Họ và tên: Quách Xuân Phúc

MSV: B20DCCN513

7.1

1. Sample:

- Sample là một dòng dữ liệu trong tập dữ liệu.
- Nó bao gồm các đầu vào được đưa vào thuật toán và một đầu ra được sử dụng để so sánh với dự đoán và tính toán lỗi.
- Ví dụ code:

```
# Một ví dụ về một sample dữ liệu
sample = (features, target)
```

2. Iteration:

- Iteration là một bước lặp trong quá trình huấn luyện mô hình.
- Mỗi iteration thường tương ứng với việc xử lý một batch hoặc một sample.
- Iteration có thể là một lần cập nhật trọng số của mô hình.
- Ví dụ code:

```
for iteration in range(total_iterations):
    # Thực hiện một lần lặp
    process_data_and_update_model(iteration)
```

3. Epoch:

- Epoch là một vòng lặp hoàn toàn qua toàn bộ tập dữ liệu huấn luyện.
- Mỗi epoch đảm bảo rằng mỗi sample trong tập dữ liệu đã được sử dụng để cập nhật trọng số của mô hình ít nhất một lần.
- Ví dụ code:

```
for epoch in range(num_epochs):
    for batch in data_batches:
        process_batch_and_update_model(batch)
```

4. Batch:

- Batch là một tập hợp gồm nhiều sample được sử dụng để cập nhật trọng số của mô hình cùng một lúc.

- Batch size là số lượng sample trong mỗi batch, và nó là một siêu tham số quyết định bởi người huấn luyện.
- Có ba dạng phổ biến của gradient descent dựa trên batch size:
  - Batch Gradient Descent: Khi tất cả các mẫu huấn luyện được sử dụng để tạo một batch, thuật toán được gọi là batch gradient descent. (Batch Size = Kích thước của tập huấn luyện)
  - Stochastic Gradient Descent (SGD): Khi kích thước batch bằng 1, tức là mỗi mẫu được sử dụng riêng lẻ để cập nhật mô hình, thuật toán được gọi là stochastic gradient descent.
  - Mini-Batch Gradient Descent: Khi kích thước batch nằm trong khoảng từ 1 đến kích thước của tập huấn luyện, thuật toán được gọi là mini-batch gradient descent. Đây là dạng phổ biến nhất của gradient descent trong thực tế.
  - Trong trường hợp của mini-batch gradient descent, các kích thước batch phổ biến bao gồm 32, 64 và 128 mẫu. Bạn có thể thấy các giá trị này được sử dụng trong các mô hình trong tài liệu và hướng dẫn.
- Nếu tập dữ liệu không chia hết cho kích thước batch:
  - Điều này thường xảy ra khi huấn luyện mô hình. Điều này đơn giản là có nghĩa rằng batch cuối cùng sẽ có ít mẫu hơn so với các batch khác.
  - Hoặc bạn có thể loại bỏ một số mẫu khỏi tập dữ liệu hoặc thay đổi kích thước batch để đảm bảo số lượng mẫu trong tập dữ liệu chia hết cho kích thước batch.
- Ví dụ code:

```
# Chia tập dữ liệu thành các batch
batches = split_data_into_batches(data, batch_size)
```

5. Update weights khi nào?

- Trọng số của mô hình được cập nhật sau khi một batch hoặc một sample đã được sử dụng để tính toán lỗi.
- Quá trình cập nhật trọng số xảy ra sau khi tính toán độ lỗi và sử dụng gradient descent để điều chỉnh các trọng số.

6. Ví dụ với data có 200 mẫu và batch size là 5, và chạy với 100 epoch:

- Số lượng iteration sẽ là: 200 (số mẫu) / 5 (batch size) = 40 iteration trong mỗi epoch.
- Tổng số lần cập nhật trọng số sẽ là: 40 (iteration mỗi epoch) * 100 (số epoch) = 4000 lần cập nhật trọng số trong toàn quá trình huấn luyện.

7.2 Stochastic Gradient Descent, hoặc SGD:

- Stochastic Gradient Descent, viết tắt là SGD, là một thuật toán tối ưu hóa được sử dụng trong quá trình huấn luyện mô hình máy học và deep learning. Thuật toán này thuộc họ các thuật toán Gradient Descent, nhưng có một số đặc điểm khác biệt.
- Ở SGD, mỗi lần cập nhật trọng số được thực hiện dựa trên một điểm dữ liệu duy nhất được chọn ngẫu nhiên từ tập dữ liệu đào tạo. Điều này làm cho quá trình cập nhật trở nên ngẫu nhiên và nhanh chóng, giảm độ phức tạp tính toán so với Gradient Descent truyền thống.
- Cụ thể, quá trình cập nhật trọng số trong SGD có thể được mô tả như sau:
  - Chọn ngẫu nhiên một sample từ tập dữ liệu đào tạo.
  - Tính gradient của hàm mất mát tại điểm này.
  - Cập nhật trọng số dựa trên gradient tính được.
- Thuật toán này được gọi là "stochastic" (ngẫu nhiên) bởi vì mỗi bước cập nhật được thực hiện trên một sample ngẫu nhiên, không theo một thứ tự cụ thể.
- SGD là một thuật toán rất phổ biến và được sử dụng rộng rãi trong nhiều mô hình học máy khác nhau, bao gồm:
  - Logistic Regression
  - Linear Regression
  - Neural Networks
  - Support Vector Machines
- Ưu điểm của SGD bao gồm:
  - Sự nhanh chóng và khả năng xử lý các tập dữ liệu lớn. Tuy nhiên, do tính ngẫu nhiên của quá trình, nó có thể làm giảm độ chính xác của mô hình và tạo ra độ dao động trong quá trình học.
  - Đơn giản và dễ dàng thực hiện
- Nhược điểm:
  - Có thể nhạy cảm với các giá trị khởi tạo tham số.
  - Có thể không hội tụ đến giá trị tối ưu của hàm chi phí.
- Để khắc phục nhược điểm của SGD, người ta thường sử dụng các kỹ thuật khác nhau như:
  - Sử dụng các giá trị khởi tạo tham số khác nhau.
  - Sử dụng các hàm chi phí khác nhau.
  - Sử dụng các kỹ thuật điều chỉnh độ học (learning rate).
- Có các biến thể của SGD như Mini-Batch Gradient Descent (sử dụng một số lượng nhỏ các mẫu thay vì một mẫu đơn lẻ) và Batch Gradient Descent (sử dụng toàn bộ tập dữ liệu để tính gradient). Mỗi biến thể có những ưu điểm và nhược điểm riêng.
- Ví dụ code:

```python
import numpy as np

# Tạo dữ liệu giả định
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Thêm cột bias (đội hệ số tự do) vào X
X_b = np.c_[np.ones((100, 1)), X]

# Khởi tạo trọng số ban đầu ngẫu nhiên
theta = np.random.randn(2, 1)

# Thiết lập siêu tham số SGD
learning_rate = 0.01
n_iterations = 1000

# Huấn luyện mô hình bằng SGD
for iteration in range(n_iterations):
    random_index = np.random.randint(100)  # Lựa chọn ngẫu nhiên một sample
    xi = X_b[random_index:random_index+1]
    yi = y[random_index:random_index+1]
    gradients = 2 * xi.T.dot(xi.dot(theta) - yi)
    theta = theta - learning_rate * gradients

# Kết quả cuối cùng là trọng số đã được tối ưu hóa bằng SGD
print("Trọng số tối ưu:", theta)
```

```
Trọng số tối ưu: [[4.35267055]
 [3.0521733 ]]
```

