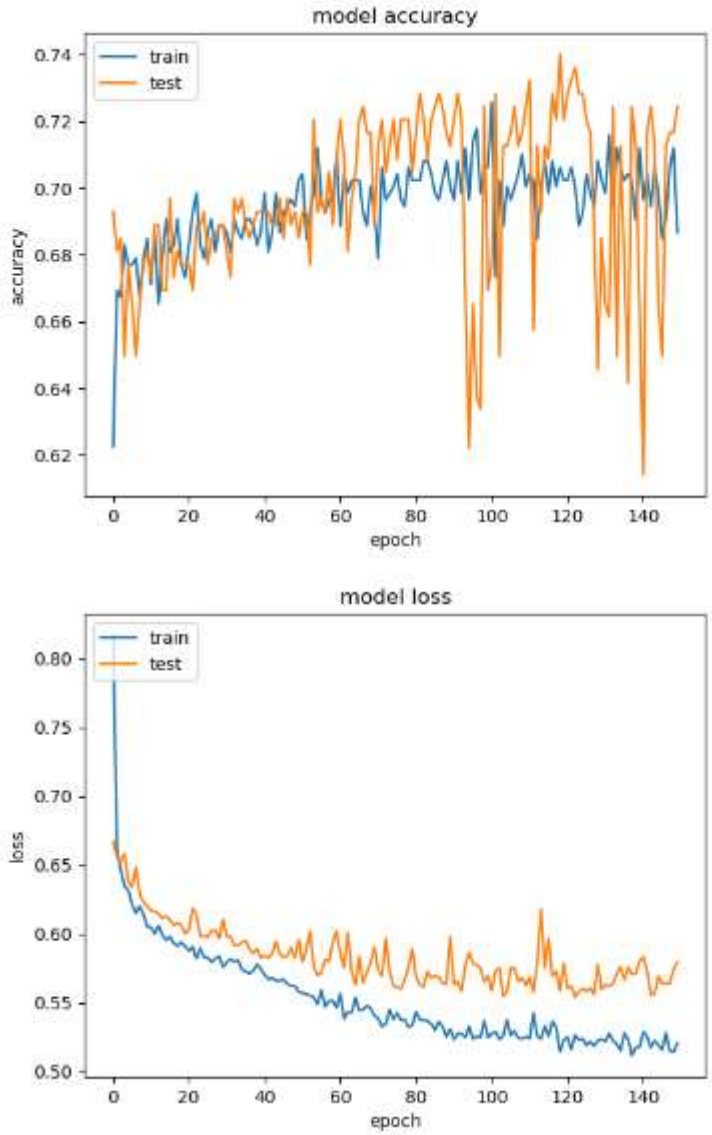


5. Sử dụng lại Bài tập 5 và Bài tập 6 (Câu 3), Sinh viên thêm làm lượt 2,3 layer với số lượng neuron khác nhau. Chạy từng trường hợp và sử dụng các Biểu đồ output để so sánh các trường hợp với nhau khi chưa thêm layer, them layer mới.

– Thử thêm 2 layer ở giữa với 16, 4 neuron:

In [1]:
```
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()

# Định nghĩa các lớp layers
layer_1 = Dense(12, input_dim=8, activation='relu')
layer_2 = Dense(8, activation='relu')
layer_3 = Dense(4, activation='relu')
layer_4 = Dense(16, activation='relu')
output_layer = Dense(1, activation='sigmoid')

# Thêm các lớp layers vào mô hình sử dụng model.add(layer)
model.add(layer_1)
model.add(layer_2)
model.add(layer_3)
model.add(layer_4)
model.add(output_layer)

# 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'])
```

