# Phần 2: Deep learning

# CHƯƠNG 5: TENSORFLOW

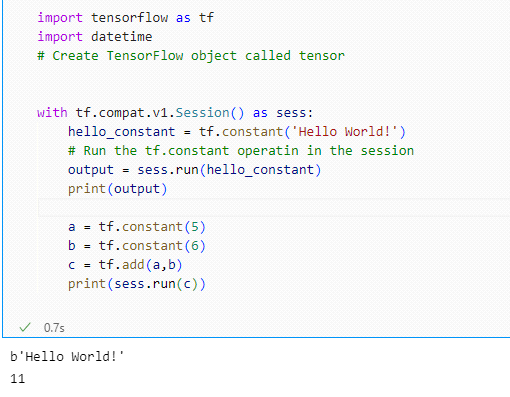
## 1 Tensorflow

* **Tensorflow:**

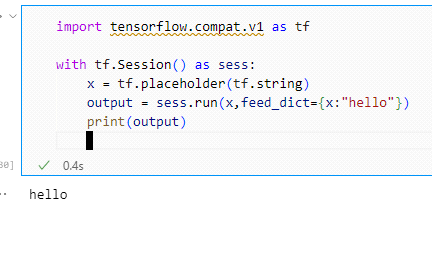
Trong tensorflow, dữ liệu không được lưu dưới dạng int, float hay string. Các giá trị này được đóng gói trong một đối tượng gọi là tensor.

tf.constant() được gọi là 1 tensor ko đổi vì giá trị của tensor không bao giờ thay đổi.

Tensorflow phân biệt rạch ròi việc định nghĩa và tính toán trong quá trình thực thi. Bao gồm:  
 + Xây dựng, định nghĩa đồ thị(Graph).   
 + Sử dụng một Session để thực thi các tính toán trong đồ thị  
sess.run() đánh giá tensor và trả về kết quả.



tf.placeholder() trả về 1 tensor nhận giá trị được truyền vào từ tf.session.run() thông qua feed\_dict, tf.placeholder() cho phép đặt kiểu dữ liệu đầu vào ngay trước khi run()



* **Phép tính trong tensorflow:**

Phép cộng: x =tf.add(5,7)

Phép trừ: y = tf.subtract(10,4)

Phép nhân: z = tf.multiply(2,5)

Phép chia: t = tf.divide(10,2)

Tính từ trong ra ngoài:

Vd: tf.subtract(tf.cast(tf.constant(2.0), tf.int32), tf.constant(1)) # 1

* **Biến trong tensor:**

x= tf.variable(5)

lớp tf.variable tạo ra 1 tensor giá trị ban đầu và có thể thay đổi giá trị.

(x1.initializer) điều này nghĩa là khởi tạo x với giá trị bằng 5

init=tf.global\_variables\_initializer()#khởi tạo trạng thái của tất cả cácbiến

with tf.Session() as sess:

sess.run(init)

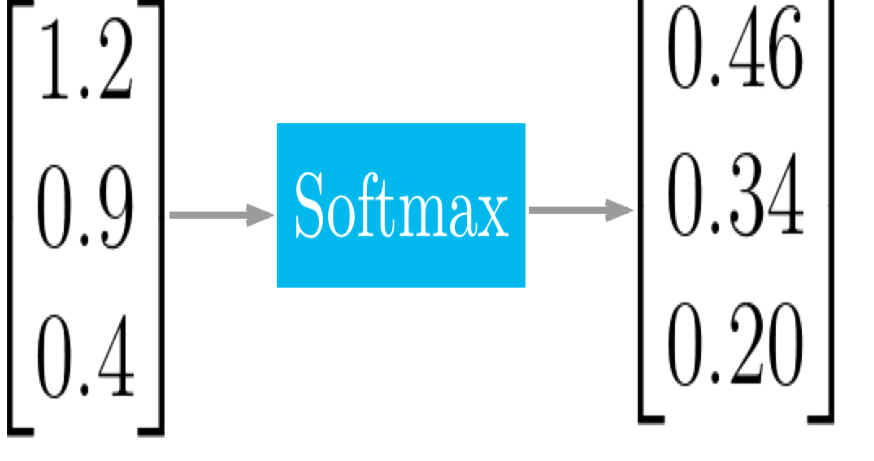
* tf.truncated\_normal()

hàm trả về 1 tensor có giá trị ngẫu nhiên, độ lớn không quá 2

* tf.zeros()

trả về 1 tensor với tất cả các số 0.

* Tensorflow softmax:

Chuẩn hóa đầu ra sao cho tất cả có tổng là 1:  


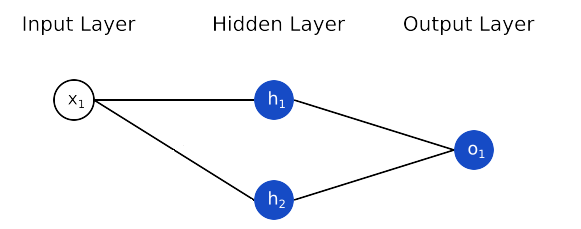
**2, trình bày hiểu biết về học sâu:**

TensorFlow cung cấp một nền tảng học máy toàn diện cung cấp cả khả năng cấp cao và cấp thấp để xây dựng và triển khai các mô hình học máy. Tuy nhiên, nó có một đường cong học tập dốc. Nó được sử dụng tốt nhất khi bạn có nhu cầu:

* Nghiên cứu học tập sâu
* Mạng nơ-ron phức tạp
* Làm việc với bộ dữ liệu lớn
* Các mô hình hiệu suất cao

Cách xác định mạng thần kinh bằng API tuần tự của Keras

API tuần tự là một khuôn khổ để tạo các mô hình dựa trên các phiên bản của lớp **sequential ()**. Mô hình có một biến đầu vào, một lớp ẩn với hai nơ-ron và một lớp đầu ra với một đầu ra nhị phân. Các lớp bổ sung có thể được tạo và thêm vào mô hình.



# Xác định mô hình:

from keras.models import Sequential

from keras.layers import Dense

model = Sequential()

model.add (Dense (2, input\_dim=1,activation=’relu'))

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

Hàm **Summary ()**được sử dụng để tạo và in tóm tắt trong bảng điều khiển Python:

from keras.models import Sequential

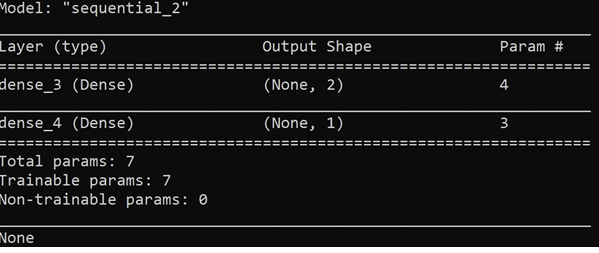
from keras.layers import Dense

model = Sequential()

model.add(Dense(2, input\_dim=1, activation='relu'))

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

print(model.summary())



## 2. Các bước xây dựng mô hình Deep Learning với Tensorflow và Keras:

Bước 1: Chuẩn bị dữ liệu

Bước 2: Xây dựng mô hình

Bước 3: Huấn luyện mô hình

Bước 4: Đánh giá mô hình

Bước 5: Dự đoán

import numpy as np

from keras.models import Sequential

from keras.layers import Dense, Activation

### # Bước 1: Chuẩn bị dữ liệu

# np.random.seed(2018) là hàm khởi tạo giá trị ngẫu nhiên cho hàm random tại seed(2018)

# với mỗi seed(value) thì các giá trị ngẫu nhiên sẽ giống nhau trong 1 phiên làm việc

# seed(0) thì các giá trị ngẫu nhiên sẽ khác seed(1), seed(2)…

np.random.seed(2018)

x\_train = np.random.random((6000,10))

y\_train = np.random.randint(2, size=(6000, 1))

# Generate dummy validation dataset

x\_val = np.random.random((2000,10))

y\_val = np.random.randint(2, size=(2000, 1))

# Generate dummy test dataset

x\_test = np.random.random((2000,10))

y\_test = np.random.randint(2, size=(2000, 1))

### # Bước 2: Xây dựng mô hình

model = Sequential()

model.add(Dense(64, input\_dim=10,activation = "relu"))