7.3

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

```
(None, 8, 16)
(None, 128)
```

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:

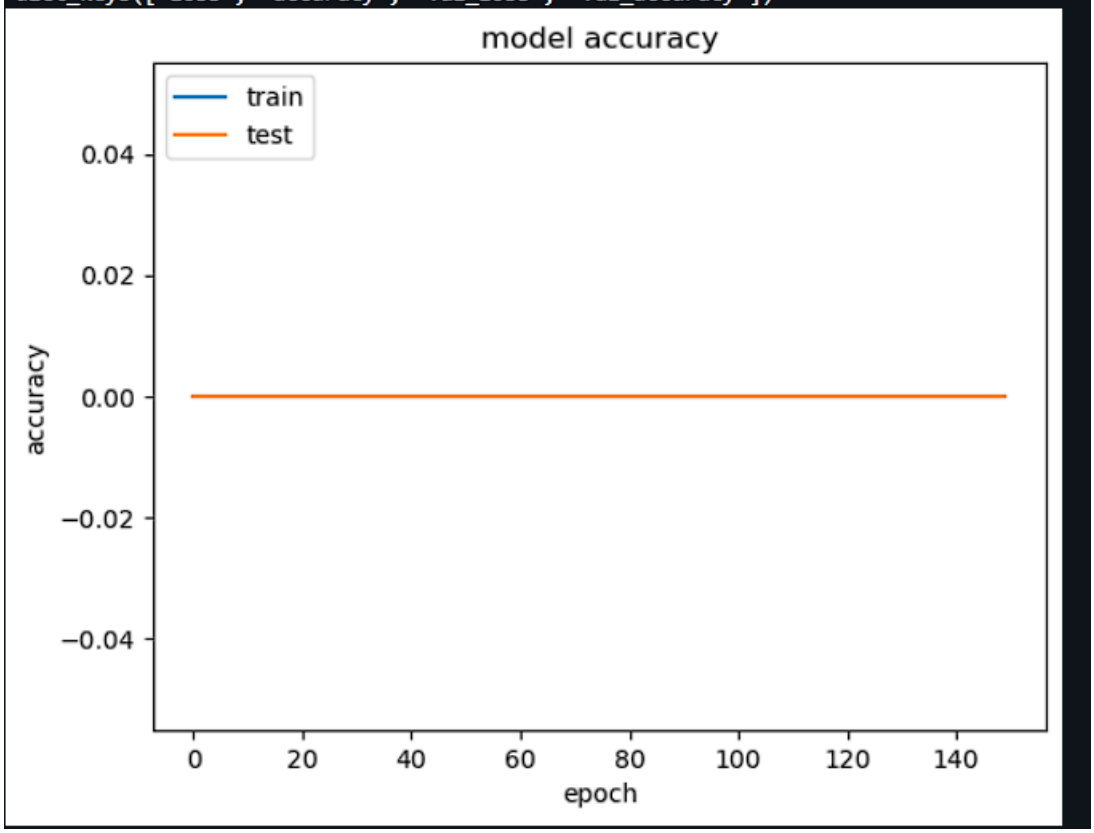
- layer_1 là một layer Dense với 16 neurons và input_shape=(8, 8). Nghĩa là nó nhận đầu vào có kích thước (8, 8), chẳng hạn như một ma trận 8x8.
- layer_2 là một layer Flatten. Khi bạn thêm Flatten sau một layer có đầu ra là ma trận, nó sẽ chuyển ma trận đó thành một vector dài bằng cách "làm phẳng" 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.