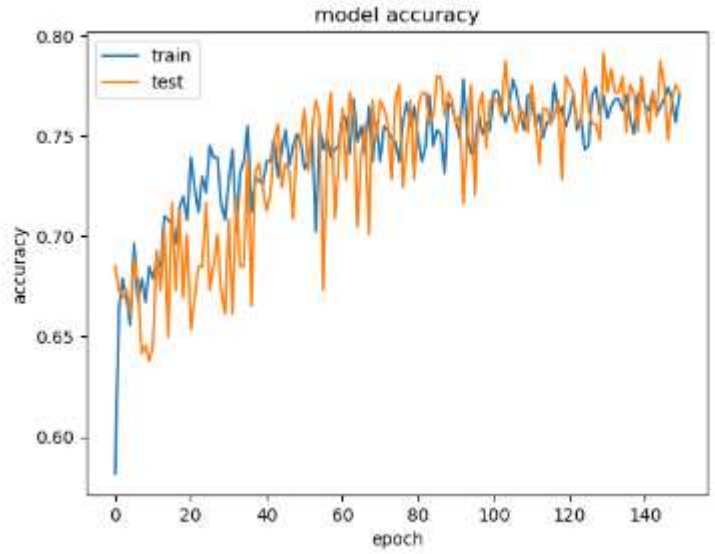


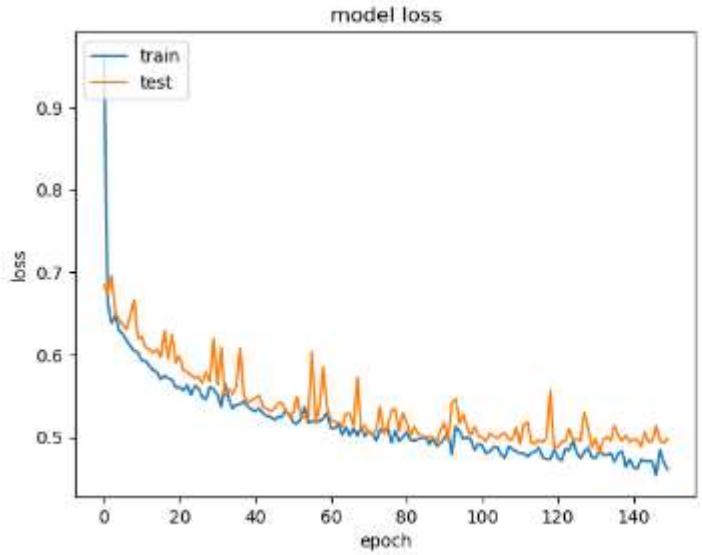
– Kết quả thấy rằng nếu các layer gần nhau có độ chênh lệch lớn và layer sau lớn hơn thìkết quả sẽ bị sai lệch khá nhiều, thử thêm 3 layer nhưng độ chênh lệch ít và layer gầngiống nhau:

In [2]:

```
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()

# Định nghĩa các lớp layers
layer_1 = Dense(12, input_dim=8, activation='relu')
layer_2 = Dense(10, activation='relu')
layer_3 = Dense(8, activation='relu')
layer_4 = Dense(6, activation='relu')
layer_5 = Dense(8, activation='relu')
output_layer = Dense(1, activation='sigmoid')

# Thêm các lớp layers vào mô hình sử dụng model.add(layer)
model.add(layer_1)
model.add(layer_2)
model.add(layer_3)
model.add(layer_4)
model.add(layer_5)
model.add(output_layer)

# 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'])
```

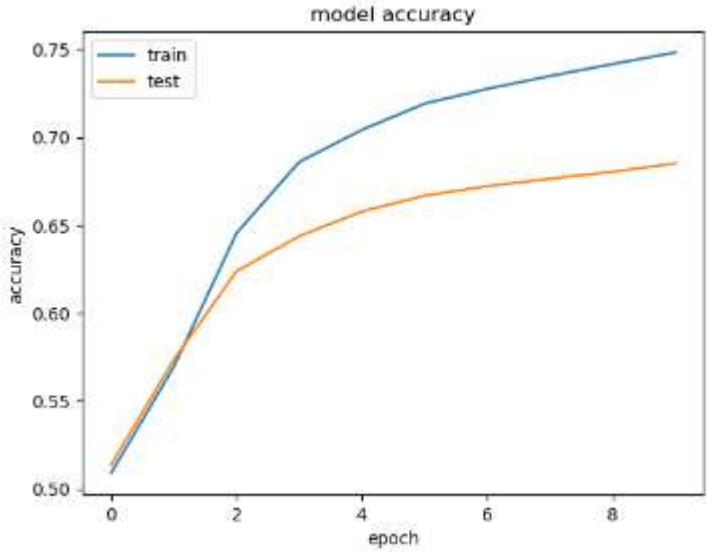


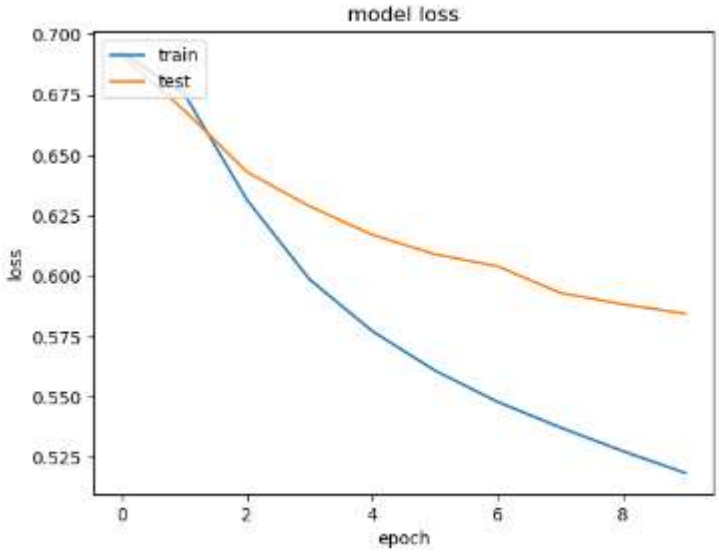


- Ta nhìn kết quả có vẻ khả quan hơn.
  Thêm layer sẽ giúp cái thiện hoặc giảm hiệu suất kết quả đầu ra. Việc thay đổi các layer sẽ làm thay đổi cấu trúc mạng neuron.
- Với VD 2 của câu 5:

In [3]:

```
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

max_features = 10000
maxlen = 20

(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)

model = Sequential()

# Định nghĩa các lớp layers
layer_1 = Embedding(10000, 8, input_length=maxlen)
layer_2 = Dense(8, activation = 'relu')
layer_3 = Dense(2, activation='relu')
layer_4 = Flatten()
output_layer = Dense(1, activation='sigmoid')

# Thêm các lớp layers vào mô hình sử dụng model.add(layer)
model.add(layer_1)
model.add(layer_2)
model.add(layer_3)
model.add(layer_4)
model.add(output_layer)

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()

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

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()

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()
```

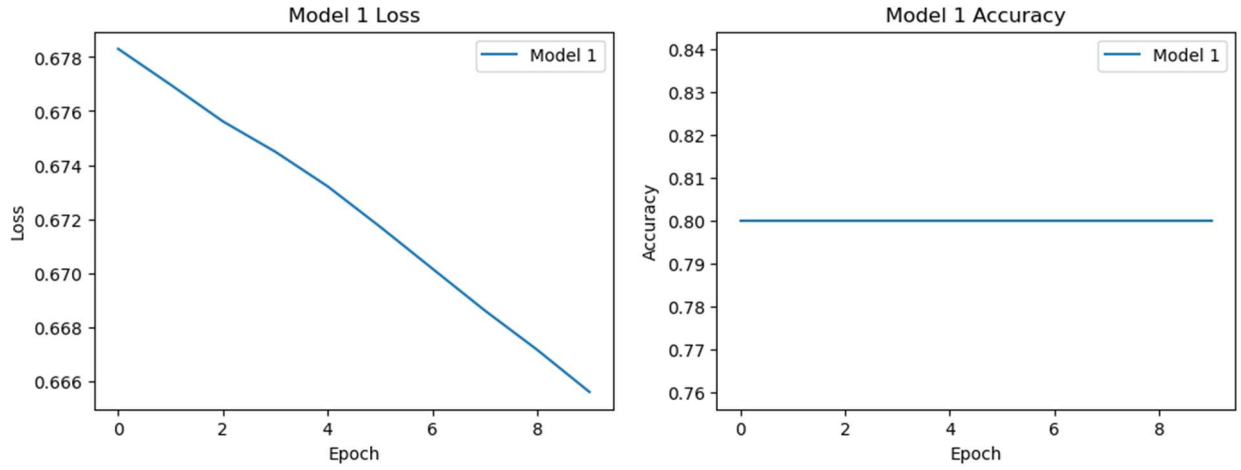




- Với câu 6.3: khi thêm 2 layer:

```python
import numpy as np
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense, Dropout
import matplotlib.pyplot as plt

# Dữ liệu đánh giá nhà hàng và nhãn lớp (positive/negative)
reviews = ['The food was great!',
           'The service was terrible.',
           'I loved the ambiance.',
           'The food and service were excellent.',
           'Not a good experience.']

labels = np.array([1, 0, 1, 1, 0])

# Sử dụng Tokenizer để mã hóa văn bản thành các chuỗi số nguyên
max_words = 1000  # Số từ tối đa trong từ vựng
tokenizer = Tokenizer(num_words=max_words, oov_token='<OOV>')
tokenizer.fit_on_texts(reviews)
sequences = tokenizer.texts_to_sequences(reviews)