#Layer 1

model.add(Dense(32,activation = "relu")) #Layer 2

model.add(Dense(16,activation = "relu")) #Layer 3

model.add(Dense(8,activation = "relu")) #Layer 4

model.add(Dense(4,activation = "relu")) #Layer 5

model.add(Dense(1,activation = "sigmoid")) #OutputLayer

#Configure the model

model.compile(optimizer='Adam',loss='binary\_crossentropy',metrics=['accuracy'])

### # Bước 3: Huấn luyện mô hình

model.fit(x\_train, y\_train, batch\_size=64, epochs=3, validation\_data=(x\_val,y\_val))

#evaluate(x=None, y=None, batch\_size=None, verbose=1,sample\_weight=None, steps=None)

### # Bước 4: Đánh giá mô hình

print(model.evaluate(x\_test,y\_test))

print(model.metrics\_names)

### # Bước 5: Dự đoán

#print 10 predictions

pred = model.predict(x\_test)

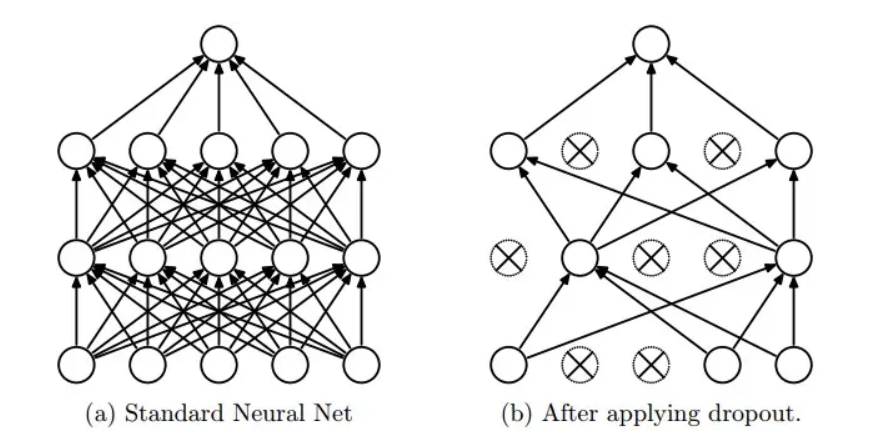
pred[:10]

# CHƯƠNG 6: HỌC SÂU VÀ KERAS

## 1. Dropout là gì?

Dropout là một kỹ thuật giúp giảm overfitting trong quá trình huấn luyện mô hình.

overfitting là hiện tượng mô hình học tập quá mức trên tập huấn luyện, nhưng lại không thể khái quát được trên tập kiểm tra.



### Model ban đầu:

from matplotlib import pyplot as plt

import numpy as np

from keras.models import Sequential

from keras.layers import Dense, Activation

# Bước 1: Chuẩn bị dữ liệu

# np.random.seed(2018) là hàm khởi tạo giá trị ngẫu nhiên cho hàm random tại seed(2018)

# với mỗi seed(value) thì các giá trị ngẫu nhiên sẽ giống nhau trong 1 phiên làm việc

# seed(0) thì các giá trị ngẫu nhiên sẽ khác seed(1), seed(2)…

np.random.seed(2018)

x\_train = np.random.random((6000,10))

y\_train = np.random.randint(2, size=(6000, 1))

# Generate dummy validation dataset

x\_val = np.random.random((2000,10))

y\_val = np.random.randint(2, size=(2000, 1))

# Generate dummy test dataset

x\_test = np.random.random((2000,10))

y\_test = np.random.randint(2, size=(2000, 1))

#Bước 2: Xây dựng mô hình

model = Sequential()

model.add(Dense(64, input\_dim=10,activation = "relu"))

#Layer 1

model.add(Dense(32,activation = "relu")) #Layer 2

model.add(Dense(16,activation = "relu")) #Layer 3

model.add(Dense(8,activation = "relu")) #Layer 4

model.add(Dense(4,activation = "relu")) #Layer 5

model.add(Dense(1,activation = "sigmoid")) #OutputLayer

#Configure the model

model.compile(optimizer='Adam',loss='binary\_crossentropy',metrics=['accuracy'])

#Bước 3: Huấn luyện mô hình

history = model.fit(x\_train, y\_train, batch\_size=64, epochs=30, validation\_data=(x\_val,y\_val))

#evaluate(x=None, y=None, batch\_size=None, verbose=1,sample\_weight=None, steps=None)

# Bước 4: Đánh giá mô hình

print(model.evaluate(x\_test,y\_test))

print(model.metrics\_names)

# Bước 5: Dự đoán

#print 10 predictions

pred = model.predict(x\_test)

pred[:10]

print(history.history.keys())

# summarize history for accuracy

#7. Giải thích Biểu đồ

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

#8. Giải thích biểu đồ

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

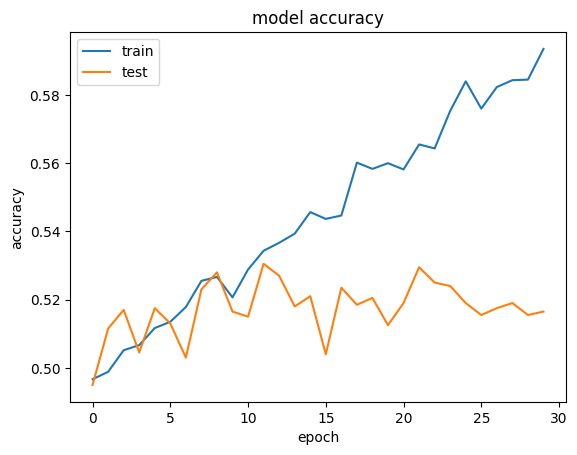
plt.ylabel('loss')

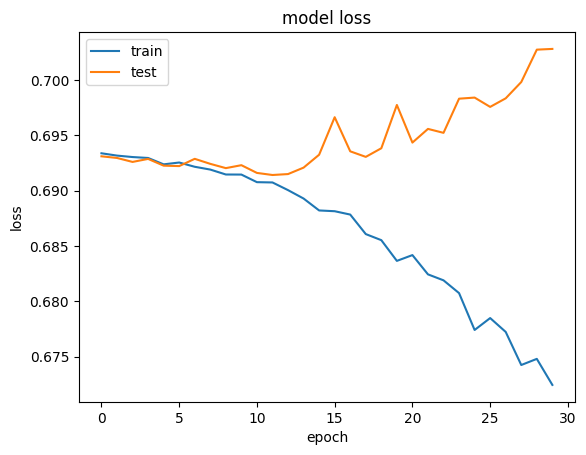
plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()







* Mô hình đã bị overfitting

### Model thêm dropout

import numpy as np

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout

# Bước 1: Chuẩn bị dữ liệu

# np.random.seed(2018) là hàm khởi tạo giá trị ngẫu nhiên cho hàm random tại seed(2018)

# với mỗi seed(value) thì các giá trị ngẫu nhiên sẽ giống nhau trong 1 phiên làm việc

# seed(0) thì các giá trị ngẫu nhiên sẽ khác seed(1), seed(2)…

np.random.seed(2018)

x\_train = np.random.random((6000,10))

y\_train = np.random.randint(2, size=(6000, 1))

# Generate dummy validation dataset

x\_val = np.random.random((2000,10))

y\_val = np.random.randint(2, size=(2000, 1))

# Generate dummy test dataset

x\_test = np.random.random((2000,10))

y\_test = np.random.randint(2, size=(2000, 1))

#Bước 2: Xây dựng mô hình

model = Sequential()

model.add(Dense(64, input\_dim=10,activation = "relu"))

model.add(Dropout(0.2))

#Layer 1

model.add(Dense(32,activation = "relu")) #Layer 2

model.add(Dropout(0.2))

model.add(Dense(16,activation = "relu")) #Layer 3

model.add(Dropout(0.2))

model.add(Dense(8,activation = "relu")) #Layer 4

model.add(Dropout(0.2))

model.add(Dense(4,activation = "relu")) #Layer 5

model.add(Dropout(0.2))

model.add(Dense(1,activation = "sigmoid")) #OutputLayer

#Configure the model

model.compile(optimizer='Adam',loss='binary\_crossentropy',metrics=['accuracy'])

#Bước 3: Huấn luyện mô hình

history = model.fit(x\_train, y\_train, batch\_size=64, epochs=30, validation\_data=(x\_val,y\_val))

print(model.evaluate(x\_test,y\_test))

print(model.metrics\_names)

print(history.history.keys())

