Họ tên: Trần Vân Anh

Mã SV: B20DCCN075

1. Giải thích các khái niệm

1.1. Sample(mẫu)

Một sample hoặc mẫu là một ví dụ dữ liệu đầu vào mà mô hình máy học sử dụng để học. Ví dụ, trong bài toán phân loại hình ảnh, mỗi hình ảnh có thể được coi là một mẫu.

1.2. Iteration(Lặp)

Mỗi iteration trong quá trình đào tạo mô hình thường bao gồm việc chạy một batch dữ liệu qua mô hình một lần. Trong mỗi iteration, mô hình tính toán độ lỗi (loss) và cập nhật trọng số dựa trên dữ liệu trong batch đó.

1.3. Epoch(Vòng lặp)

Một epoch là khi toàn bộ tập dữ liệu huấn luyện đã được đưa qua mô hình máy học một lần. Trong mỗi epoch, mô hình có cơ hội học từ toàn bộ dữ liệu huấn luyện.

1.4. Batch(Lô)

Một batch là một tập con của dữ liệu huấn luyện được sử dụng trong mỗi iteration. Sử dụng batch giúp làm giảm tải cho việc tính toán, đặc biệt khi tập dữ liệu lớn. Kích thước của batch được xác định trước và thường là một số nguyên như 32, 64, 128, và còn nhiều giá trị khác.

1.5. Update weights(Cập nhật trọng số)

Sau mỗi iteration, mô hình tính toán độ lỗi (loss) dựa trên kết quả dự đoán và các giá trị thực tế trong batch, sau đó cập nhật các trọng số của mô hình để giảm thiểu độ lỗi. Quá trình này thường được thực hiện bằng cách sử dụng các thuật toán tối ưu hóa như gradient descent.

1.6. 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?

Số mẫu: 200

Kích thước batch: 5

Số epoch: 100

Số lần iteration = Số mẫu / Kích thước batch = 200 / 5 = 40 iterations trong mỗi epoch.

Số lần cập nhật trọng số = Số lần iteration * Số epoch = 40 * 100 = 4000 lần cập nhật trọng số trong quá trình đào tạo.

2. Stochastic Gradient Descent, or SGD

Thuật toán tối ưu hóa là một phần quan trọng của học máy. Gần như mọi thuật toán học máy đều có một thuật toán tối ưu hóa ở trung tâm của nó.

Trong bài viết này, bạn sẽ tìm hiểu về một thuật toán tối ưu hóa đơn giản mà bạn có thể sử dụng với bất kỳ thuật toán học máy nào. Nó dễ hiểu và dễ triển khai. Sau khi đọc bài viết này, bạn sẽ biết:

- Gradient Descent là gì?
- Làm thế nào Gradient Descent có thể được sử dụng trong các thuật toán như hồi quy tuyến tính?
- Làm thế nào Gradient Descent có thể mở rộng để làm việc với tập dữ liệu rất lớn?
- Một số gợi ý để tận dụng tối đa Gradient Descent.

2.1. Gradient Descent Cho Học Máy

Gradient Descent là một thuật toán tối ưu hóa được sử dụng để tìm các giá trị của các tham số (hệ số) của một hàm (f) sao cho giảm thiểu một hàm chi phí (cost).

Gradient Descent thường được sử dụng khi các tham số không thể được tính toán phân tích (ví dụ: sử dụng đại số tuyến tính) và phải được tìm kiếm bằng một thuật toán tối ưu hóa.

2.2. Trực quan về Gradient Descent

Hãy tưởng tượng một cái bát lớn giống như cái bát bạn ăn bữa sáng hoặc lưu trữ trái cây. Cái bát này là biểu đồ của hàm chi phí (f).

Một vị trí ngẫu nhiên trên mặt phẳng của cái bát là chi phí của các giá trị hiện tại của các hệ số (cost).

Đáy của cái bát là chi phí của tập hợp tốt nhất các hệ số, là giá trị tối thiểu của hàm.

Mục tiêu là tiếp tục thử nghiệm các giá trị khác nhau cho các hệ số, đánh giá chi phí của chúng và chọn các hệ số mới có chi phí tốt hơn (thấp hơn) một chút.

Lặp lại quá trình này đủ lần sẽ dẫn đến đáy của cái bát và bạn sẽ biết các giá trị của các hệ số dẫn đến chi phí tối thiểu.

2.3. Quy trình Gradient Descent

Quy trình bắt đầu với các giá trị ban đầu cho hệ số hoặc các hệ số của hàm. Chúng có thể là 0.0 hoặc một giá trị ngẫu nhiên nhỏ.

```
coefficient = 0.0
```

Chi phí của các hệ số được đánh giá bằng cách đưa chúng vào hàm và tính toán chi phí.

```
cost = f(coefficient)
```

hoặc

```
cost = evaluate(f(coefficient))
```

Đạo hàm của chi phí được tính toán. Đạo hàm là một khái niệm từ tính toán và tham chiếu đến độ dốc của hàm tại một điểm cụ thể. Chúng ta cần biết độ dốc để biết hướng (dấu) để di chuyển giá trị của hệ số để có chi phí thấp hơn trong lần lặp kế tiếp.

```
delta = derivative(cost)
```

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

coefficient = coefficient – (alpha * delta)

Quy trình này được lặp lại cho đến khi chi phí của các hệ số (cost) là 0.0 hoặc đủ gần với số không để đủ tốt.

Như bạn có thể thấy, Gradient Descent rất đơn giản. Nó yêu cầu bạn biết đạo hàm của hàm chi phí hoặc hàm bạn đang tối ưu hóa, nhưng ngoài ra, nó rất dễ hiểu. Tiếp theo, chúng ta sẽ xem cách chúng ta có thể sử dụng nó trong các thuật toán máy học.

2.4. Batch Gradient Descent cho Học Máy

Mục tiêu của tất cả các thuật toán máy học có giám sát là ước tính một hàm mục tiêu (f) tốt nhất mô tả dữ liệu đầu vào (X) thành các biến đầu ra (Y). Điều này áp dụng cho tất cả các bài toán phân loại và hồi quy.

Một số thuật toán máy học có các hệ số mô tả ước tính của thuật toán cho hàm mục tiêu (f). Các thuật toán khác có biểu diễn khác nhau và các hệ số khác nhau, nhưng nhiều trong số chúng yêu cầu quá trình tối ưu hóa để tìm bộ các hệ số dẫn đến ước tính tốt nhất cho hàm mục tiêu.

Việc đánh giá mức độ phù hợp của mô hình máy học ước tính cho hàm mục tiêu có thể được tính toán bằng một số cách khác nhau, thường cụ thể cho thuật toán máy học. Hàm chi phí liên quan đến việc đánh giá các hệ số trong mô hình máy học bằng cách tính toán dự đoán cho mô hình cho mỗi trường hợp đào tạo trong tập dữ liệu và so sánh dự đoán với các giá trị đầu ra thực tế và tính toán một tổng hoặc lỗi trung bình (như Tổng Bình phương Dư thừa hoặc SSR trong trường hợp của hồi quy tuyến tính).

Từ hàm chi phí, có thể tính toán đạo hàm cho từng hệ số để có thể cập nhật chúng bằng cách sử dụng công thức cập nhật được mô tả ở trên.

Chi phí được tính toán cho một thuật toán máy học trên toàn bộ tập dữ liệu đào tạo 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 phổ biến nhất của Gradient Descent được mô tả trong máy học.

2.5. Stochastic Gradient Descent cho Học Máy

Gradient Descent có thể chậm khi chạy trên các tập dữ liệu rất lớn.

Bởi vì một lần lặp của thuật toán Gradient Descent yêu cầu dự đoán cho mỗi trường hợp trong tập dữ liệu đào tạo, nó có thể mất nhiều thời gian khi bạn có hàng triệu trường hợp.

Trong những tình huống có lượng dữ liệu lớn, bạn có thể sử dụng biến thể của Gradient Descent được gọi là stochastic gradient descent.

Trong biến thể này, quy trình Gradient Descent được mô tả ở trên được chạy, nhưng việc cập nhật các hệ số được thực hiện cho mỗi trường hợp đào tạo, thay vì ở cuối lô trường hợp.

Bước đầu tiên của quy trình yêu cầu việc thay đổi thứ tự của tập dữ liệu đào tạo. Điều này là để trộn lên thứ tự mà các cập nhật được thực hiện cho các hệ số. Bởi vì các hệ số được cập nhật sau mỗi trường hợp đào tạo, các cập nhật sẽ là ngẫu nhiên nhảy quanh, và cũng như hàm chi phí tương ứng. Bằng cách trộn lên thứ tự cho các cập nhật đối với các hệ số, nó tận dụng sự đi lại ngẫu nhiên này và tránh bị sao lạc hoặc bị kẹt.