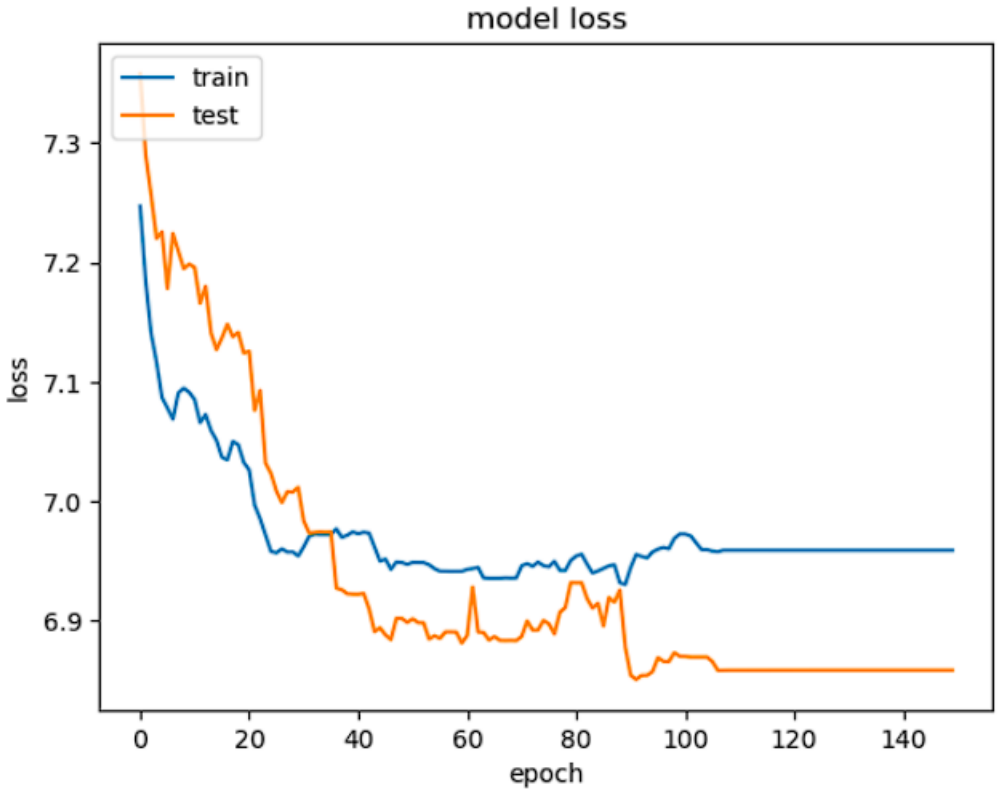
7.4

Áp dụng kiểu khai báo layer_1, model.add(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
# 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()
layer_1 = Dense(16, input_shape=(8,8))
model.add(layer_1)
# 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'])
```

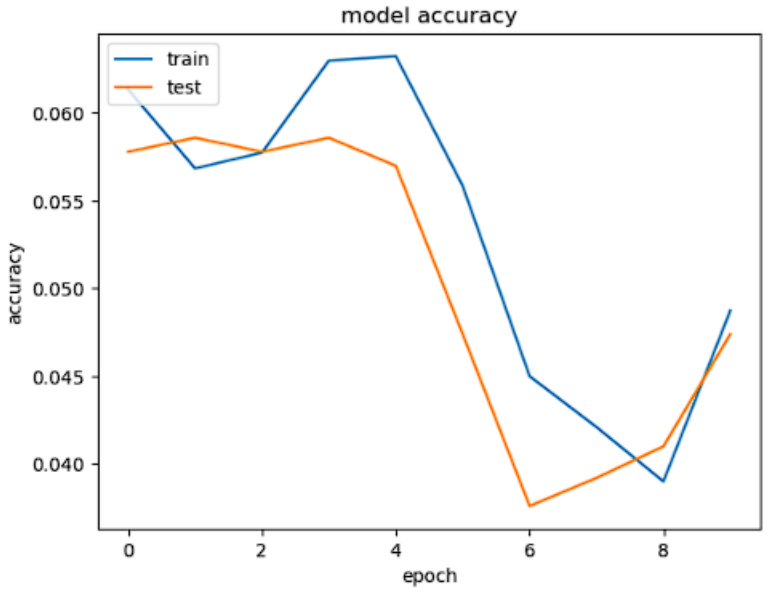


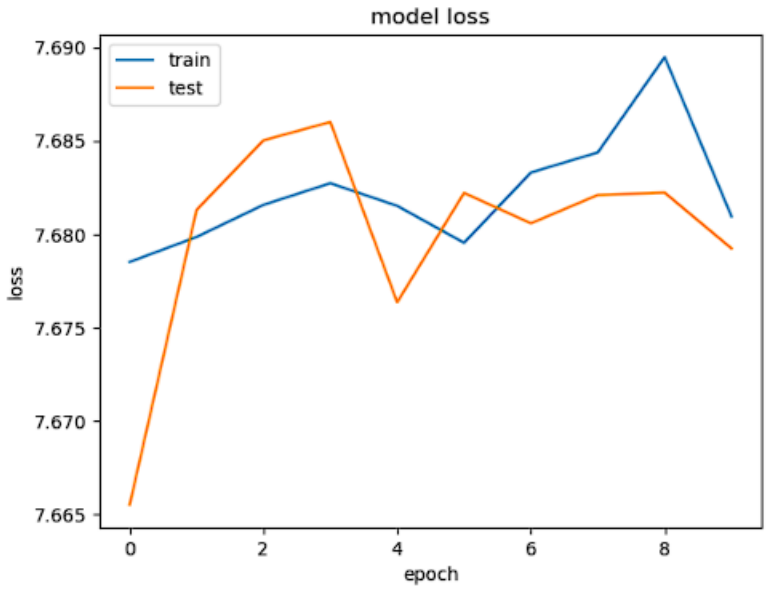


Áp dụng kiểu khai báo layer_1, model.add(layer_1) cho 5.2

```
from keras.datasets import imdb
from keras import preprocessing
from keras.layers import Embedding
from keras.models import Sequential
from keras.layers import Dense, Flatten
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()
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)

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_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_14 (Dense)            (None, 16)                336

=================================================================
Total params: 336 (1.31 KB)
Trainable params: 336 (1.31 KB)
Non-trainable params: 0 (0.00 Byte)
```


model accuracy
train
test
0.060
0.055
0.050
0.045
0.040
accuracy
0
2
4
6
8
epoch

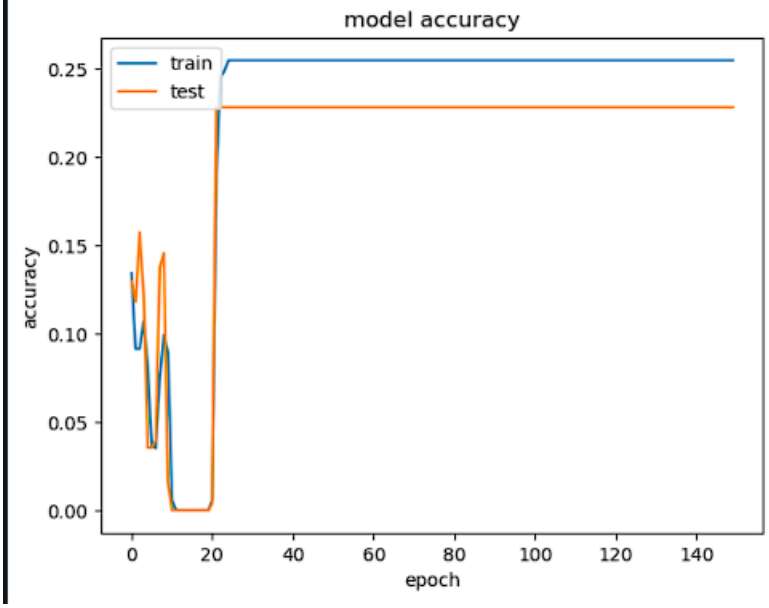

model loss
train
test
7.690
7.685
7.680
7.675
7.670
7.665
loss
0
2
4
6
8
epoch

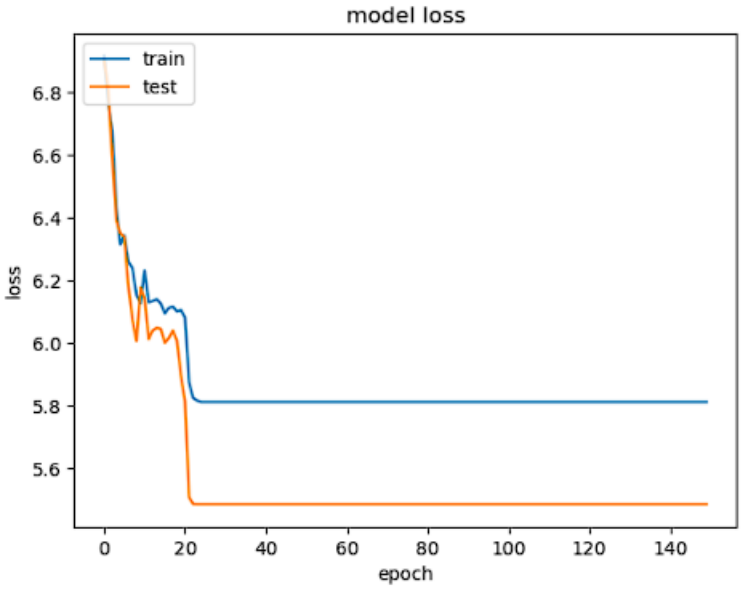
7.5

* Áp dụng 2 layer cho 5.1