# summarize history for accuracy

#7. Giải thích Biểu đồ

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

#8. Giải thích biểu đồ

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

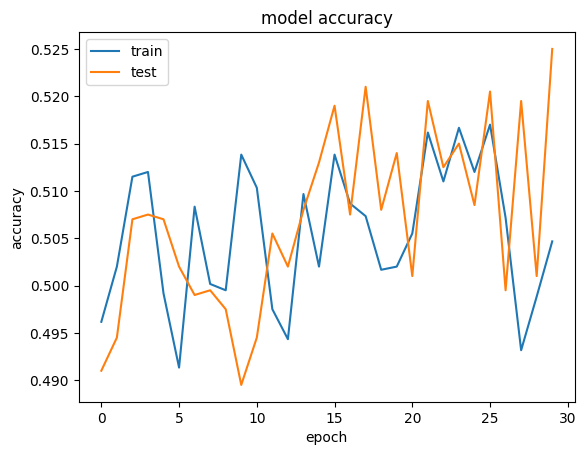
plt.ylabel('loss')

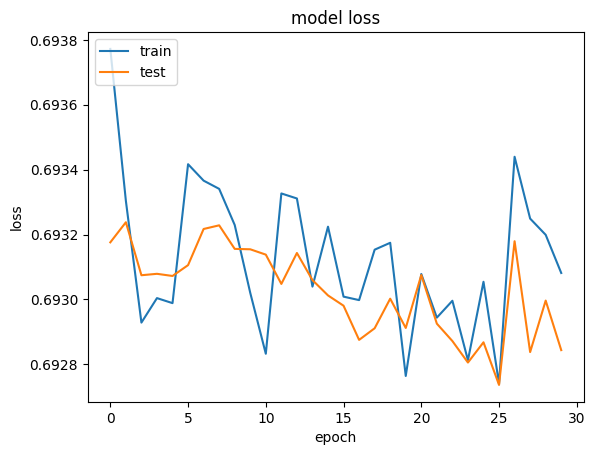
plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()







* Giảm overfitting sau khi them dropout 0.2,
* Accuracy giảm
* Overfit từ epochs 25 so với mô hình ban đầu từ epochs 10

### Model sau khi giảm neural

import numpy as np

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout

# Bước 1: Chuẩn bị dữ liệu

# np.random.seed(2018) là hàm khởi tạo giá trị ngẫu nhiên cho hàm random tại seed(2018)

# với mỗi seed(value) thì các giá trị ngẫu nhiên sẽ giống nhau trong 1 phiên làm việc

# seed(0) thì các giá trị ngẫu nhiên sẽ khác seed(1), seed(2)…

np.random.seed(2018)

x\_train = np.random.random((6000,10))

y\_train = np.random.randint(2, size=(6000, 1))

# Generate dummy validation dataset

x\_val = np.random.random((2000,10))

y\_val = np.random.randint(2, size=(2000, 1))

# Generate dummy test dataset

x\_test = np.random.random((2000,10))

y\_test = np.random.randint(2, size=(2000, 1))

#Bước 2: Xây dựng mô hình

model = Sequential()

model.add(Dense(32, input\_dim=10,activation = "relu"))

#Layer 1

model.add(Dense(16,activation = "relu")) #Layer 2

model.add(Dense(8,activation = "relu")) #Layer 3

model.add(Dense(4,activation = "relu")) #Layer 4

model.add(Dense(2,activation = "relu")) #Layer 5

model.add(Dense(1,activation = "sigmoid")) #OutputLayer

#Configure the model

model.compile(optimizer='Adam',loss='binary\_crossentropy',metrics=['accuracy'])

#Bước 3: Huấn luyện mô hình

history = model.fit(x\_train, y\_train, batch\_size=64, epochs=30, validation\_data=(x\_val,y\_val))

print(model.evaluate(x\_test,y\_test))

print(model.metrics\_names)

print(history.history.keys())

# summarize history for accuracy

#7. Giải thích Biểu đồ

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

#8. Giải thích biểu đồ

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

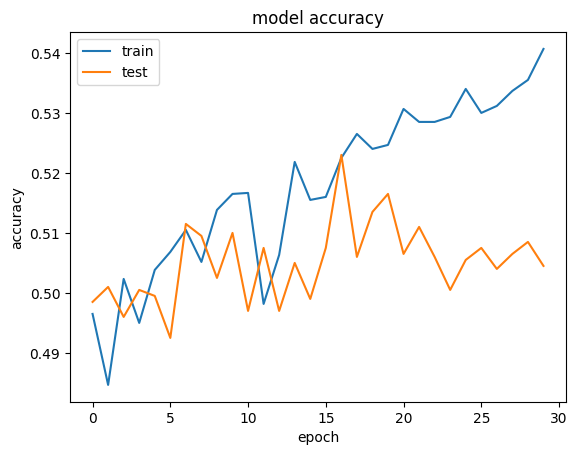
plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()







* Bị overfit từ khoảng epochs 12, 13
* Tăng accuracy 1% ~ 2%
* Độ chính xác của model thay đổi

## 2. Trình bày các hiểu biết của bạn về các khái niệm với keras code cụ thể:

import numpy as np

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout

# Bước 1: Chuẩn bị dữ liệu

# np.random.seed(2018) là hàm khởi tạo giá trị ngẫu nhiên cho hàm random tại seed(2018)

# với mỗi seed(value) thì các giá trị ngẫu nhiên sẽ giống nhau trong 1 phiên làm việc

# seed(0) thì các giá trị ngẫu nhiên sẽ khác seed(1), seed(2)…

np.random.seed(2018)

x\_train = np.random.random((6000,10))

y\_train = np.random.randint(2, size=(6000, 1))

# Generate dummy validation dataset

x\_val = np.random.random((2000,10))

y\_val = np.random.randint(2, size=(2000, 1))

# Generate dummy test dataset

x\_test = np.random.random((2000,10))

y\_test = np.random.randint(2, size=(2000, 1))

#Bước 2: Xây dựng mô hình

model = Sequential()

model.add(Dense(32, input\_dim=10,activation = "relu"))

#Layer 1

model.add(Dense(16,activation = "relu")) #Layer 2

model.add(Dense(8,activation = "relu")) #Layer 3

model.add(Dense(4,activation = "relu")) #Layer 4

model.add(Dense(2,activation = "relu")) #Layer 5

model.add(Dense(1,activation = "sigmoid")) #OutputLayer

#Configure the model

model.compile(optimizer='Adam',loss='binary\_crossentropy',metrics=['accuracy'])

#Bước 3: Huấn luyện mô hình

model.fit(x\_train, y\_train, batch\_size=64, epochs=30, validation\_data=(x\_val,y\_val))

print(model.evaluate(x\_test,y\_test))

print(model.metrics\_names)

Phân loại trong trường hợp đầu ra nhiều hơn hai danh mục, sử dụng chức năng kích hoạt [softmax](https://keras.io/activations/) ,

trường hợp một biến thể thì sử dụng sigmoid .

### 2.1 Epoch

Một epoch là một vòng lặp qua toàn bộ tập dữ liệu huấn luyện. Thay đổi epoch có thể làm tăng độ chính xác của mô hình.

Epoch càng lớn thì càng tốt

### 2.2 Batch\_size:

số lượng mẫu dữ liệu được sử dụng để cập nhật các tham số trong mô hình. Thay đổi batch\_size có thể làm tăng độ chính xác của mô hình.

Cần chia batch\_size vì tập dữ liệu rất lớn

### 2.3 optimizer:

thuật toán hoặc phương pháp sử dụng để thay đổi trọng số hoặc tốc độ học của mạng neroun để giảm tổn thất(loss) và cung cấp kết quả chính xác nhất có thể

Ví dụ: Adam(lr=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

Một số thuật toán tối ưu optimizer

* Gradient Descent
* Stochastic Gradient Descent
* Stochastic Gradient descent with momentum
* Mini-Batch Gradient Descent
* Adagrad
* RMSProp
* AdaDelta
* Adam

gradient descent

cho mạng nơ-ron là một phương pháp tối ưu hóa có tác dụng giảm thiểu sai sót trong quá trình đào tạo

Mục đích của gradient descent là để có được điểm mà lỗi ít nhất.