# Điền (padding) các chuỗi số nguyên để có độ dài tối đa
max_sequence_length = 10
padded_sequences = pad_sequences(sequences, maxlen=max_sequence_length, padding='post')

# Xây dựng các mô hình mạng nơ-ron khác nhau

# Mô hình 1: Mô hình đơn giản
model1 = Sequential()
model1.add(Embedding(max_words, 8, input_length=max_sequence_length))
model1.add(Flatten())

model1.add(Dense(16, activation='relu'))
model1.add(Dense(8, activation='relu'))

model1.add(Dense(1, activation='sigmoid'))

# Mô hình 2: Mô hình với thêm lớp Dropout để tránh overfitting
model2 = Sequential()
model2.add(Embedding(max_words, 8, input_length=max_sequence_length))
model2.add(Flatten())
model2.add(Dense(4, activation='relu'))
model2.add(Dropout(0.5))
model2.add(Dense(1, activation='sigmoid'))

# Mô hình 3: Mô hình với kiến trúc phức tạp hơn
model3 = Sequential()
model3.add(Embedding(max_words, 16, input_length=max_sequence_length))
model3.add(Flatten())
model3.add(Dense(8, activation='relu'))

model3.add(Dense(8, activation='relu'))
model3.add(Dense(1, activation='sigmoid'))
```

```python
# Biên dịch và huấn luyện các mô hình
models = [model1, model2, model3]

for i, model in enumerate(models):
    print(f"Model {i + 1} Summary:")
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    print(model.summary())
    model.fit(padded_sequences, labels, epochs=10, verbose=0)
    loss, accuracy = model.evaluate(padded_sequences, labels, verbose=0)
    print(f"Accuracy (Model {i + 1}): {accuracy * 100}%")