```
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()
layer_1 = Dense(16, input_shape=(8,))
model.add(layer_1)
layer_2 = Dense(8, activation='relu')
model.add(layer_2)

# 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'])
```

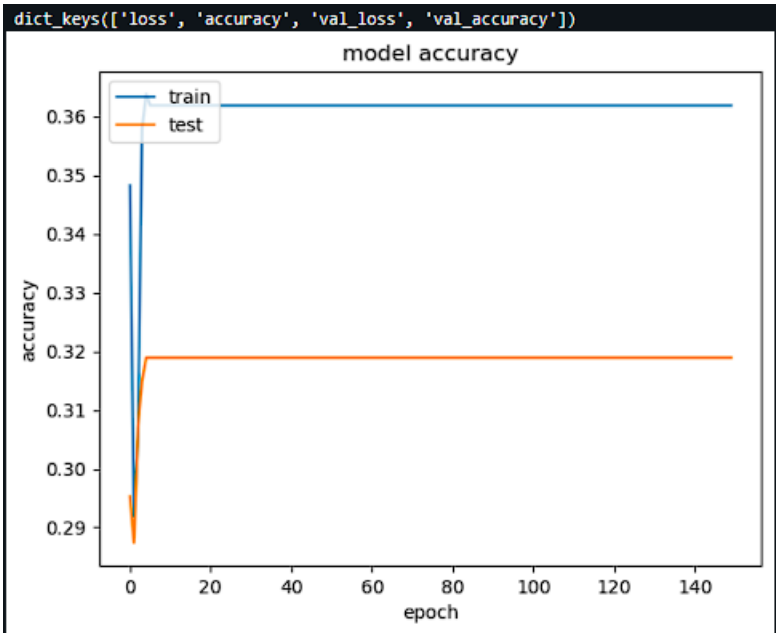


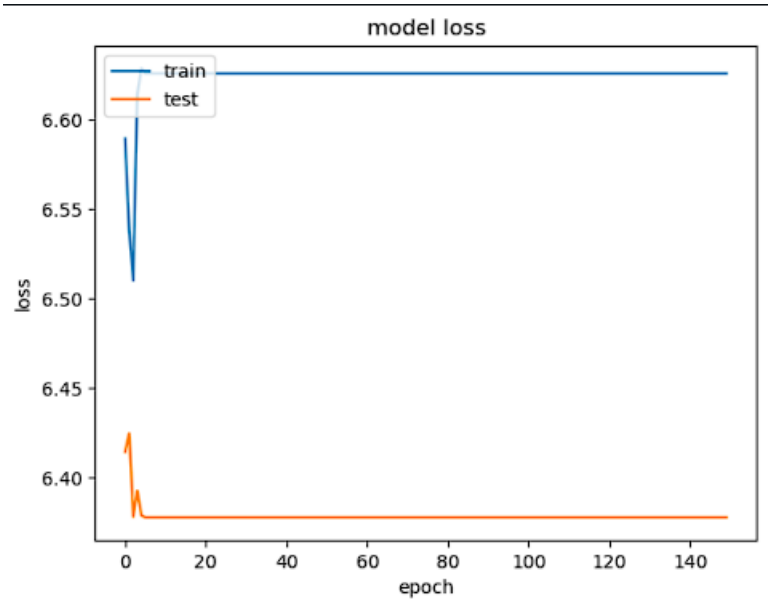


Dựa vào các biểu đồ output, có thể thấy rằng model với 2 layer có độ chính xác cao hơn so với model với 1 layer. Tuy nhiên, độ chính xác của model với 2 layer chỉ cao hơn một chút so với model với 1 layer. Ngoài ra, model với 2 layer cũng có độ mất mát (loss) thấp hơn so với model với 1 layer.

* Áp dụng 3 layer cho 5.1

```
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()
layer_1 = Dense(16, input_shape=(8,8))
model.add(layer_1)
layer_2 = Dense(8, activation='relu')
model.add(layer_2)
layer_3 = Dense(4, activation='relu')
model.add(layer_3)