### 2.4 Loss

là một hàm số để đo lường sai số giữa giá trị dự đoán và giá trị thực tế.

h = [1, 2, 3, 4, 5] # giá trị dự đoán

    w = [1, 2, 3, 4, 5] # giá trị thực tế

    Mục đích tìm w sao cho h gần w để loss nhỏ nhất bằng các phương pháp tối ưu như mean\_squared\_error, mean\_absolute\_error, mean\_absolute\_percentage\_error, mean\_squared\_logarithmic\_error, squared\_hinge, hinge, categorical\_hinge, logcosh, categorical\_crossentropy, sparse\_categorical\_crossentropy, binary\_crossentropy, kullback\_leibler\_divergence, poisson, cosine\_proximity

### 2.5 Cách chọn loss:

* Regression Loss Functions
  + Mean Squared Error Loss
  + Mean Squared Logarithmic Error Loss
  + Mean Absolute Error Loss
* Binary Classification Loss Functions
  + Binary Cross-Entropy
  + Hinge Loss
  + Squared Hinge Loss
* Multi-Class Classification Loss Functions
  + Multi-Class Cross-Entropy Loss
  + Sparse Multiclass Cross-Entropy Loss
  + Kullback Leibler Divergence Loss

### 2.6 metrics:

là một danh sách các đánh giá (metrics) để đánh giá hiệu quả của mạng nơ-ron. Có nhiều loại metrics khác nhau như accuracy, binary\_accuracy, categorical\_accuracy, top\_k\_categorical\_accuracy, sparse\_top\_k\_categorical\_accuracy, mean\_squared\_error, mean\_absolute\_error, mean\_absolute\_percentage\_error, mean\_squared\_logarithmic\_error, squared\_hinge, hinge, categorical\_hinge, logcosh, categorical\_crossentropy, sparse\_categorical\_crossentropy, binary\_crossentropy, kullback\_leibler\_divergence, poisson, cosine\_proximity. Mỗi loại metrics đánh giá hiệu quả của mạng nơ-ron khác nhau.

# CHƯƠNG 7:CASE STUDY

# 1. Case study:

## a. [1đ] Chạy lại Chương 12 ML cơ bản. Trình bày lại phần ML cơ bản với hiểu biết bổ sung, cập nhật:

### Xử lý dữ liệu

# load data

from sklearn.neighbors import KNeighborsClassifier

import matplotlib.pyplot as plt

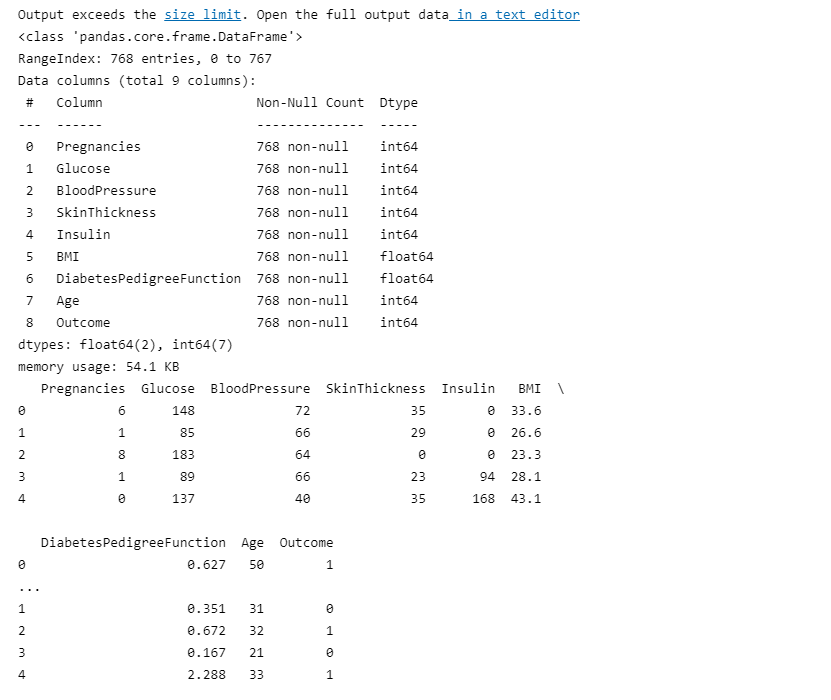
import numpy as np

import pandas as pd

df = pd.read\_csv('./diabetes.csv')

df.info()

print(df.head())

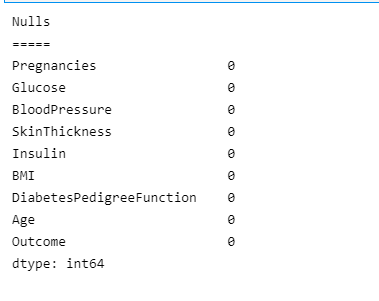


# ---check for null values---

print("Nulls")

print("=====")

print(df.isnull().sum())

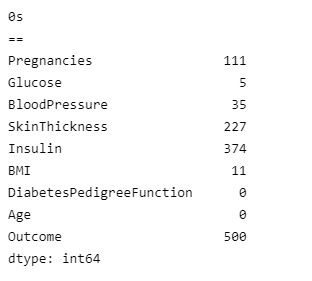


# ---check for 0s---

print("0s")

print("==")

print(df.eq(0).sum())



df[['Glucose','BloodPressure','SkinThickness',

 'Insulin','BMI','DiabetesPedigreeFunction','Age']] = \

 df[['Glucose','BloodPressure','SkinThickness',

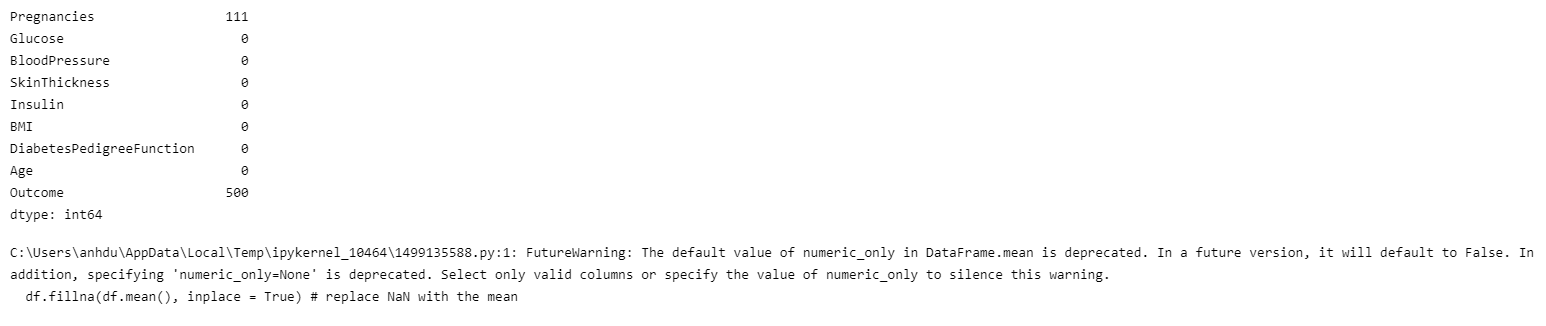
 'Insulin','BMI','DiabetesPedigreeFunction','Age']].replace

(0,np.NaN)



df.fillna(df.mean(), inplace = True) # replace NaN with the mean

print(df.eq(0).sum())



### Ma trận tương quan

corr = df.corr()

print(corr)

fig, ax = plt.subplots(figsize=(10, 10))

cax = ax.matshow(corr, cmap='coolwarm', vmin=-1, vmax=1)

fig.colorbar(cax)

ticks = np.arange(0, len(df.columns), 1)

ax.set\_xticks(ticks)

ax.set\_xticklabels(df.columns)

plt.xticks(rotation=90)

ax.set\_yticklabels(df.columns)

ax.set\_yticks(ticks)

# ---print the correlation factor---

for i in range(df.shape[1]):

    for j in range(9):

        text = ax.text(j, i, round(corr.iloc[i][j], 2), ha="center", va="center", color="w")

plt.show()

import seaborn as sns

sns.heatmap(df.corr(),annot=True)

#---get a reference to the current figure and set its size---

fig = plt.gcf()