# Số lượng mô hình
num_models = len(models)

# Biểu đồ loss và accuracy cho từng mô hình
for i, model in enumerate(models):
    # Huấn luyện mô hình (nếu chưa được huấn luyện)
    history = model.fit(padded_sequences, labels, epochs=10, verbose=0)

    # Lấy lịch sử (history) của mô hình
    loss = history.history['loss']
    accuracy = history.history['accuracy']

    # Vẽ biểu đồ loss
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(loss, label=f'Model {i + 1}')
    plt.title(f'Model {i + 1} Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    # Vẽ biểu đồ accuracy
    plt.subplot(1, 2, 2)
    plt.plot(accuracy, label=f'Model {i + 1}')
    plt.title(f'Model {i + 1} Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

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

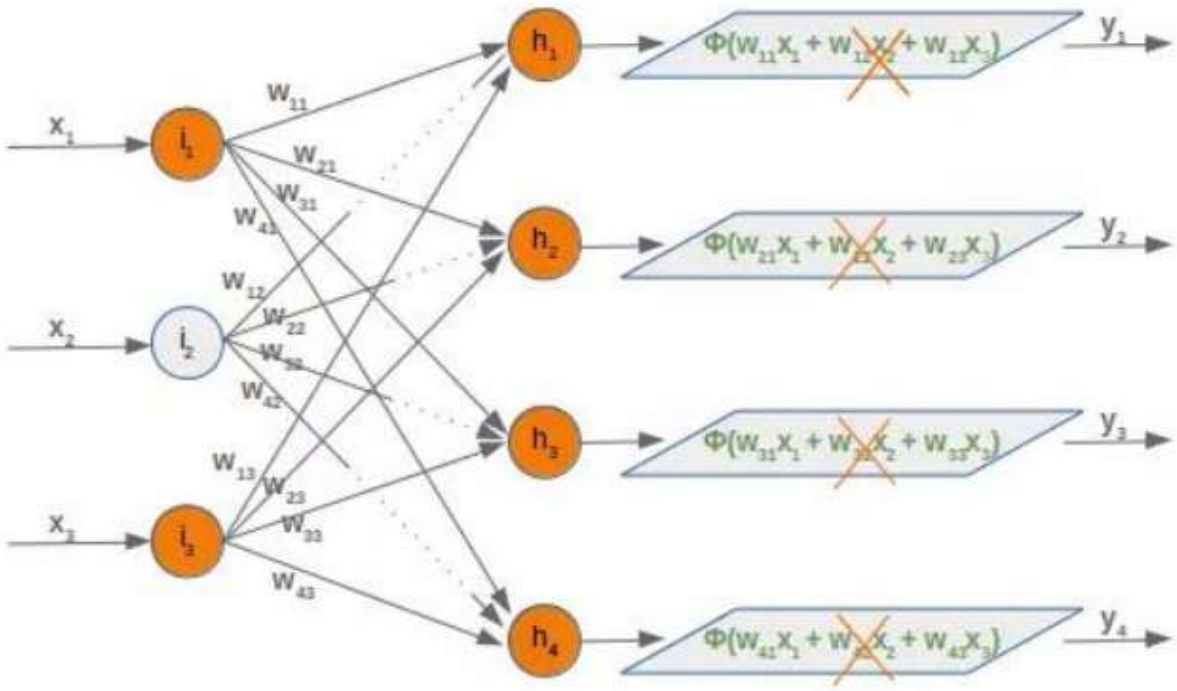


Thấy được loss là 1 đường cong thay vì 1 đường thẳng

6. Drop out là gì? Sử dụng drop out để chay và so sánh 3 dang trong Câu 5

- Trong neural network, việc cuối cùng là tối ưu các tham số để làm cho giảm loss

- function, nhưng đôi khi có unit thay đổi theo cách sửa lại lỗi của các unit khác dẫnđến việc hòa trộn làm giảm tính dự đoán của model, hay còn gọi là overfitting.
- Dropout là cách thức mà chúng ta giả định một phần các unit bị ẩn đi trong quá trình training, qua đó làm giảm tích hòa trộn (hay nói cách khác là 1 hidden unit không thể dựa vào 1 unit khác để sửa lỗi lầm của nó, dễ cho chúng ta thấy các hidden unit không đáng tin cậy).