Quy trình cập nhật cho các hệ số giống như mô tả ở trên, ngoại trừ việc chi phí không được tổng hợp qua tất cả các mẫu đào tạo, mà thay vào đó được tính toán cho một mẫu đào tạo.

Học tập có thể nhanh hơn với stochastic gradient descent cho các tập dữ liệu đào tạo lớn và thường chỉ cần một số lượng nhỏ các l ần đi qua tập dữ liệu để đạt được bộ các hệ số tốt hoặc đủ tốt, ví dụ từ 1 đến 10 lần đi qua tập dữ liệu.

2.6. Một số tips

Phần này liệt kê một số lời khuyên và mẹo để tận dụng tối đa thuật toán Gradient Descent cho máy học.

Vẽ Biểu Đồ Chi Phí theo Thời Gian: Thu thập và vẽ biểu đồ các giá trị chi phí được tính toán bởi thuật toán ở mỗi lần lặp. Mong đợi của một lần chạy Gradient Descent hiệu quả là giảm chi phí ở mỗi lần lặp. Nếu không giảm, hãy thử giảm tỷ lệ học của bạn.

Tỷ Lệ Học (Learning Rate): Giá trị tỷ lệ học là một giá trị thực nhỏ như 0.1, 0.001 hoặc 0.0001. Hãy thử các giá trị khác nhau cho vấn đề của bạn và xem cái nào hoạt động tốt nhất.

Thay Đổi Tỷ Lệ Học: Một cách ngẫu nhiên thay đổi tỷ lệ học theo thời gian có thể giúp cải thiện tốc độ học tập.

Thay Đổi Biến Đầu Vào: Thuật toán sẽ đạt được giá trị tối thiểu nhanh hơn nếu hình dạng của hàm chi phí không bị méo mó và biến dạng. Bạn có thể thực hiện điều này bằng cách đặt lại tất cả các biến đầu vào (X) cùng một phạm vi, chẳng hạn [0, 1] hoặc [-1, 1].

Ít Lần Đi Qua: Stochastic gradient descent thường không cần nhiều lần đi qua tập dữ liệu để hội tụ về một bộ các hệ số tốt hoặc đủ tốt.

Vẽ Biểu Đồ Trung Bình Chi Phí: Cập nhật cho từng trường hợp đào tạo có thể dẫn đến biểu đồ chi phí nhiễu theo thời gian khi sử dụng stochastic gradient descent. Lấy trung bình qua 10, 100 hoặc 1000 cập nhật có thể cho bạn cái nhìn tốt hơn về xu hướng học tập của thuật toán.

2.7. Tổng kết

Trong bài viết này, bạn đã tìm hiểu về Gradient Descent cho máy học. Bạn đã biết rằng:

Tối ưu hóa là một phần quan trọng của máy học.

Gradient Descent là một thủ tục tối ưu đơn giản mà bạn có thể sử dụng với nhiều thuật toán máy học khác nhau.

Batch Gradient Descent tham chiếu đến việc tính toán đạo hàm từ toàn bộ dữ liệu đào tạo trước khi tính toán một lần cập nhật.

Stochastic Gradient Descent tham chiếu đến việc tính toán đạo hàm từng mẫu đào tạo và tính toán cập nhật ngay lập tức.

3. Flatten

3.1. Code

```
# 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)
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 #(None, 8, 16)
layer_2.output_shape #(None, 128)
```

```
(None, 128)
```

3.2. Giải thích

Layer Flatten được sử dụng để "làm phẳng" (flatten) các đầu ra từ layer trước đó để chúng có thể được sử dụng làm đầu vào cho layer kế tiếp.

Ở đây:

• layer_1 là một layer Dense với 16 neurons và input_shape=(8, 8). Nghĩa là nó nhận đầu vào có kích thước (8, 8), chẳng hạn như một ma trận 8x8.

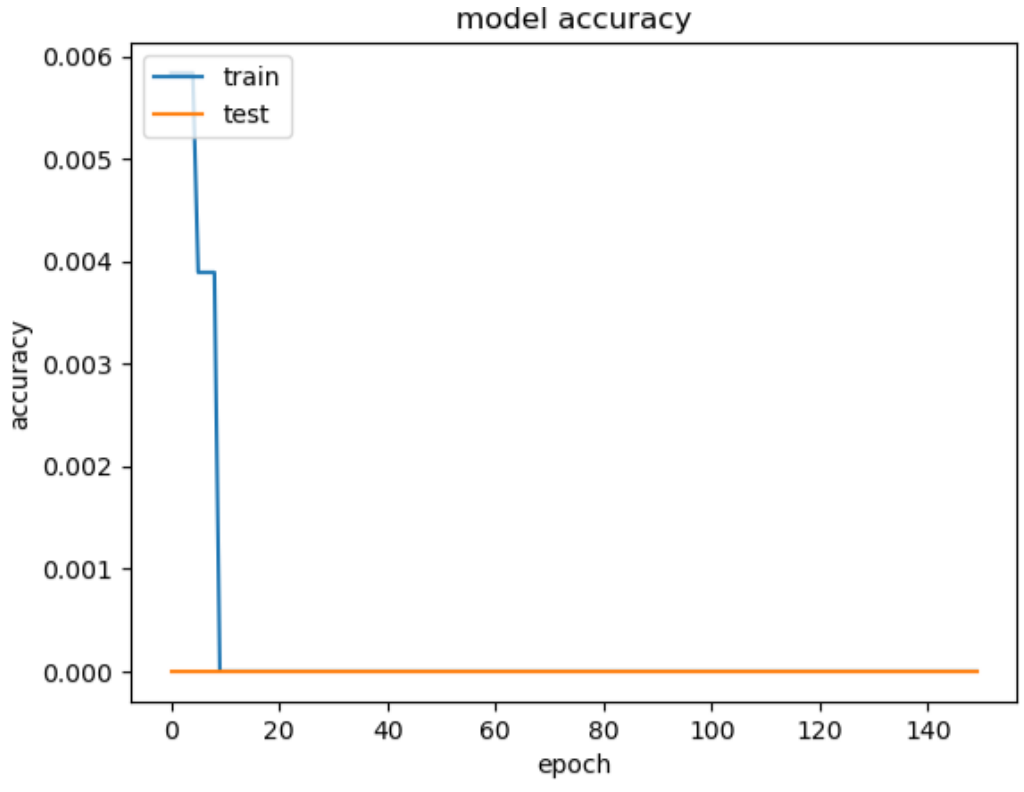
• layer_2 là một layer Flatten. Khi bạn thêm Flatten sau một layer có đầu ra là ma trận, nó sẽ chuyển ma trận đó thành một vector dài bằng cách "làm phẳng" ma trận. Trong trường hợp này, kích thước đầu vào của layer_2 là (None, 8, 16) và đầu ra của nó là (None, 128). Điều này nghĩa là mỗi ma trận (8, 16) đầu vào được chuyển thành một vector có độ dài 128.

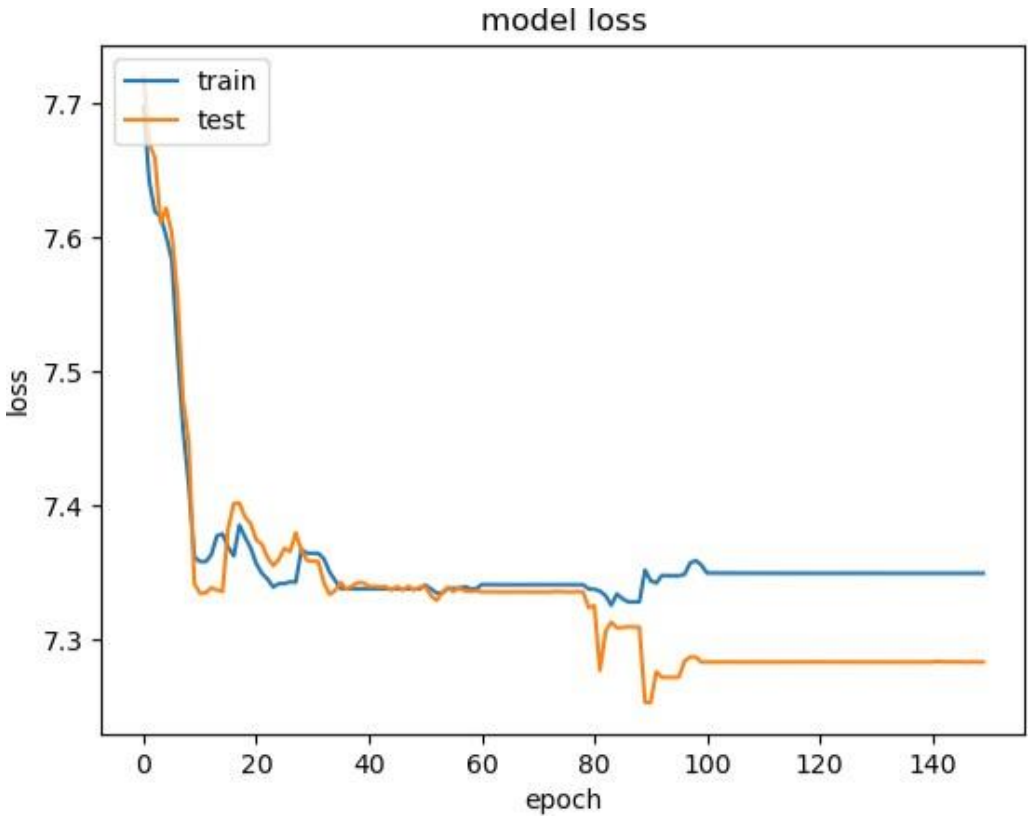
• None trong kích thước là một chiều linh hoạt, thường được sử dụng khi bạn không xác định kích thước của batch một cách cụ thể.