# 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()
```





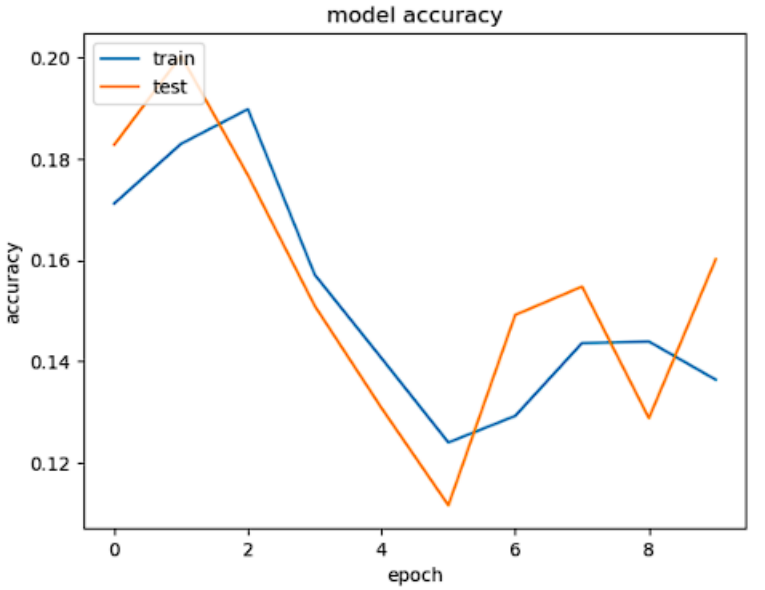
Dựa vào các biểu đồ output, có thể thấy rằng model với 3 layer có độ chính xác cao hơn so với model với 1 layer. Tuy nhiên, độ chính xác của model với 3 layer chỉ cao hơn một chút so với model với 1 layer. Ngoài ra, model với 3 layer cũng có độ mất mát (loss) thấp hơn so với model với 1 layer.

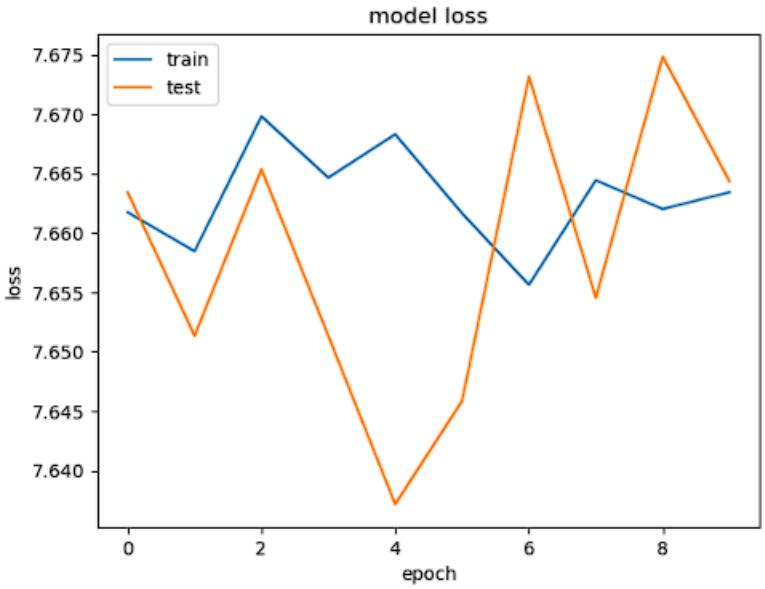
Qua việc so sánh các trường hợp với nhau khi 1 layer, 2 layer và 3 layer, có thể thấy rằng model với số lượng layer càng nhiều thì độ chính xác càng cao. Tuy nhiên, việc sử dụng nhiều layer hơn sẽ khiến cho model trở nên phức tạp hơn và khó huấn luyện hơn.

* Áp dụng 2 layer cho 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 Dense, Flatten
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()
layer_1 = Dense(16, input_dim=maxlen)
model.add(layer_1)
layer_2 = Dense(8, 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)

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_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_16 (Dense)            (None, 16)                336

 dense_17 (Dense)            (None, 8)                 136

=================================================================
Total params: 472 (1.84 KB)
Trainable params: 472 (1.84 KB)
Non-trainable params: 0 (0.00 Byte)
```



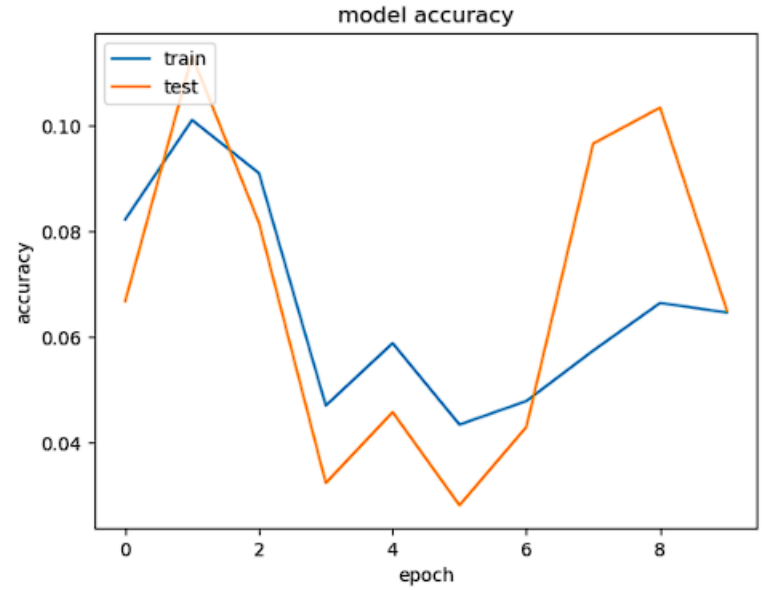


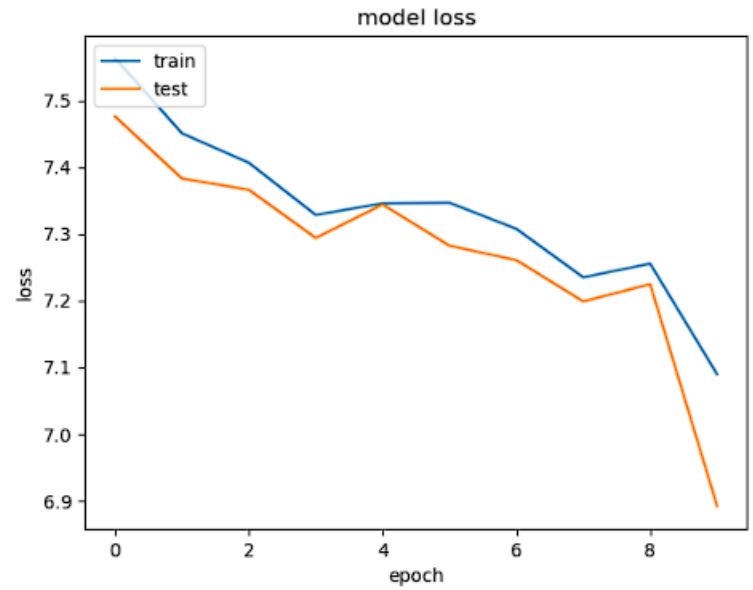
Dựa vào các biểu đồ output, có thể thấy rằng model với 2 layer có độ chính xác cao hơn so với model với 1 layer. Tuy nhiên, độ chính xác của model với 2 layer chỉ cao hơn một chút so với model với 1 layer. Ngoài ra, model với 2 layer cũng có độ mất mát (loss) thấp hơn so với model với 1 layer.