- Tại mỗi step trong quá trình training, khi thực hiện Forward Propagation (Lan truyền xuôi) đến layer sử dụng Drop-Out, thay vì tính toán tất cả unit có trên layer, tại mỗi unit ta "gieo xúc xắc" xem unit đó có được tính hay không dựa trên xác suất p.

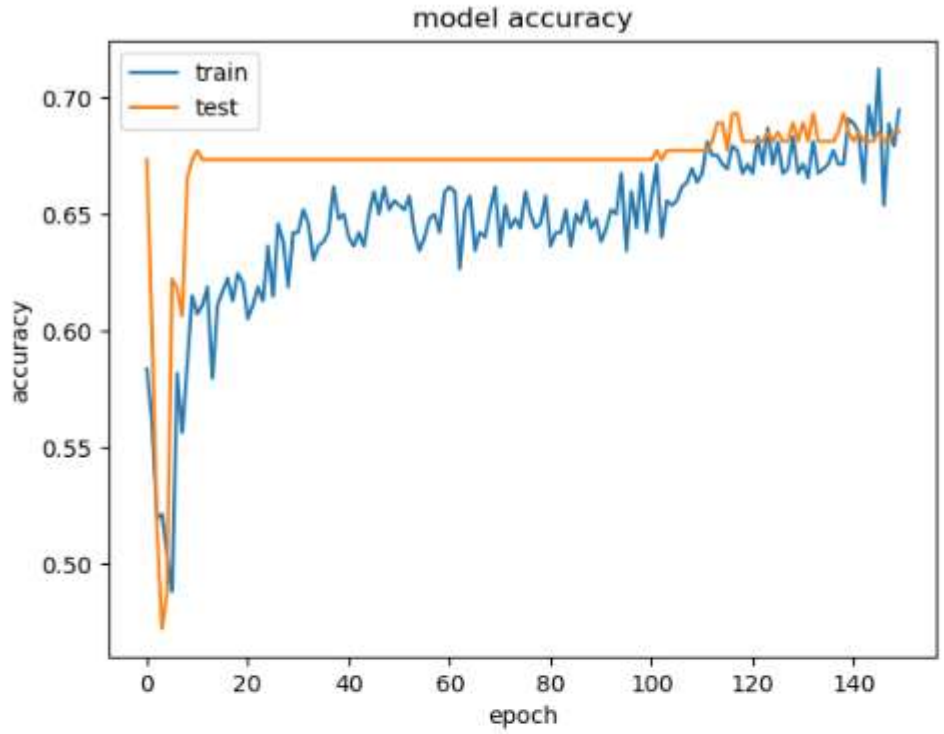


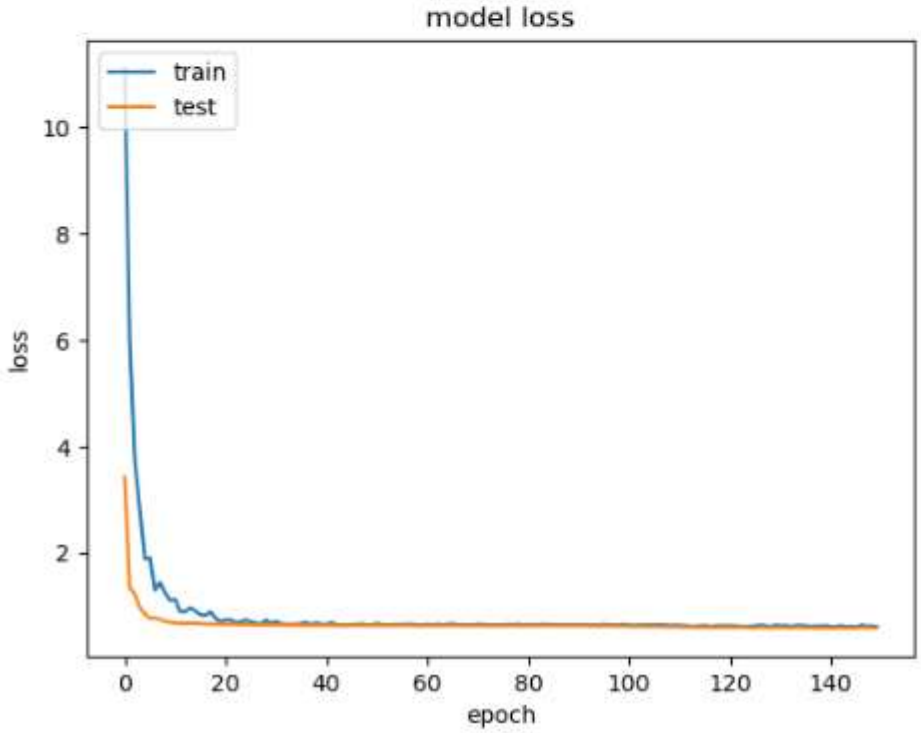
- Cách thức hoạt động của dropout là để đạt được kết qủa trung bình của việc train nhiều mạng con trong network (bằng việc giả định ẩn đi % unit) thay vì chỉ lấy kết quả dựa trên việc train 1 mạng mẹ.
- Dropout là một kỹ thuật quan trọng trong học máy và mạng nơ-ron sâu, được sử dụng để giảm hiện tượng quá khớp (overfitting). Quá khớp xảy ra khi mô hình mạng nơ-ron học rất tốt trên dữ liệu huấn luyện, nhưng không tổng quát hóa tốt cho dữ liệu mới.

- Dropout hoạt động bằng cách tạm thời "tắt" ngẫu nhiên một số lượng các đơn vị nơ-ron trong một lớp trong quá trình huấn luyện. Nó ngăn một lớp nơ-ron cụ thể khỏi việc học dựa trên các mẫu cụ thể trong dữ liệu huấn luyện. Mục tiêu của Dropout là tạo ra một mô hình mạng nơ-ron tốt hơn trong việc tổng quát hóa cho dữ liệu mới.
- Lý do Dropout hoạt động là nó làm cho mạng trở nên kháng lại việc "nhớ" dữ liệu huấn luyện cụ thể và buộc nó phải học các đặc trưng tổng quát hơn.
- Sử dụng drop out: model.add(Dropout(rate)) trong đó rate là tỉ lệ loại bỏ nút, vd 0.2 = 20%
  Với câu 5.1: thử drop out với tỉ lệ 30% 2 lần sau mỗi layer

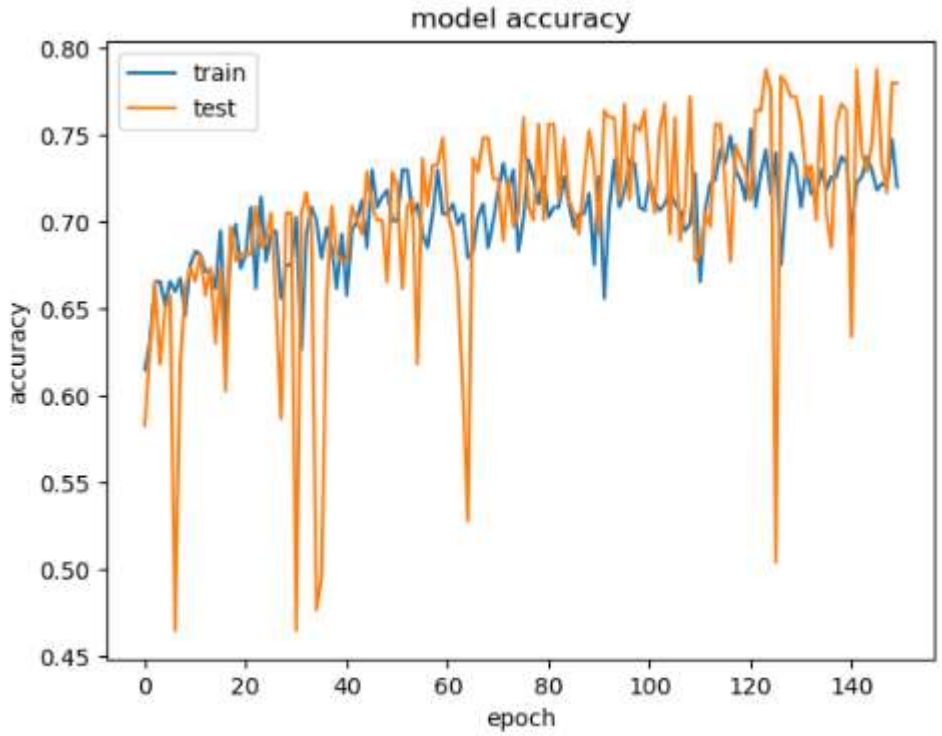
In [2]:
```
# Import các thư viện cần thiết
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.layers import Dropout