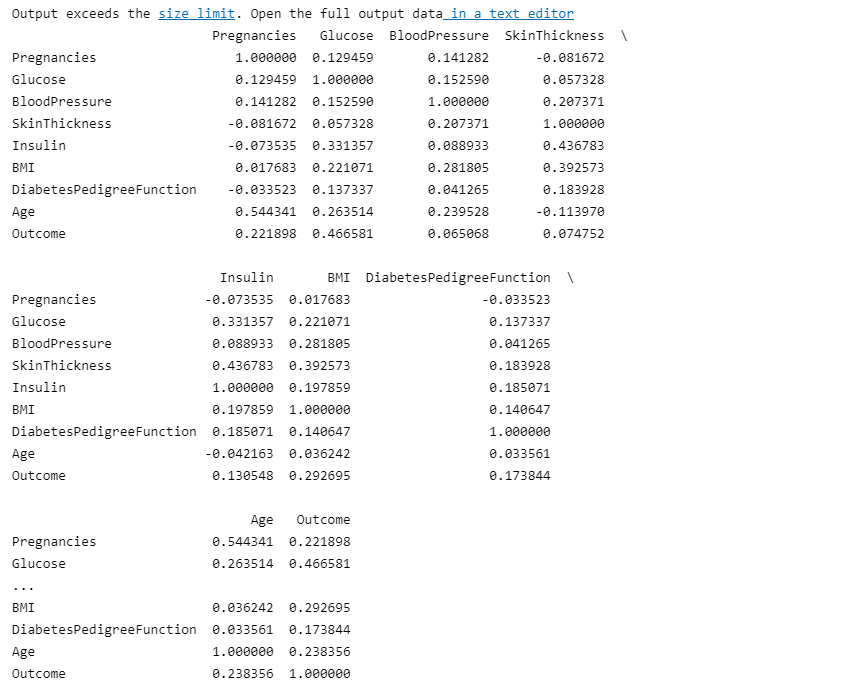
fig.set\_size\_inches(8,8)

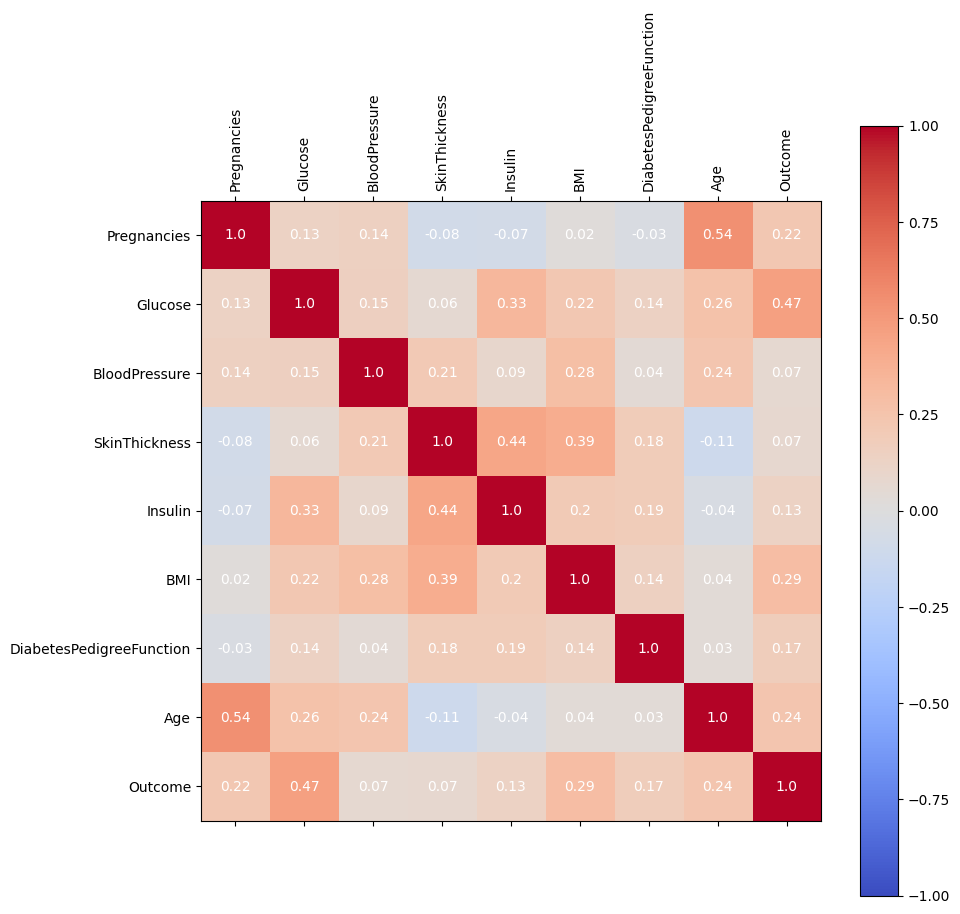
#---get the top four features that has the highest correlation---

print(df.corr().nlargest(4, 'Outcome').index)

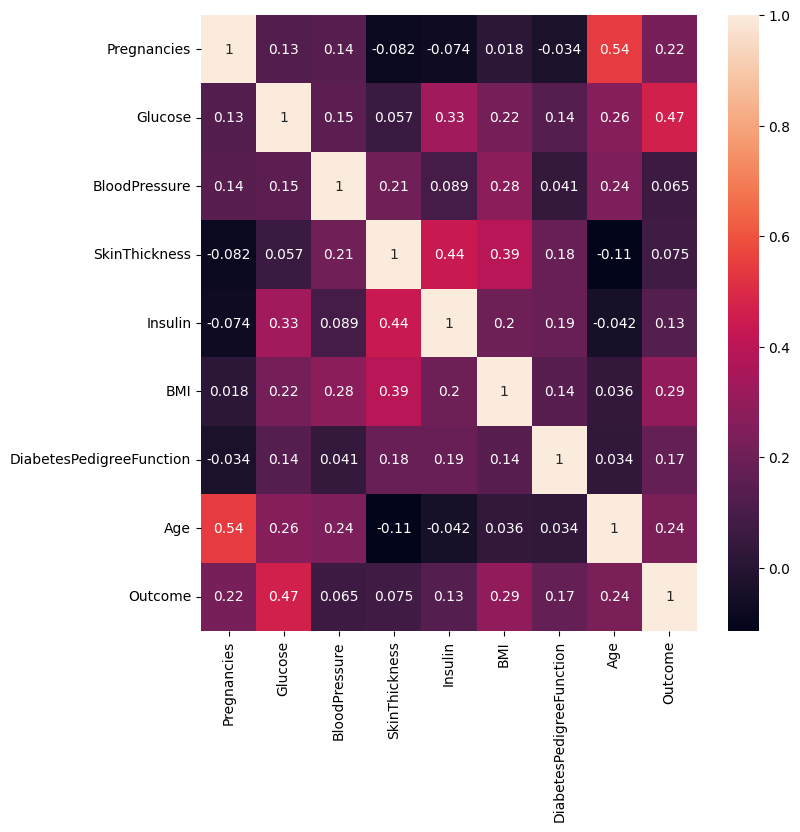
#---print the top 4 correlation values---

print(df.corr().nlargest(4, 'Outcome').values[:,8])





### Bản đồ nhiệt



### Đánh giá mô hình bằng các kỹ thuật cơ bản

# Logistics Regression

from sklearn import linear\_model

from sklearn.model\_selection import cross\_val\_score

#---features---

X = df[['Glucose','BMI','Age']]

#---label---

y = df.iloc[:,8]

log\_regress = linear\_model.LogisticRegression()

log\_regress\_score = cross\_val\_score(log\_regress, X, y, cv=10,

scoring='accuracy').mean()

print(log\_regress\_score)

result = {}

result.update({'Logistic Regression':log\_regress\_score})

# K-Nearest Neighbors

# ---empty list that will hold cv (cross-validates) scores---

cv\_scores = []  # ---number of folds---

folds = 10

# ---creating odd list of K for KNN---

ks = list(range(1, int(len(X) \* ((folds - 1)/folds)), 2))

# ---perform k-fold cross validation---

for k in ks:

    knn = KNeighborsClassifier(n\_neighbors=k)

    score = cross\_val\_score(knn, X, y, cv=folds, scoring='accuracy').mean()

    cv\_scores.append(score)

# ---get the maximum score---

knn\_score = max(cv\_scores)

# ---find the optimal k that gives the highest score---

optimal\_k = ks[cv\_scores.index(knn\_score)]

print(f"The optimal number of neighbors is {optimal\_k}")

print(knn\_score)

result.update({'K-Nearest Neighbors':knn\_score})

# Support Vector Machines

from sklearn import svm

linear\_svm = svm.SVC(kernel='linear')

linear\_svm\_score = cross\_val\_score(linear\_svm, X, y,

 cv=10, scoring='accuracy').mean()

print(linear\_svm\_score)