* Áp dụng 3 layer cho 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 Dense, Flatten
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()
layer_1 = Dense(16, input_dim=maxlen)
model.add(layer_1)
layer_2 = Dense(8, activation='relu')
model.add(layer_2)
layer_3 = Dense(4, activation='relu')
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)

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_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_18 (Dense)            (None, 16)                336

 dense_19 (Dense)            (None, 8)                 136

 dense_20 (Dense)            (None, 4)                 36

=================================================================
Total params: 508 (1.98 KB)
Trainable params: 508 (1.98 KB)
Non-trainable params: 0 (0.00 Byte)
```





Dựa vào các biểu đồ output, có thể thấy rằng model với 3 layer có độ chính xác cao hơn so với model với 1 layer. Tuy nhiên, độ chính xác của model với 3 layer chỉ cao hơn một chút so với model với 1 layer. Ngoài ra, model với 3 layer cũng có độ mất mát (loss) thấp hơn so với model với 1 layer.

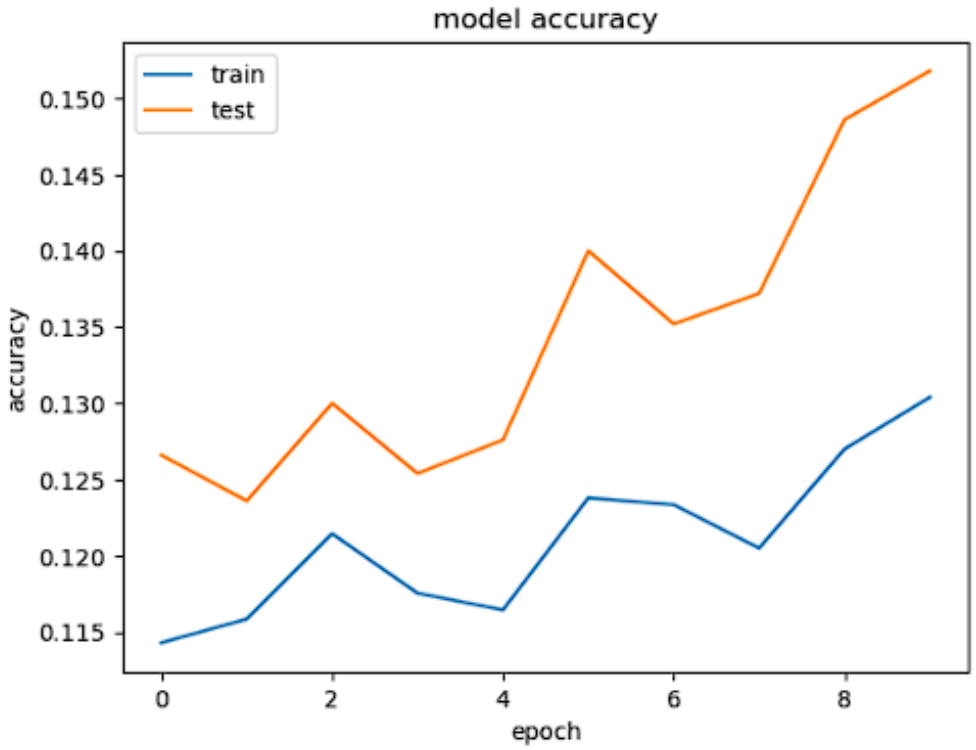
7.6

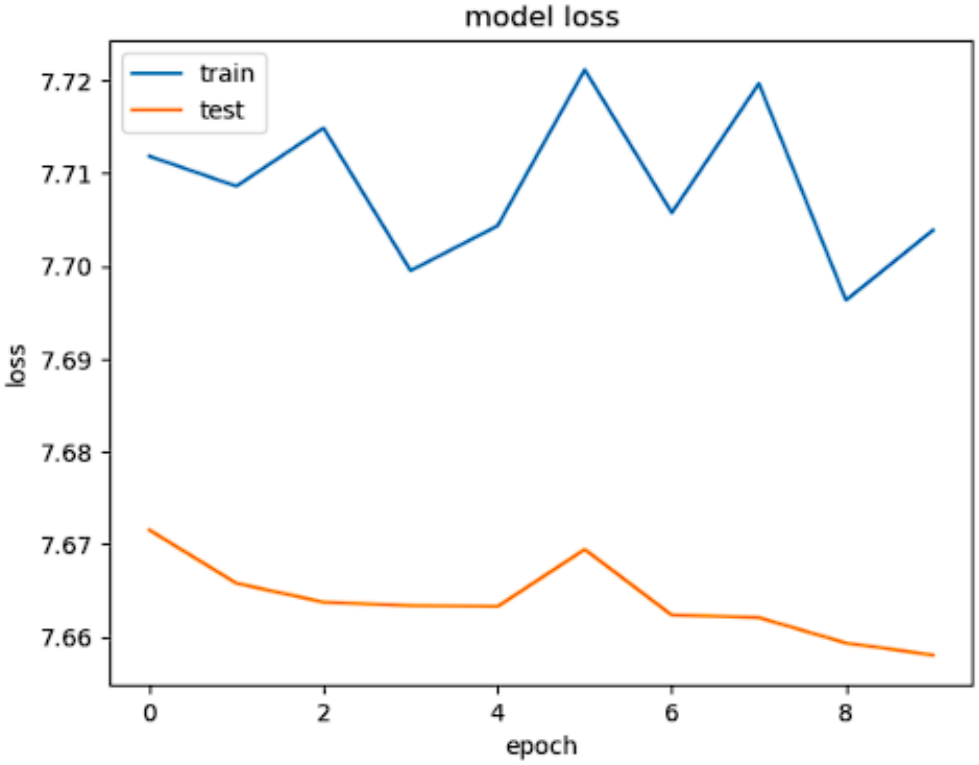
Dropout là một kỹ thuật chính regularization trong deep learning. Nó là một phần của mô hình thường được sử dụng để ngăn chặn việc overfitting. Kỹ thuật này hoạt động bằng cách ngẫu nhiên "bỏ" (tắt) một số lượng các neuron trong quá trình huấn luyện. Nó có thể được áp dụng vào các lớp ẩn trong mô hình để ngăn chặn sự phụ thuộc quá mức vào các neuron cụ thể và đảm bảo tính tổng quát của mô hình.

Dạng 1: Áp dụng dropout với model 1 layer