• Quá trình làm phẳng là quan trọng khi bạn chuyển từ các layer 2D (như Convolutional layers) sang các layer hoàn toàn kết nối (Dense layers) trong các mô hình deep learning. Các layer Dense yêu cầu đầu vào của chúng là vector một chiều, và Flatten giúp thực hiện điều này

4. 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

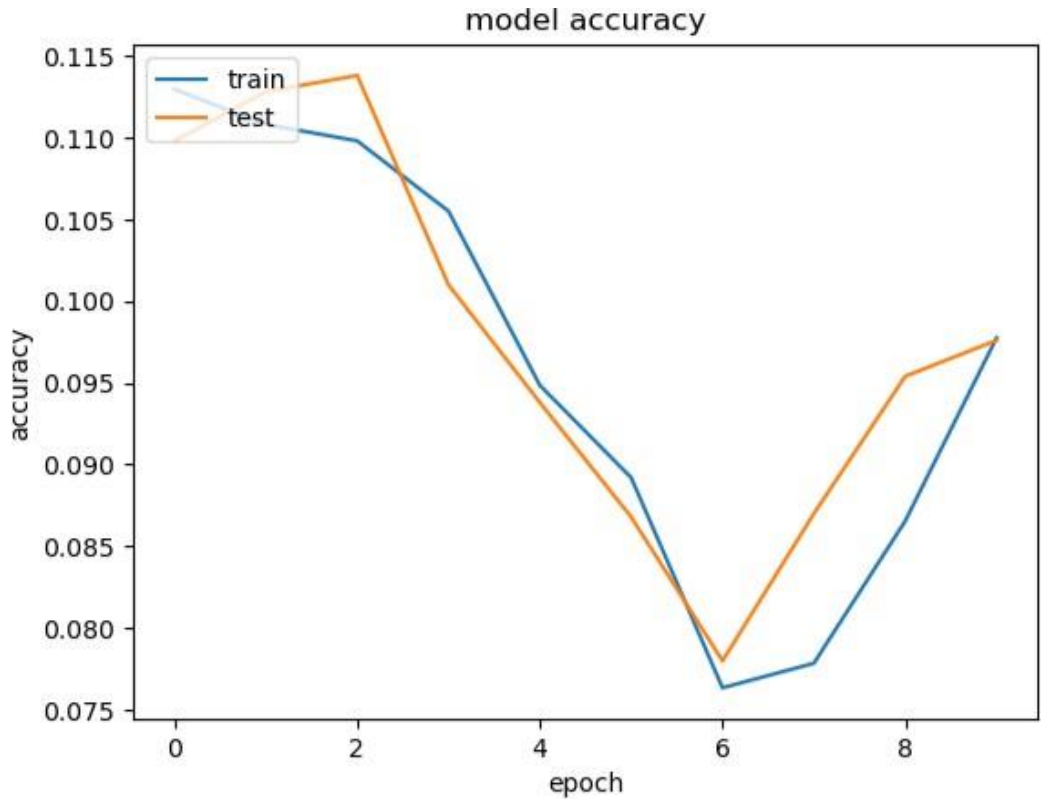
In [6]:
```
# Áp dụng khai báo layer_1 cho 5.1
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
import numpy as np

dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter=",")
#split into input X and output Y var
X = dataset[:,0:8]
Y = dataset[:,8]

#create model
model = Sequential()
layer_1 = Dense(16, input_shape=(8,8))
model.add(layer_1)

#Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
#Fit 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.xlabel('epoch')
plt.ylabel('accuracy')
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.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

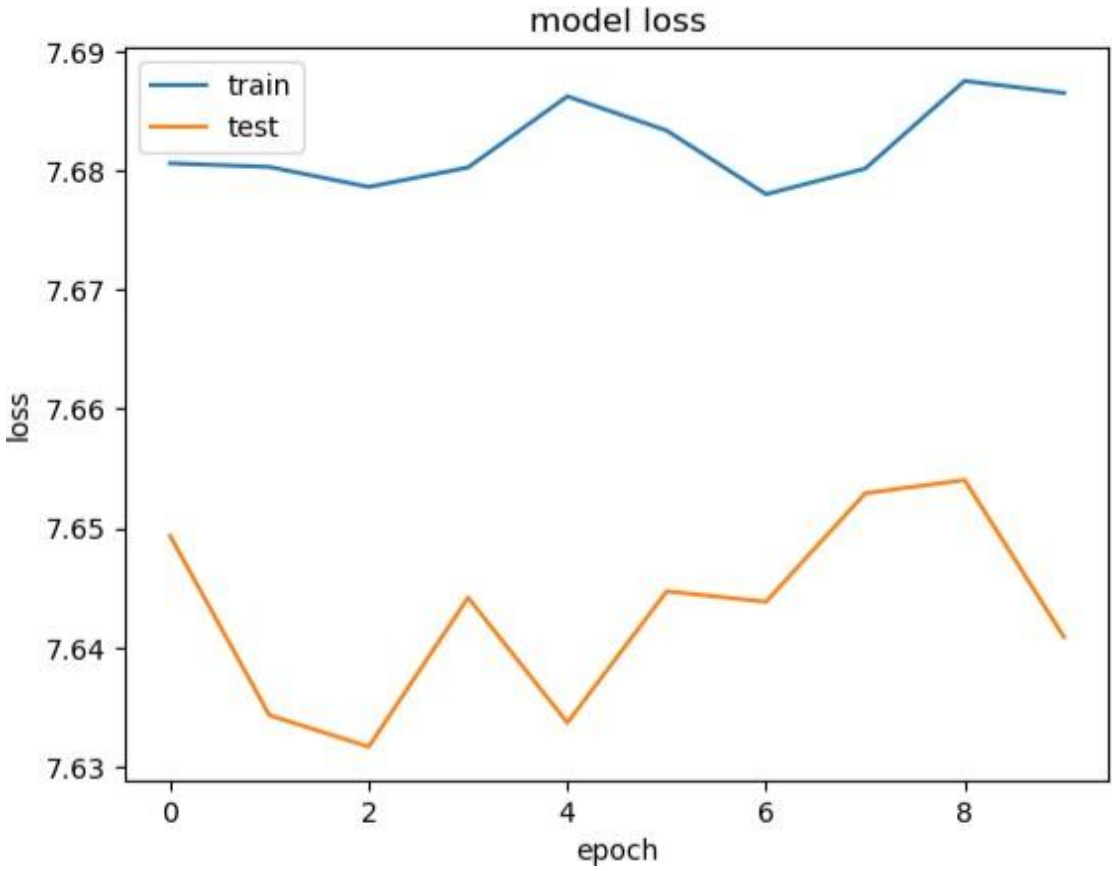




In [11]:
```
# Áp dụng cho 5.2
from keras.datasets import imdb
from keras import preprocessing
from keras.layers import Embedding, Dense, Flatten
from keras.models import Sequential
import matplotlib.pyplot as plt
import numpy as np

max_feature = 10000
maxlen = 20
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_feature)
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)

model = Sequential()
layer_1 = Dense(16, input_dim = maxlen)
model.add(layer_1)
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)
#summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
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.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