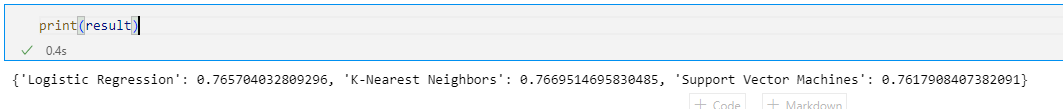
result.update({'Support Vector Machines':linear\_svm\_score})

rbf = svm.SVC(kernel='rbf')

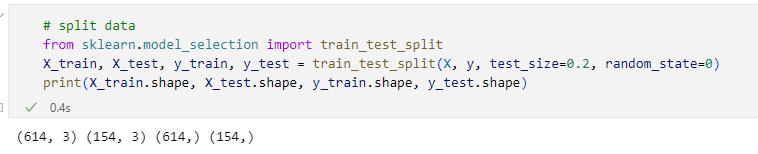
rbf\_score = cross\_val\_score(rbf, X, y, cv=10, scoring='accuracy').mean()

print(rbf\_score)

result.update({'Support Vector Machines':rbf\_score})

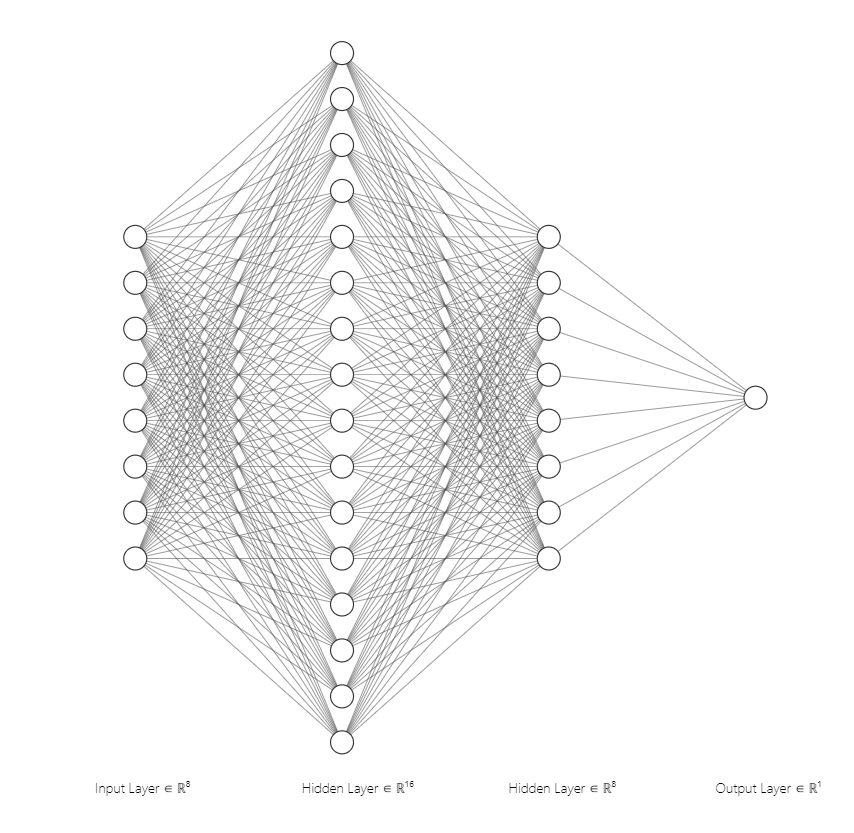


### Train, test:



## b. [5đ] Xây dựng mô hình deep learning để dự đoán dựa trên data như trong Chap 12

• Vẽ cấu trúc mạng neuron mà bạn định sử dụng



### • Dữ liệu: train, test, validation….

# ---load data---

import pandas as pd

df = pd.read\_csv('./diabetes.csv')

# ---features---

X = df.iloc[:,0:8]

# ---label---

y = df.iloc[:,8]

# ---split the data into train and test and validate---

# ---train: 60%, test: 20%, validate: 20%---

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=0)

X\_test, X\_val, y\_test, y\_val = train\_test\_split(X\_test, y\_test, test\_size=0.5, random\_state=0)

### • Các hàm loss, optimizer…mà mô hình sử dụng

# ---build the model---

import keras

from keras.models import Sequential

from keras.layers import Dense

# Tạo model Sequential

model = Sequential()

# Thêm 1 lớp Dense với 16 nơ ron, 8 đầu vào, activation='relu' cho đầu ra không âm 0 đến max(đầu vào)

model.add(Dense(16, input\_dim=8, activation='relu'))

# Thêm 1 lớp Dense với 8 nơ ron, activation='relu' cho đầu ra không âm 0 đến max(đầu vào)

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

# Thêm 1 lớp Dense với 1 nơ ron, activation='sigmoid' cho đầu ra 0 đến 1

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

# ---compile the model---

# loss: hàm mất mát, optimizer: thuật toán tối ưu giảm thiểu loss, metrics: đánh giá

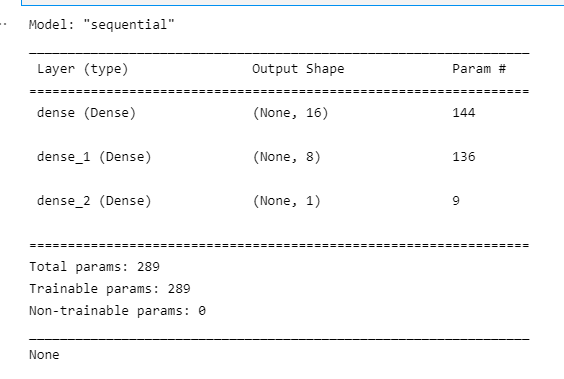
# binary\_crossentropy là hàm mất mát cho bài toán phân loại 2 lớp (0,1)

# adam là thuật toán tối ưu

# accuracy là đánh giá

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

print (model.summary())



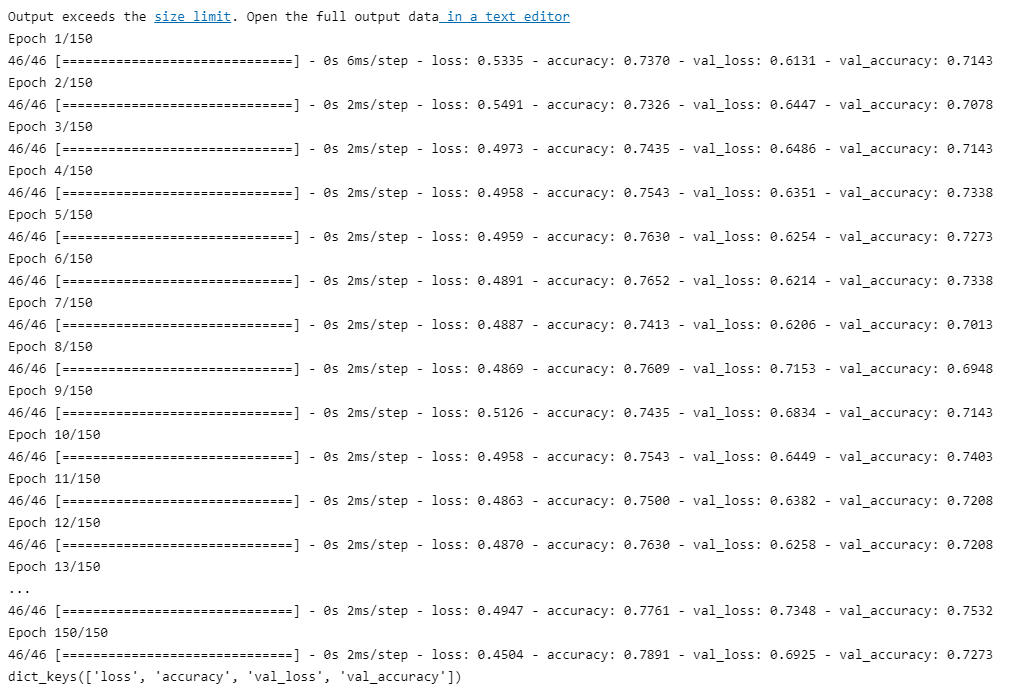
### • Đánh giá kết quả dự đoán…độ chính xác….