```
from keras.datasets import imdb
from keras import preprocessing
from keras.layers import Embedding
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout
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()
layer_1 = Dense(16, input_dim=maxlen)
model.add(layer_1)
model.add(Dropout(0.5))
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_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_23 (Dense)            (None, 16)                336

 dropout_1 (Dropout)         (None, 16)                0

=================================================================
Total params: 336 (1.31 KB)
Trainable params: 336 (1.31 KB)
Non-trainable params: 0 (0.00 Byte)
```

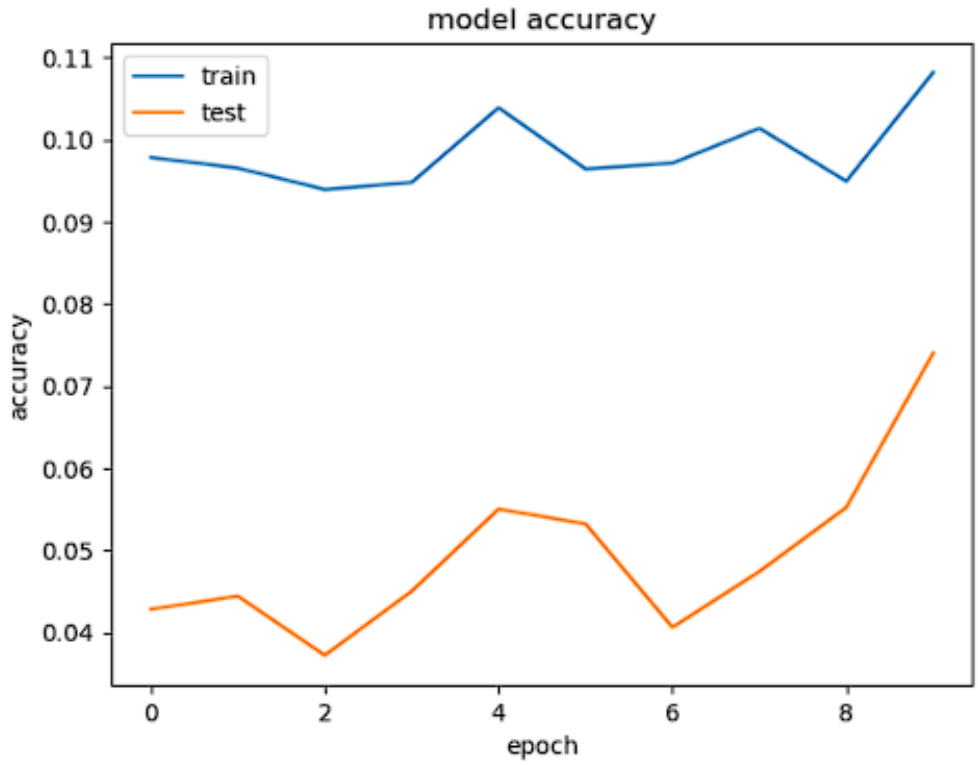

model accuracy
train
test
accuracy
0.150
0.145
0.140
0.135
0.130
0.125
0.120
0.115
0
2
4
6
8
epoch

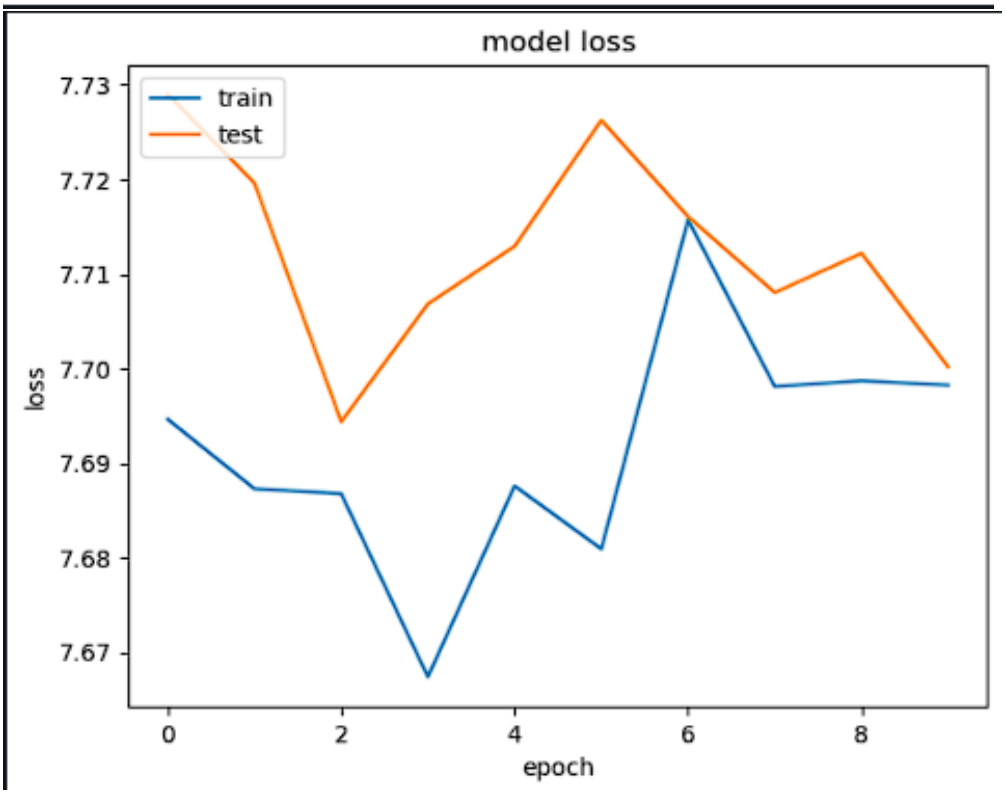

model loss
train
test
loss
7.72
7.71
7.70
7.69
7.68
7.67
7.66
0
2
4
6
8
epoch

Dạng 2: Áp dụng dropout với model 2 layer