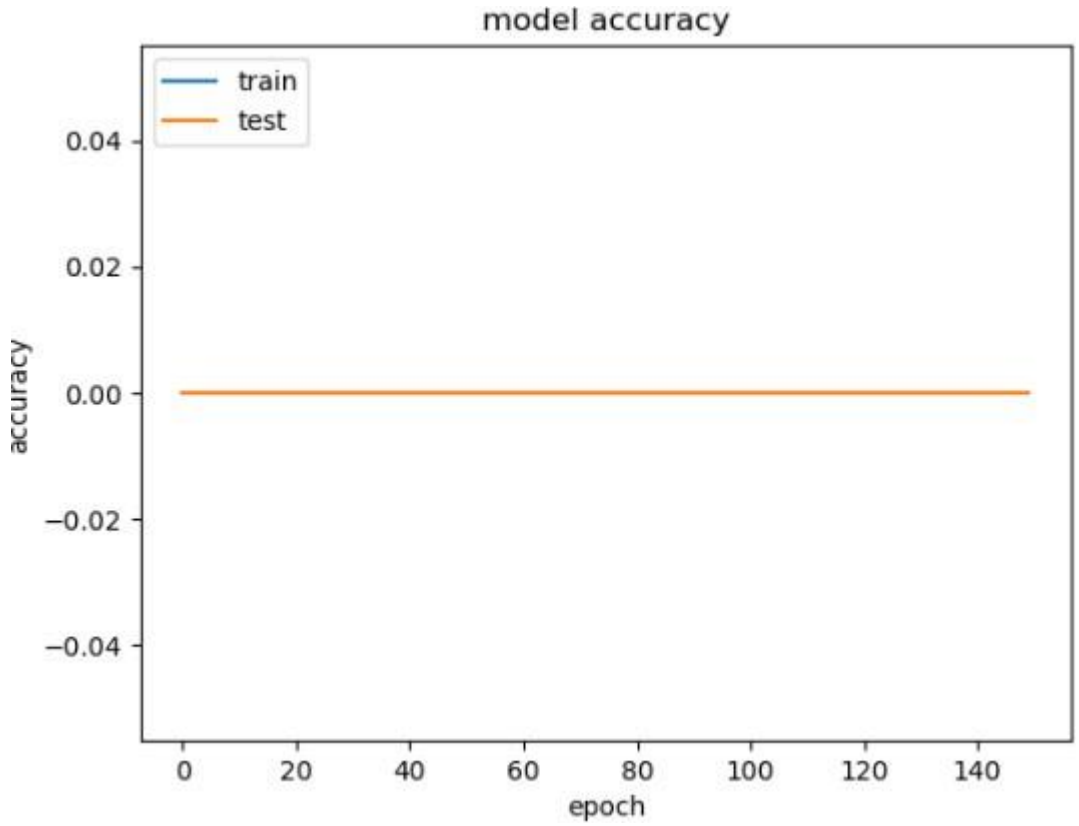
```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 16)                336

=================================================================
Total params: 336 (1.31 KB)
Trainable params: 336 (1.31 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
625/625 [==============================] - 3s 3ms/step - loss: 7.6806 - acc: 0.1129 - val_loss: 7.6494 - val_acc: 0.1098
Epoch 2/10
625/625 [==============================] - 1s 2ms/step - loss: 7.6803 - acc: 0.1108 - val_loss: 7.6343 - val_acc: 0.1128
Epoch 3/10
625/625 [==============================] - 1s 2ms/step - loss: 7.6786 - acc: 0.1098 - val_loss: 7.6317 - val_acc: 0.1138
Epoch 4/10
625/625 [==============================] - 1s 2ms/step - loss: 7.6802 - acc: 0.1055 - val_loss: 7.6442 - val_acc: 0.1010
Epoch 5/10
625/625 [==============================] - 2s 4ms/step - loss: 7.6862 - acc: 0.0949 - val_loss: 7.6337 - val_acc: 0.0938
Epoch 6/10
625/625 [==============================] - 3s 4ms/step - loss: 7.6833 - acc: 0.0892 - val_loss: 7.6447 - val_acc: 0.0868
Epoch 7/10
625/625 [==============================] - 3s 4ms/step - loss: 7.6780 - acc: 0.0764 - val_loss: 7.6438 - val_acc: 0.0780
Epoch 8/10
625/625 [==============================] - 3s 4ms/step - loss: 7.6802 - acc: 0.0778 - val_loss: 7.6529 - val_acc: 0.0870
Epoch 9/10
625/625 [==============================] - 2s 3ms/step - loss: 7.6875 - acc: 0.0865 - val_loss: 7.6540 - val_acc: 0.0954
Epoch 10/10
625/625 [==============================] - 1s 2ms/step - loss: 7.6865 - acc: 0.0978 - val_loss: 7.6409 - val_acc: 0.0976
```

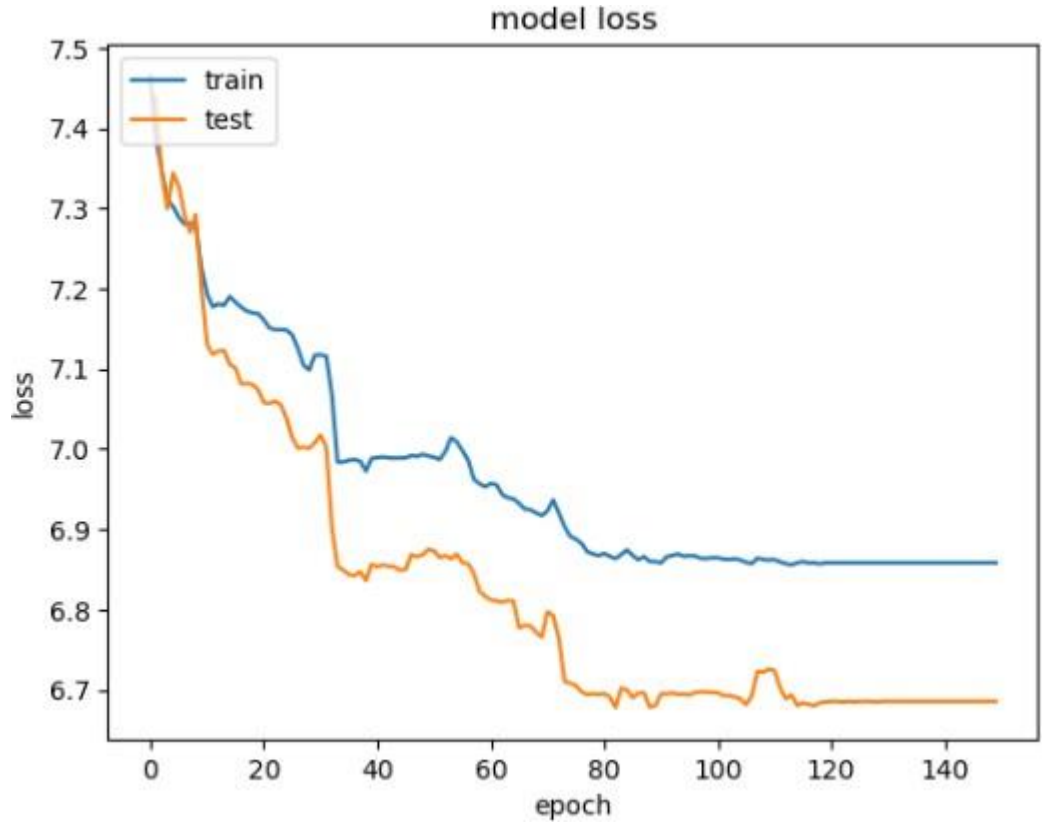




5. **Sử dụng lại Bài tập 5 và Bài tập 6 (Câu 3), Sinh viên thêm làm lượt 2,3 layer với số lượng neuron khác nhau. Chạy từng trường hợp và sử dụng các Biểu đồ output để so sánh các trường hợp với nhau khi chưa thêm layer với thêm layer mới**
   - Chạy lại với 5.1