#### Mô hình ban đầu

# Đánh giá kết quả dự đoán…độ chính xác….accuracy\_score

# ---train the model---

model.fit(X\_train, y\_train, epochs=150, batch\_size=10, validation\_data=(X\_val, y\_val))



##### Dự đoán

# ---predict the test set---

y\_pred = model.predict(X\_test)

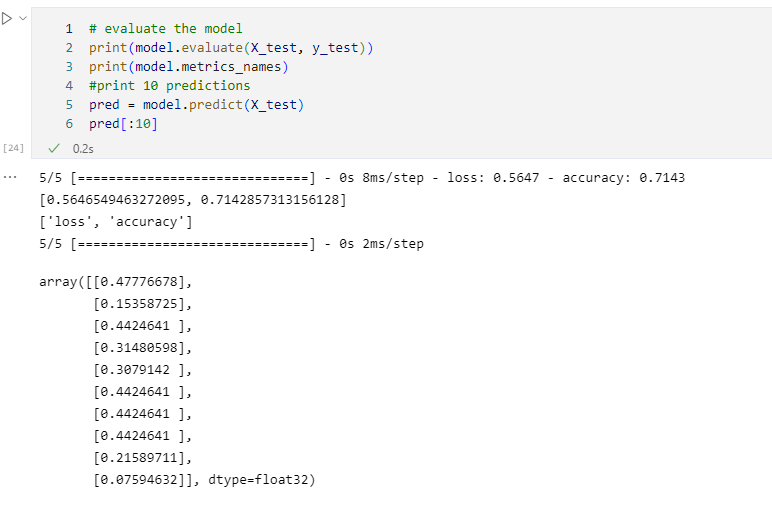
y\_pred = (y\_pred > 0.5)

# evaluate the model

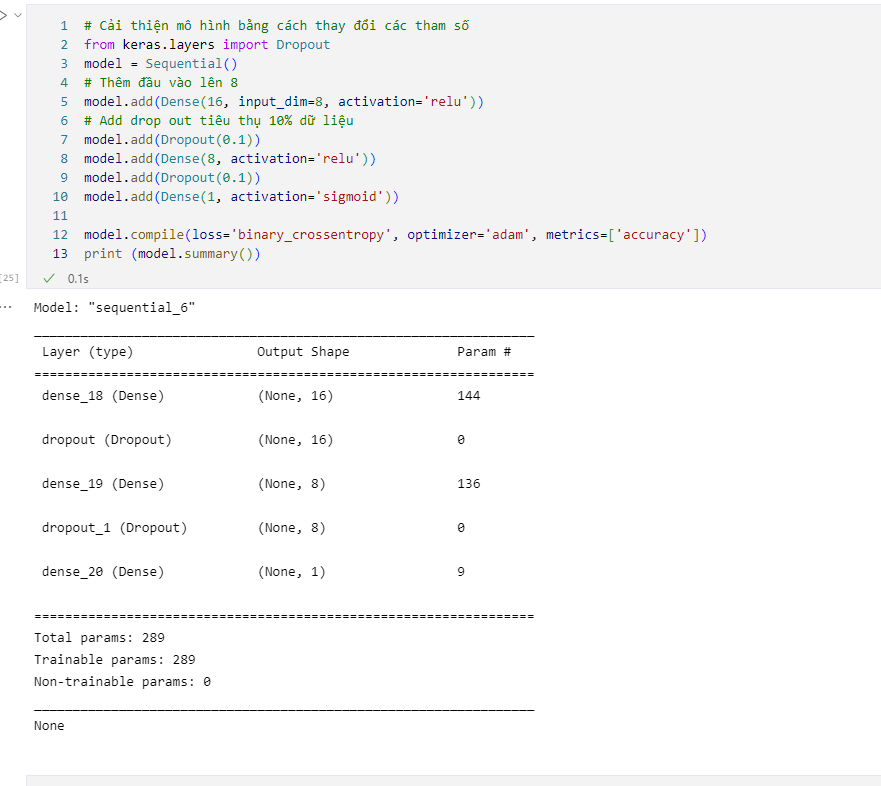
from sklearn.metrics import accuracy\_score

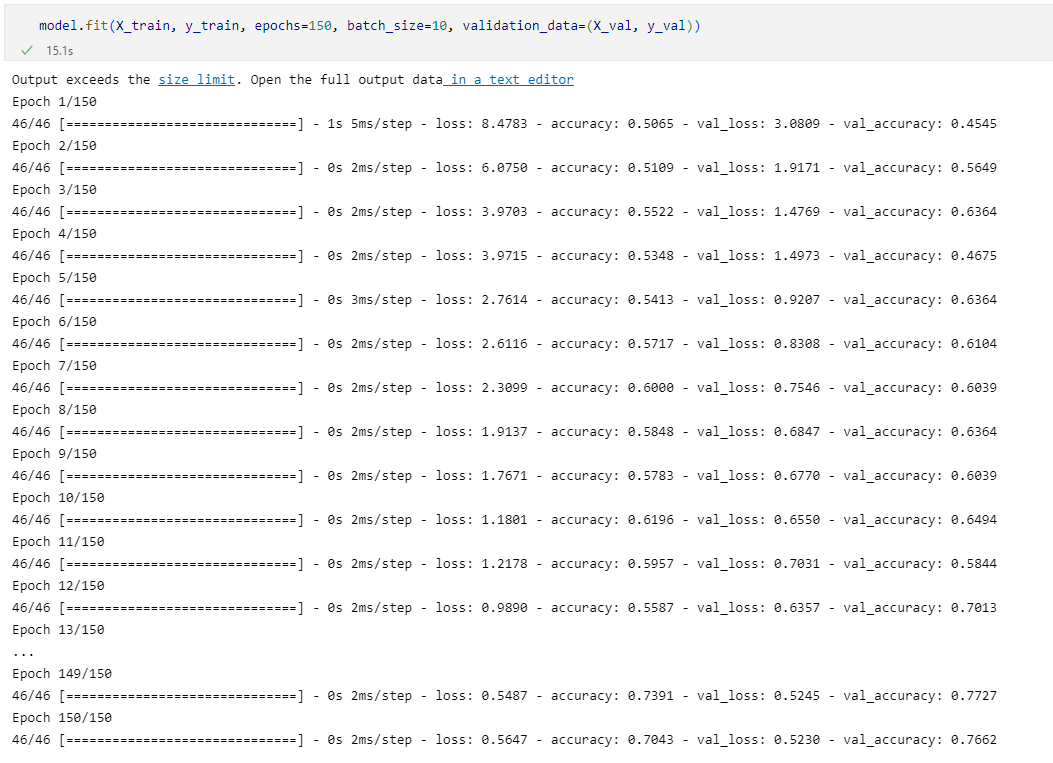
print(accuracy\_score(y\_test, y\_pred))