# 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(Dropout(0.3))
model.add(Dense(8, activation='relu'))             # Lớp ẩn với 8 đơn vị và hàm kích hoạt ReLU để giúp mô hình phi tuyến tính
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))          # Lớp đầu ra với hàm kích hoạt Sigmoid để đối đầu ra các giá trị trong khoảng

# 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
```

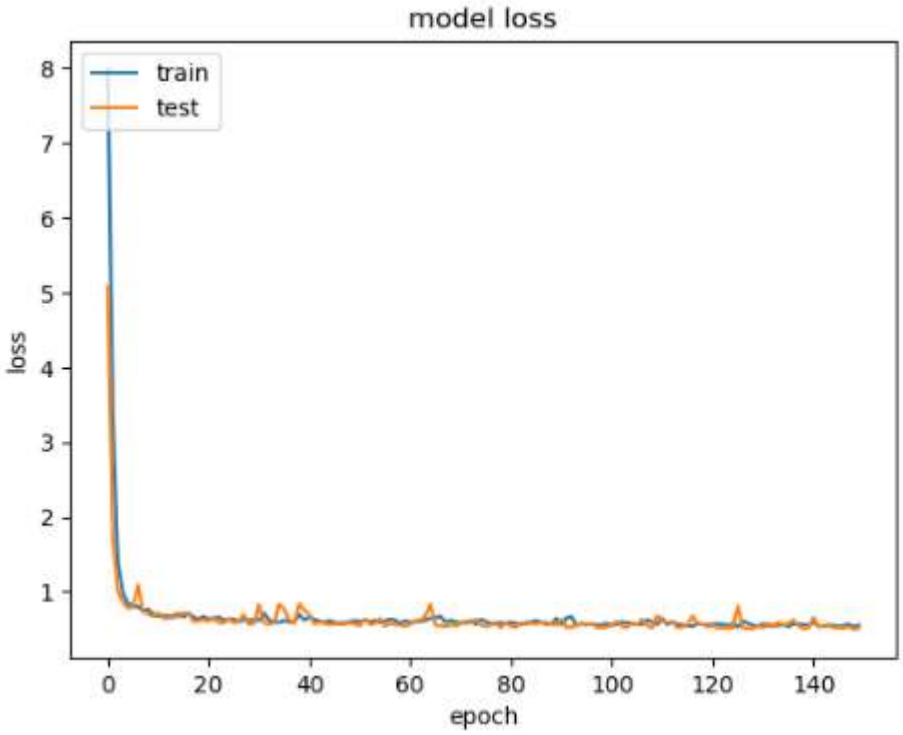
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```





Kết quả trước đó:

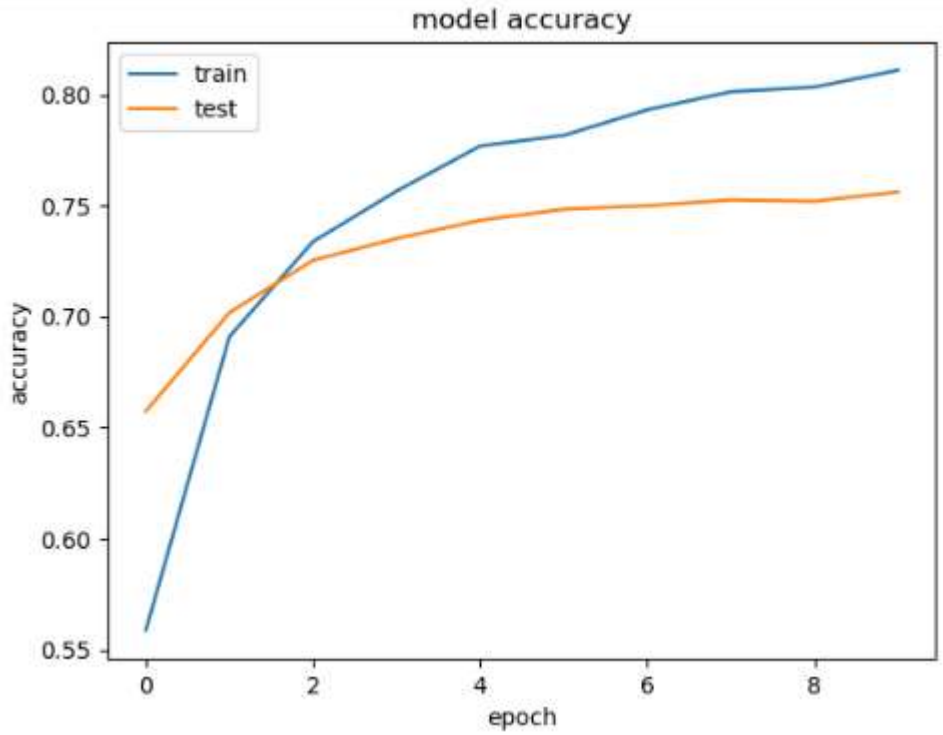


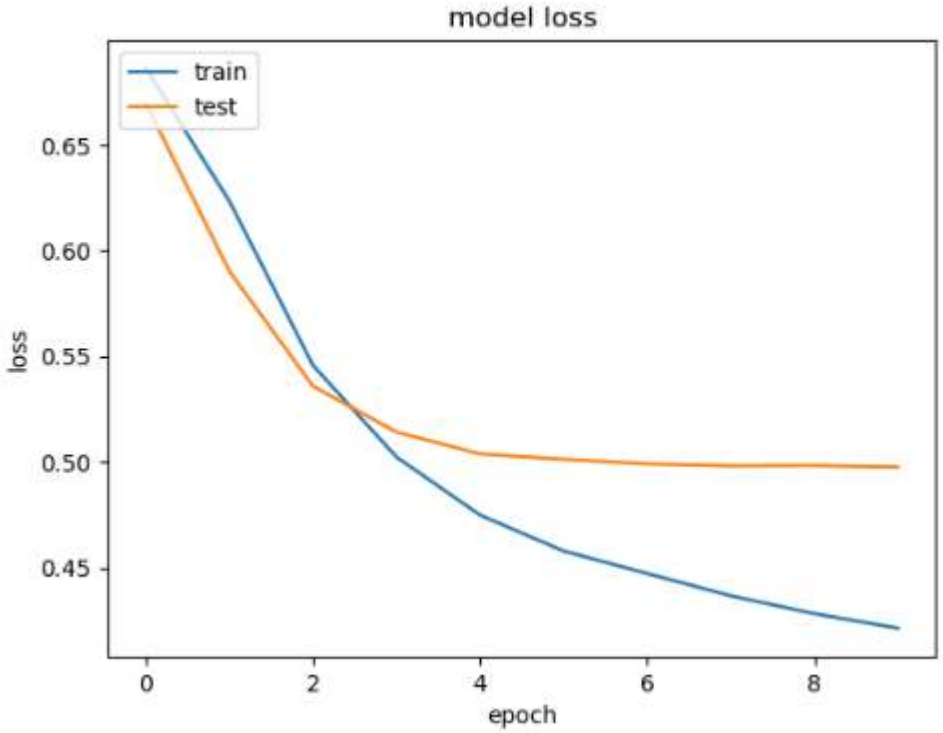