2]:
```
# Áp dụng khai báo 2 layer (layer_1 và layer_2) cho 5.1
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
import numpy as np

dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter=",")
#split into input X and output Y var
X = dataset[:,0:8]
Y = dataset[:,8]

#create model
model = Sequential()
layer_1 = Dense(16, input_shape=(8,8))
model.add(layer_1)
# 64 đơn vị noron
layer_2 = Dense(64, activation='relu')
model.add(layer_2)

#Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
#Fit 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.xlabel('epoch')
plt.ylabel('accuracy')
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.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

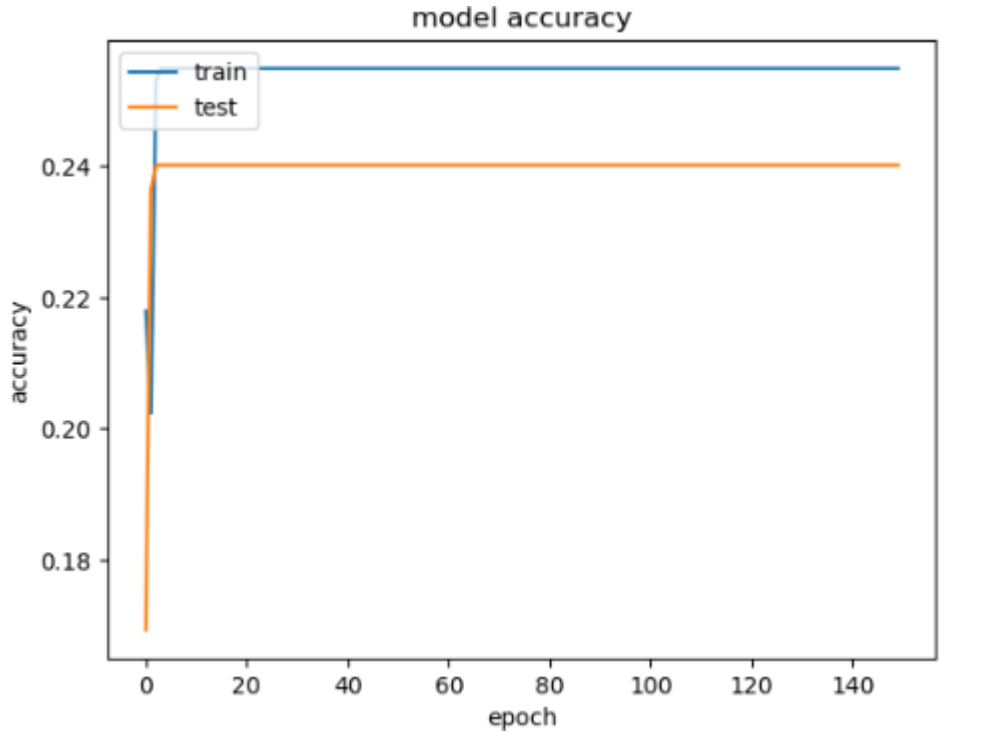

model accuracy
train
test
0.04
0.02
0.00
−0.02
−0.04
accuracy
0
20
40
60
80
100
120
140
epoch

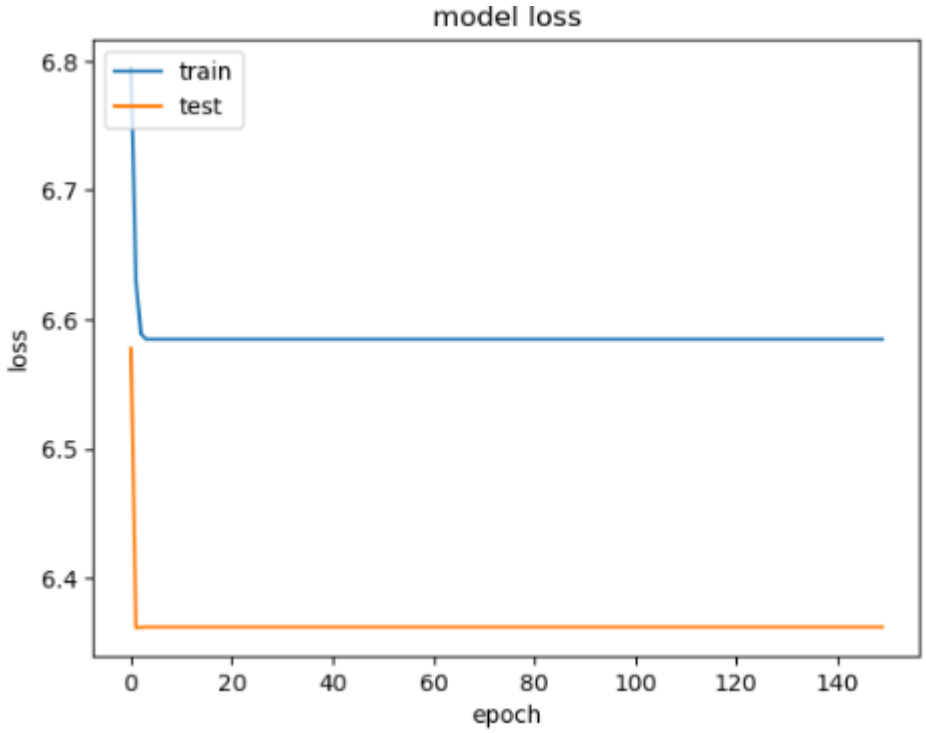

model loss
train
test
7.5
7.4
7.3
7.2
7.1
7.0
6.9
6.8
6.7
loss
0
20
40
60
80
100
120
140
epoch

In [1]:

```
# Áp dụng khai báo 3 layer (layer_1 và layer_2 và layer_3) cho 5.1
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
import numpy as np

dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter=",")
#split into input X and output Y var
X = dataset[:,0:8]
Y = dataset[:,8]

#create model
model = Sequential()
layer_1 = Dense(16, input_shape=(8,8))
model.add(layer_1)
# 64 đơn vị noron
layer_2 = Dense(64, activation='relu')
model.add(layer_2)
layer_3 = Dense(8)
model.add(layer_3)

#Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
#Fit 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.xlabel('epoch')
plt.ylabel('accuracy')
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.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

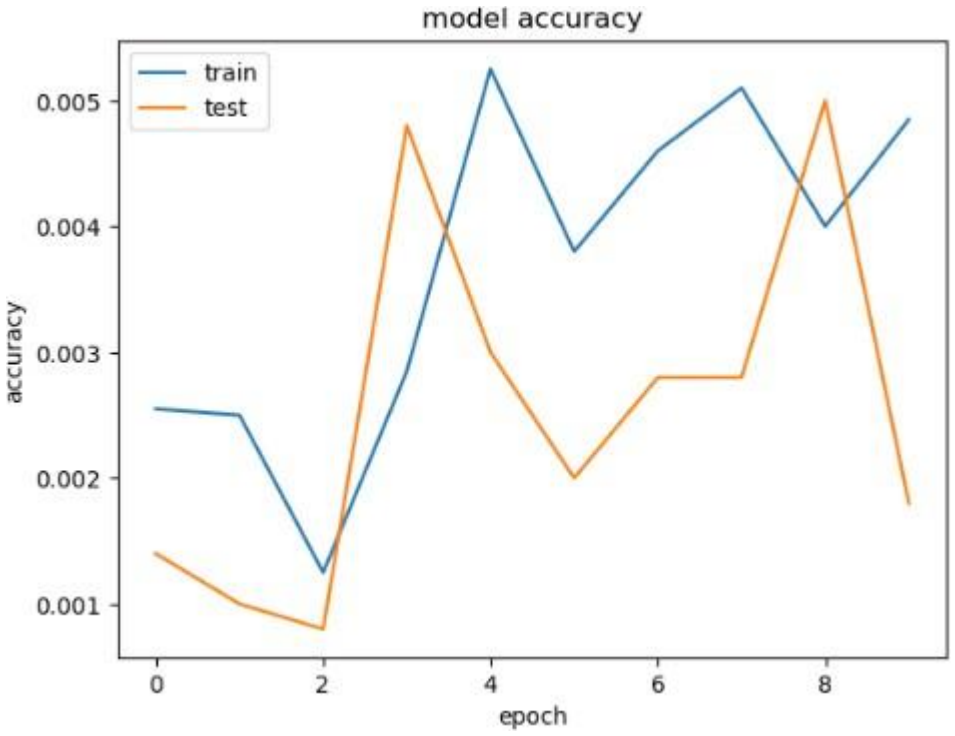
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