```
from keras.datasets import imdb
from keras import preprocessing
from keras.layers import Embedding
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout
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()
layer_1 = Dense(16, input_dim=maxlen)
model.add(layer_1)
layer_2 = Dense(8, activation='relu')
model.add(layer_2)
model.add(Dropout(0.5))
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_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_24 (Dense)            (None, 16)                336

 dense_25 (Dense)            (None, 8)                 136

 dropout_2 (Dropout)         (None, 8)                 0

=================================================================
Total params: 472 (1.84 KB)
Trainable params: 472 (1.84 KB)
Non-trainable params: 0 (0.00 Byte)
```

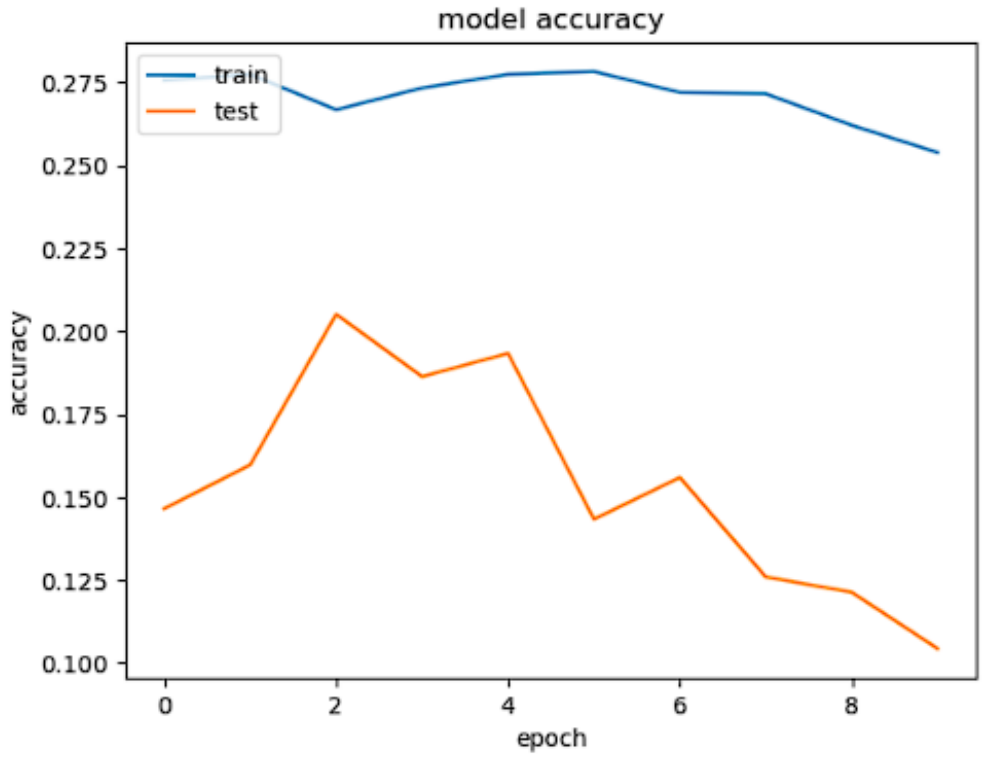

model accuracy
train
test
accuracy
epoch
0.11
0.10
0.09
0.08
0.07
0.06
0.05
0.04
0
2
4
6
8

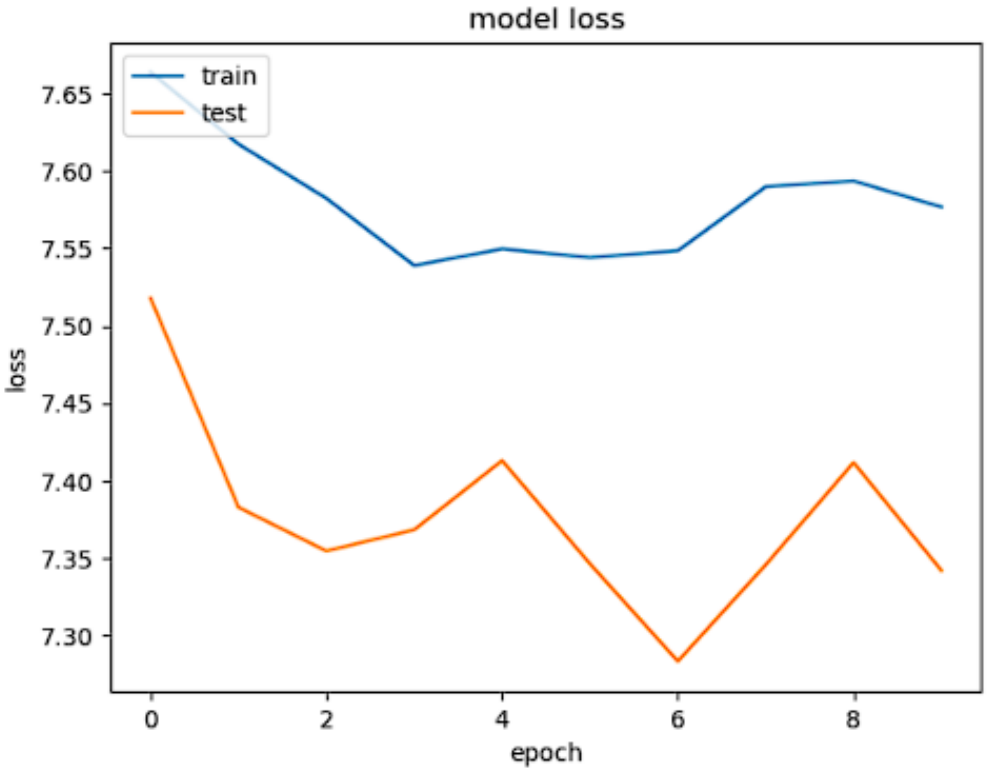

model loss
train
test
loss
epoch
7.73
7.72
7.71
7.70
7.69
7.68
7.67
0
2
4
6
8

Dạng 3: Áp dụng dropout với model 3 layer

```
from keras.datasets import imdb
from keras import preprocessing
from keras.layers import Embedding
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout
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()
layer_1 = Dense(16, input_dim=maxlen)
model.add(layer_1)
layer_2 = Dense(8, activation='relu')
model.add(layer_2)
layer_3 = Dense(4, activation='relu')
model.add(layer_3)
model.add(Dropout(0.5))
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_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_26 (Dense)            (None, 16)                336

 dense_27 (Dense)            (None, 8)                 136

 dense_28 (Dense)            (None, 4)                 36

 dropout_3 (Dropout)         (None, 4)                 0

=================================================================
Total params: 508 (1.98 KB)
Trainable params: 508 (1.98 KB)
Non-trainable params: 0 (0.00 Byte)
```





Dropout đã giúp cải thiện độ chính xác và độ mất mát của tất cả các model. Đặc biệt, model với 3 layer + dropout có độ chính xác và độ mất mát tốt nhất.