Kết quả là độ chính xác sẽ giảm đi tuy nhiên khi thực tế với bộ dữ liệu mới thì mô hình sẽ cho ra kết quả chính xác với kết quả hiệu suất mà mô hình đã huấn luyện.

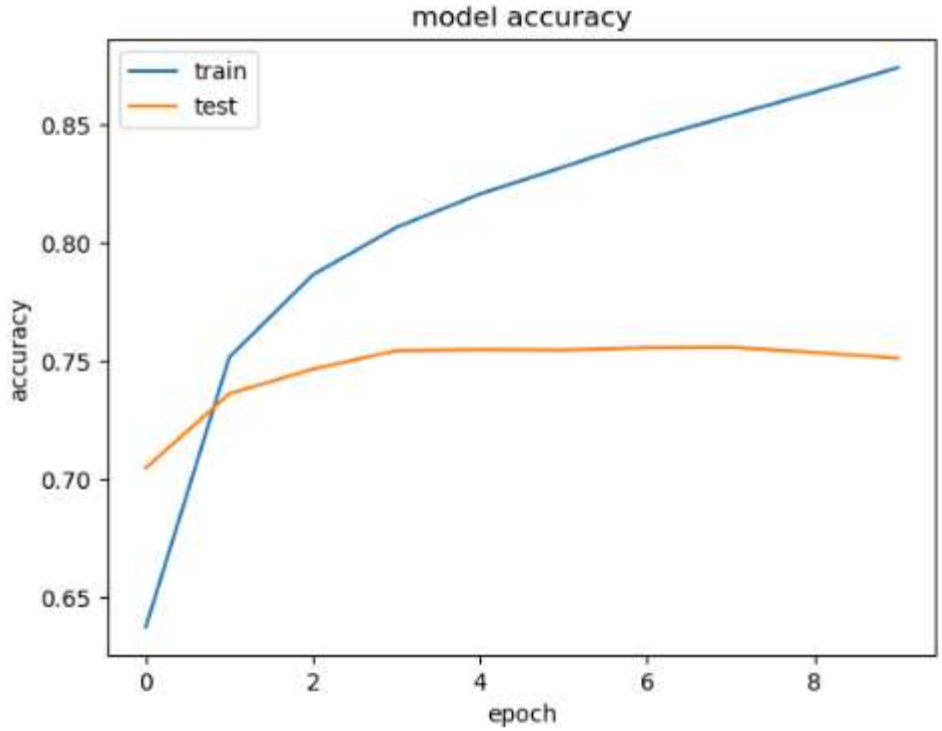
Với câu 5.2: cho tỉ lệ drop out với 0.5

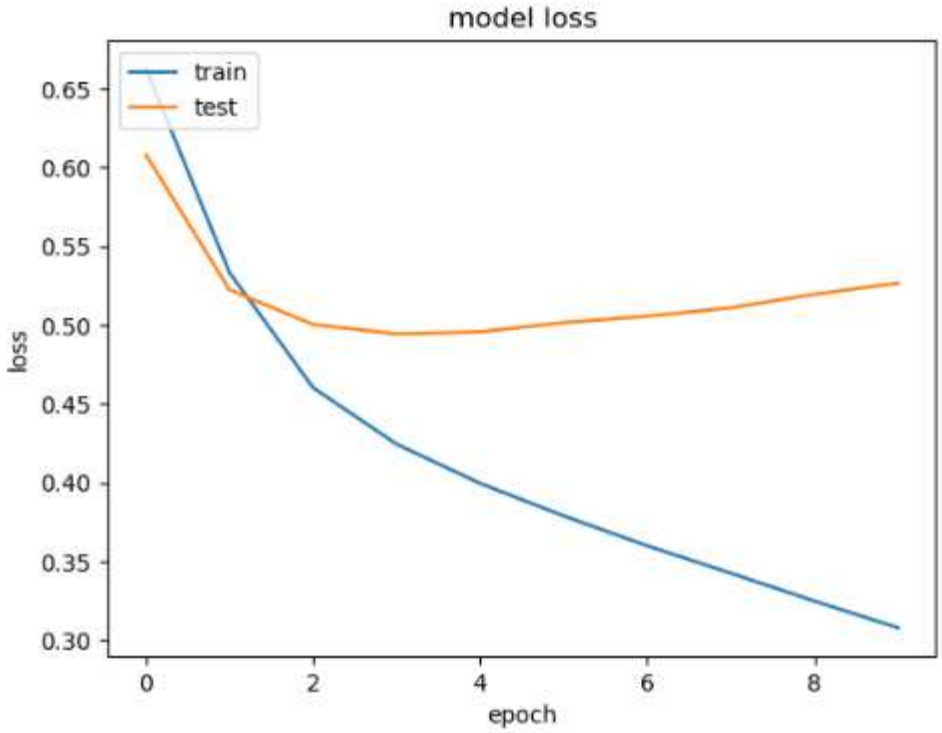
```
# Thêm lớp Embedding để biểu diễn từ
model.add(Embedding(10000, 8, input_length=maxlen))

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

model.add(Dropout(0.5))

# 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)
```


model accuracy
train
test
0.80
0.75
0.70
0.65
0.60
0.55
accuracy
0
2
4
6
8
epoch


model loss
train
test
0.65
0.60
0.55
0.50
0.45
loss
0
2
4
6
8
epoch

Trước đó:





Ta lại thấy kết quả cũng có sự thay đổi. Test và train gần nhau hơn.