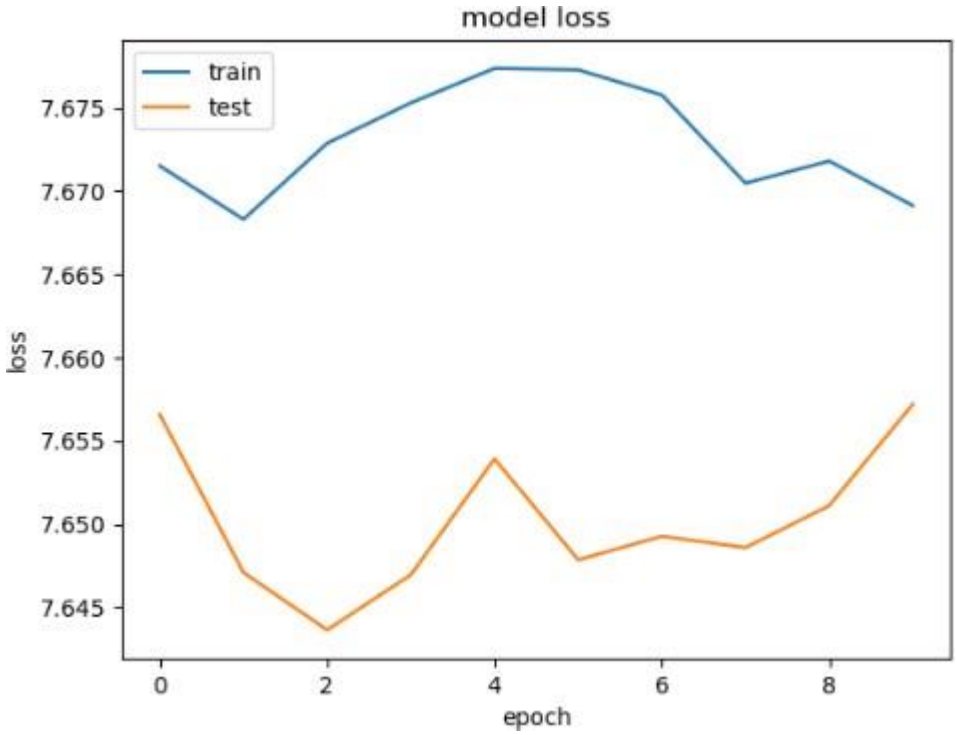




- Chạy lại với 5.2

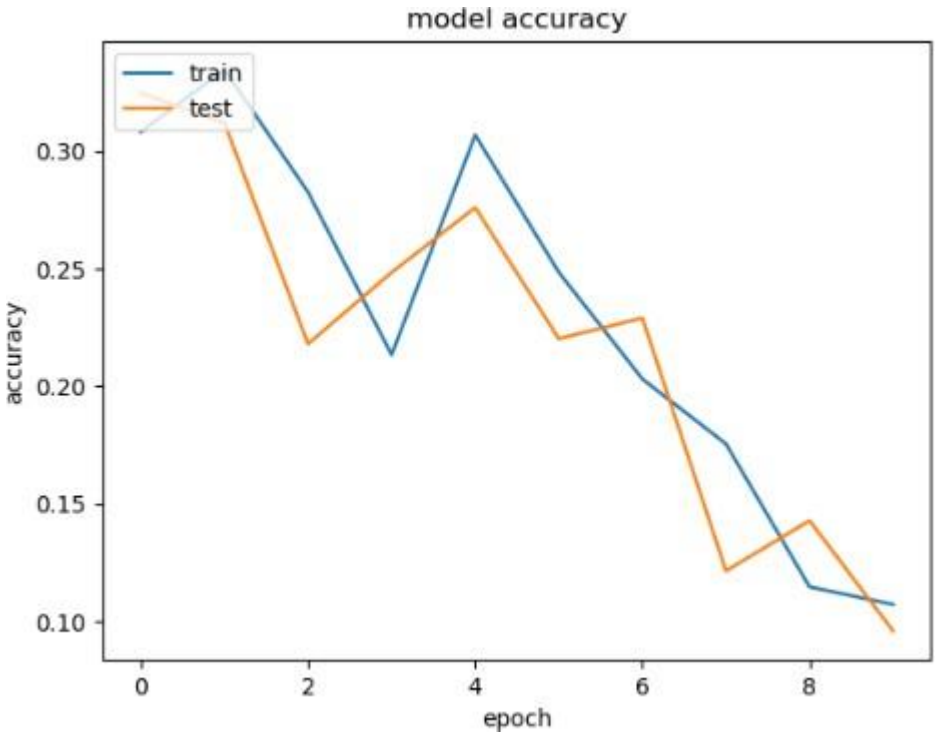
In [*]:
```
# Áp dụng cho 5.2 với 2 layer (layer_1 và layer_2)
from keras.datasets import imdb
from keras import preprocessing
from keras.layers import Embedding, Dense, Flatten
from keras.models import Sequential
import matplotlib.pyplot as plt
import numpy as np

max_feature = 10000
maxlen = 20
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_feature)
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)

model = Sequential()
layer_1 = Dense(16, input_dim = maxlen)
model.add(layer_1)
layer_2 = Dense(64, activation='relu')
model.add(layer_2)
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)
#summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
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.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

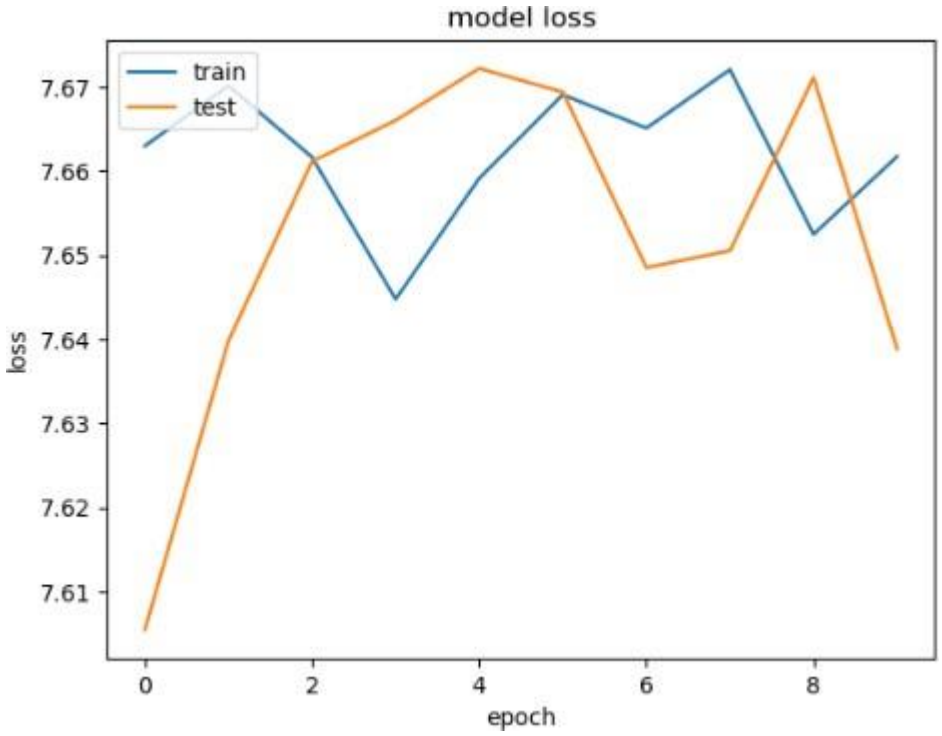

model accuracy
train
test
0.005
0.004
0.003
0.002
0.001
accuracy
0
2
4
6
8
epoch


model loss
train
test
7.675
7.670
7.665
7.660
7.655
7.650
7.645
loss
0
2
4
6
8
epoch

In [3]:
```
# Áp dụng cho 5.2 với 3 layer (layer_1 và layer_2 và layer_3)
from keras.datasets import imdb
from keras import preprocessing
from keras.layers import Embedding, Dense, Flatten
from keras.models import Sequential
import matplotlib.pyplot as plt
import numpy as np

max_feature = 10000
maxlen = 20
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_feature)
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)

model = Sequential()
layer_1 = Dense(16, input_dim = maxlen)
model.add(layer_1)
layer_2 = Dense(64, activation='relu')
model.add(layer_2)
layer_3 = Dense(8)
model.add(layer_3)
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)
#summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
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.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
Model: "sequential_2"
```





6. Drop out

- Khái niệm Drop out:

- Dropout là một kỹ thuật chính trong mạng nơ-ron sử dụng để ngăn chặn việc quá khớp (overfitting) trong quá trình đào tạo mô hình. Kỹ thuật này được áp dụng bằng cách ngẫu nhiên "tắt" một số đơn vị nơ-ron trong mạng trong mỗi lần đào tạo, điều này có nghĩa là chúng không tham gia vào quá trình tính toán gradient và cập nhật trọng số trong một lượt đào tạo cụ thể. Điều này giúp mô hình trở nên tổng quát hóa tốt hơn và giảm nguy cơ quá khớp.

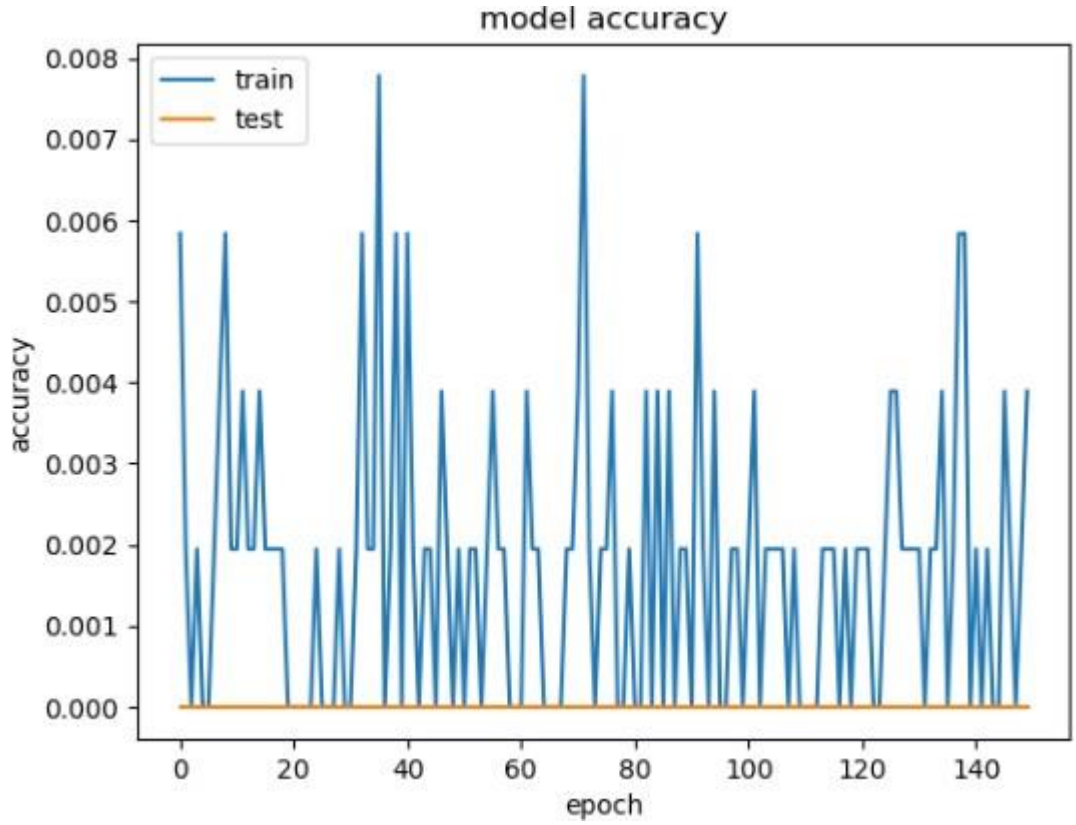
- Khi sử dụng Dropout, bạn chỉ định tỷ lệ (thường nằm trong khoảng từ 0,0 đến 1,0) để xác định xác suất một đơn vị nơ-ron bị "tắt" trong mỗi lần đào tạo. Ví dụ, nếu bạn đặt tỷ lệ là 0,5, thì một nửa số đơn vị nơron sẽ bị tắt một cách ngẫu nhiên trong mỗi lượt đào tạo.

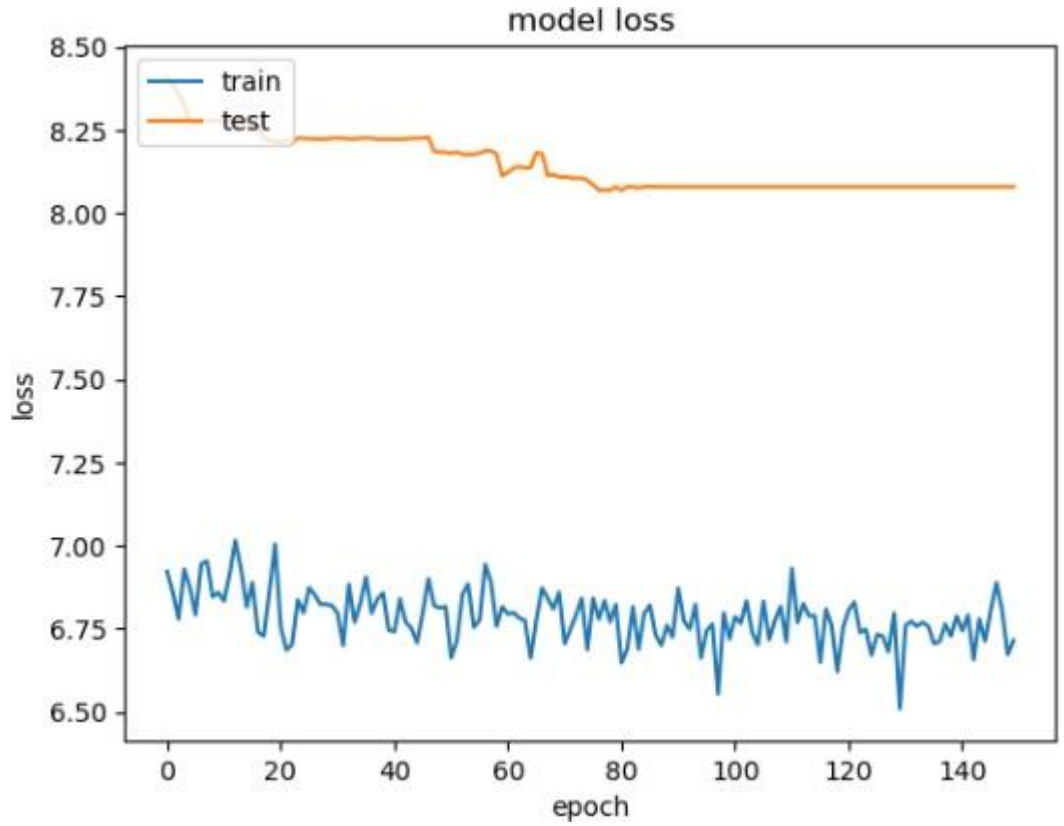
- Áp dụng với mô hình 1 layer

In [6]:

```
# Áp dụng dropout với 1 model layer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
import matplotlib.pyplot as plt
import numpy as np

dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter=",")
#split into input X and output Y var
X = dataset[:,0:8]
Y = dataset[:,8]

#create model
model = Sequential()
layer_1 = Dense(16, input_shape=(8,8))
model.add(layer_1)
model.add(Dropout(0.5))


#Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
#Fit 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.xlabel('epoch')
plt.ylabel('accuracy')
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.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

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

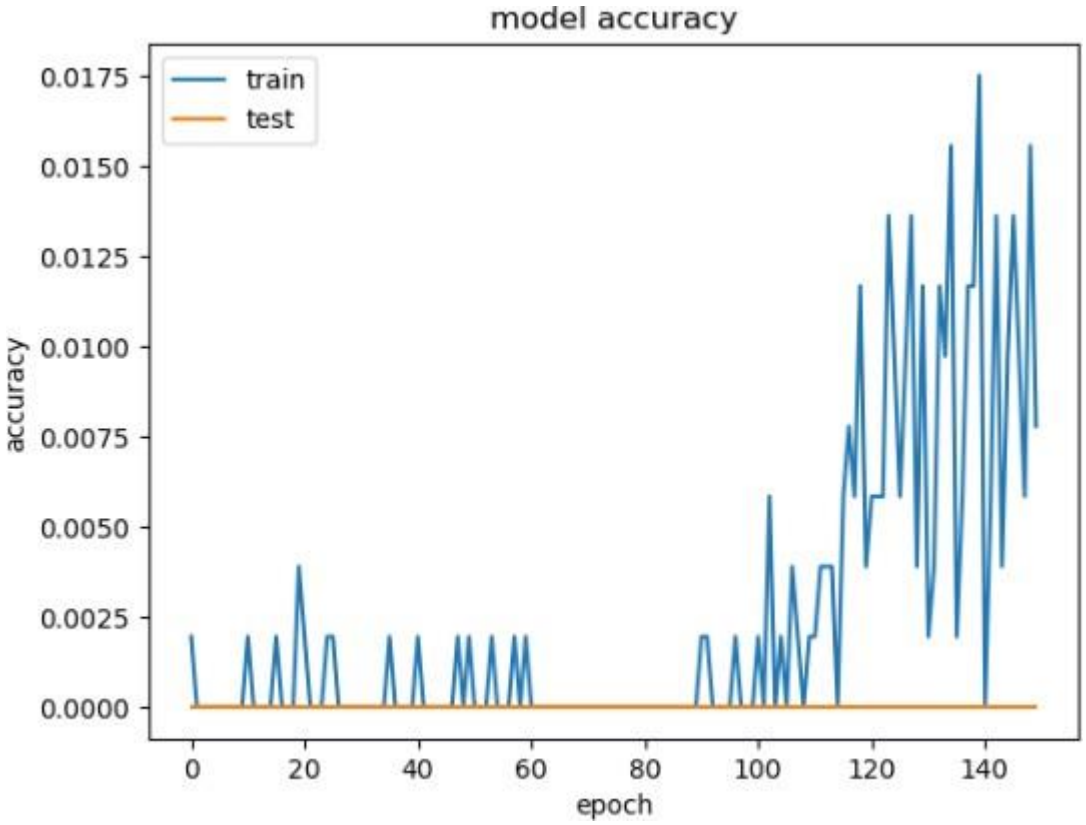


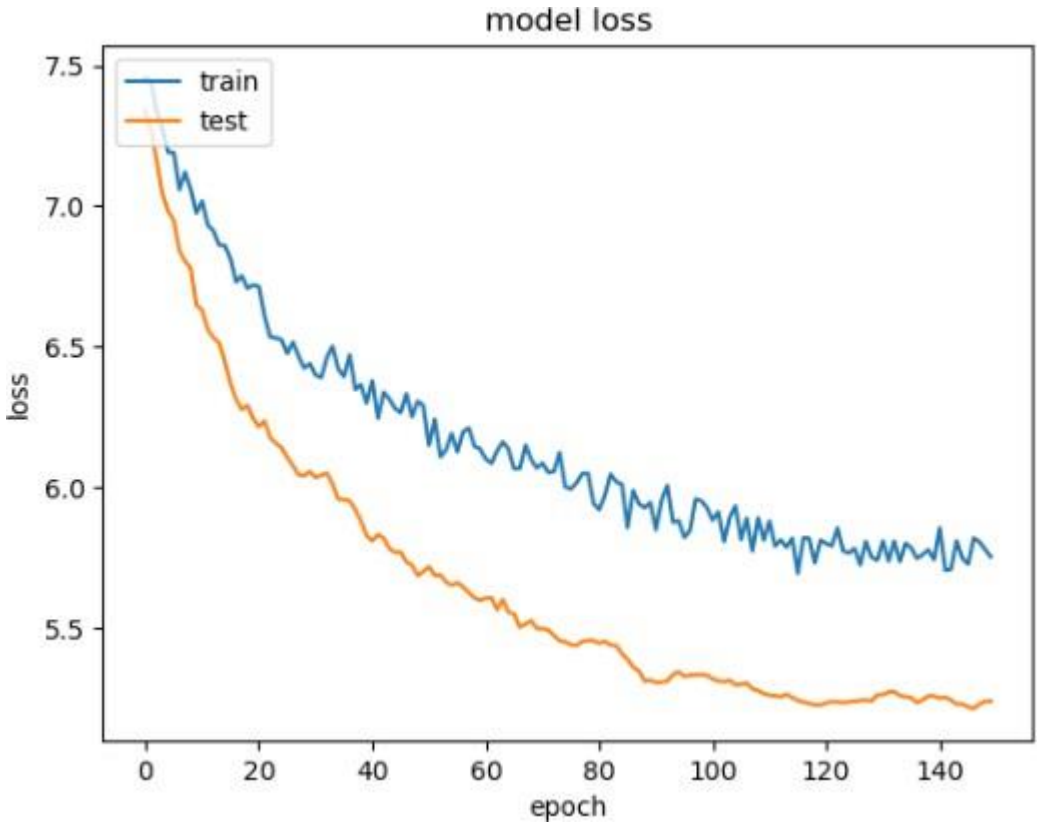


- Áp dụng với mô hình 2 layer

In [*]:

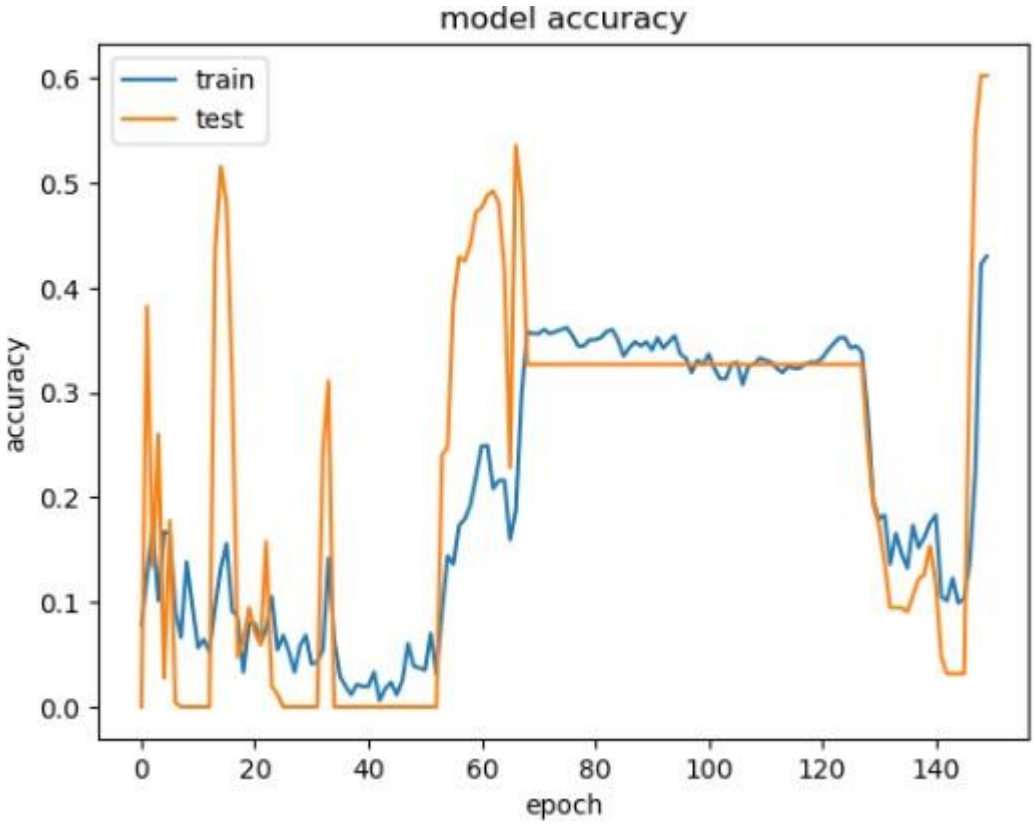
```
# Áp dụng dropout với 2 model layer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
import matplotlib.pyplot as plt
import numpy as np

dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter=",")
#split into input X and output Y var
X = dataset[:,0:8]
Y = dataset[:,8]

#create model
model = Sequential()
layer_1 = Dense(16, input_shape=(8,8))
model.add(layer_1)
model.add(Dropout(0.5))
# # 64 đơn vị noron
layer_2 = Dense(64, activation='relu')
model.add(layer_2)


#Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
#Fit 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.xlabel('epoch')
plt.ylabel('accuracy')
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.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

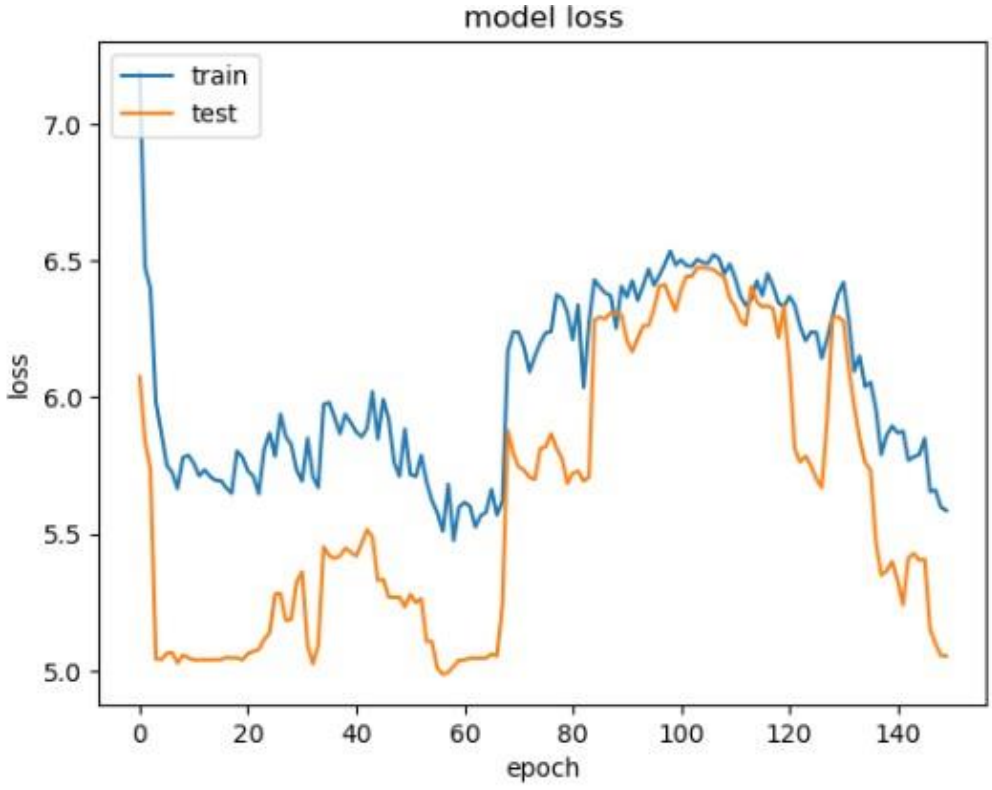




- Áp dụng mô hình 3 layer

In [*]:
```
# Áp dụng dropout với 3 model layer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
import matplotlib.pyplot as plt
import numpy as np

dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter=",")
#split into input X and output Y var
X = dataset[:,0:8]
Y = dataset[:,8]

#create model
model = Sequential()
layer_1 = Dense(16, input_shape=(8,8))
model.add(layer_1)
model.add(Dropout(0.5))
# # 64 đơn vị noron
layer_2 = Dense(64, activation='relu')
model.add(layer_2)
layer_3 = Dense(8)
model.add(layer_3)

#Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
#Fit 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.xlabel('epoch')
plt.ylabel('accuracy')
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.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```


model accuracy
train
test
accuracy
epoch
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
20
40
60
80
100
120
140


model loss
train
test
loss
epoch
5.0
5.5
6.0
6.5
7.0
0
20
40
60
80
100
120
140