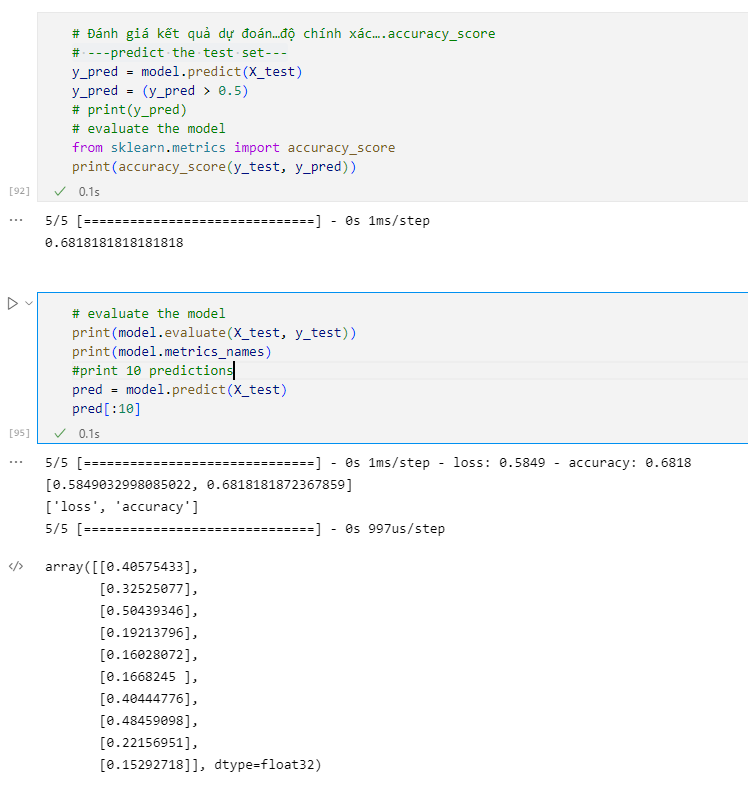


#### Mô hình sau khi dropout:





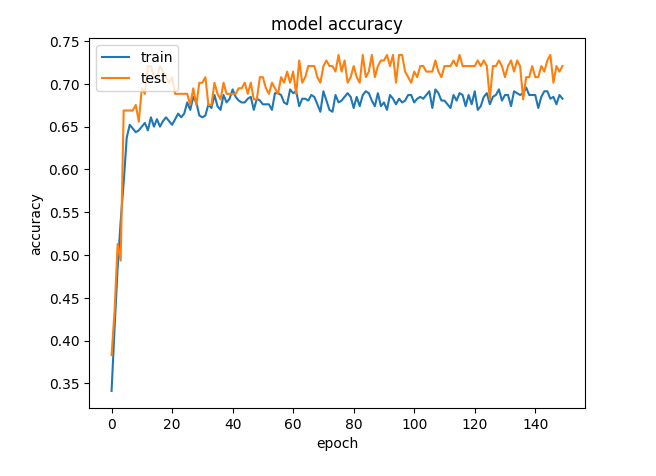
##### Dự đoán

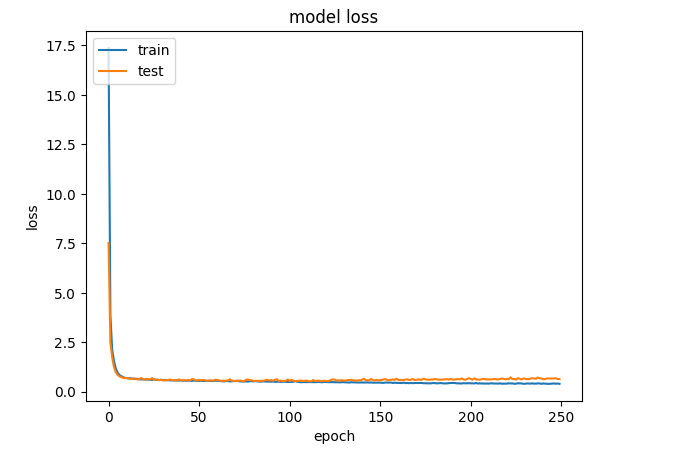


* Loss tăng lên và acc giảm đi

#### • Biểu đồ thể hiện và giải thích

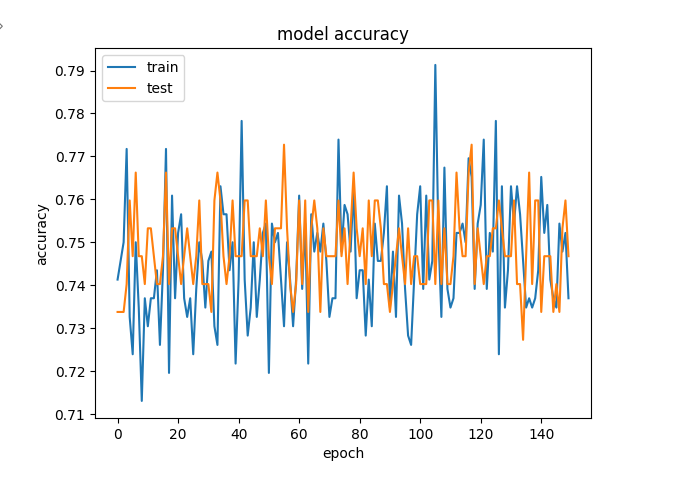
##### Mô hình ban đầu:





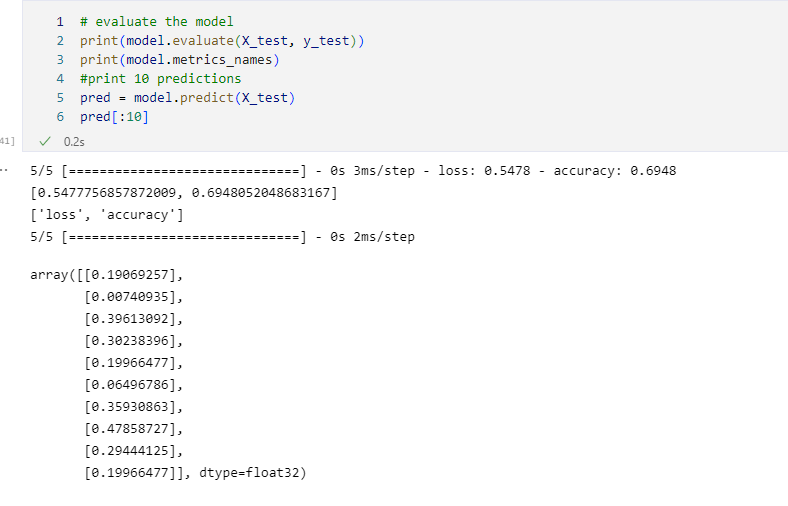
* Mô hình bị overfiting
* Chêch lệch trên hàm loss được tối ưu tốt

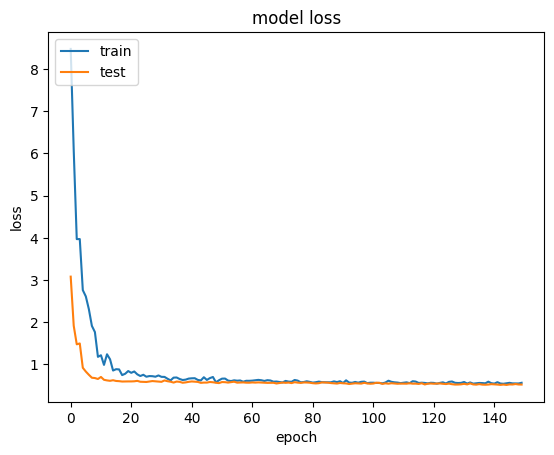
##### Mô hình sau khi dropout:



Đường màu cam thể hiện độ chính xác của validation set theo từng epochs

Đường màu xanh thể hiện độ chính xác của train set theo từng epochs





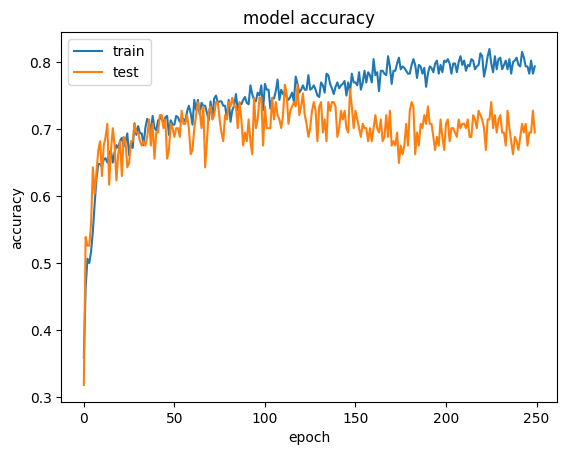
* Giảm được overfit

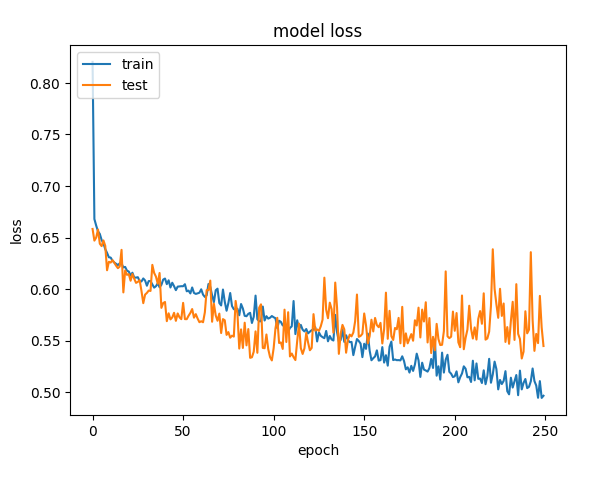
Đường màu cam thể hiện độ mất mát của validation set theo từng epochs

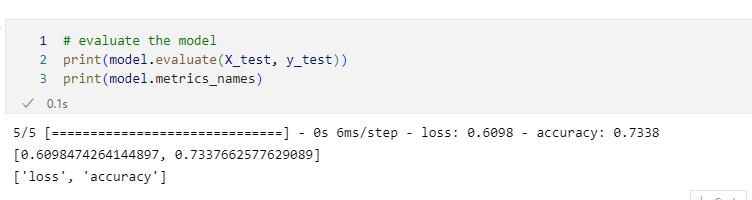
Đường màu xanh thể hiện độ mất mát của train set theo từng epochs

##### Model tăng layer và epochs lên 250:

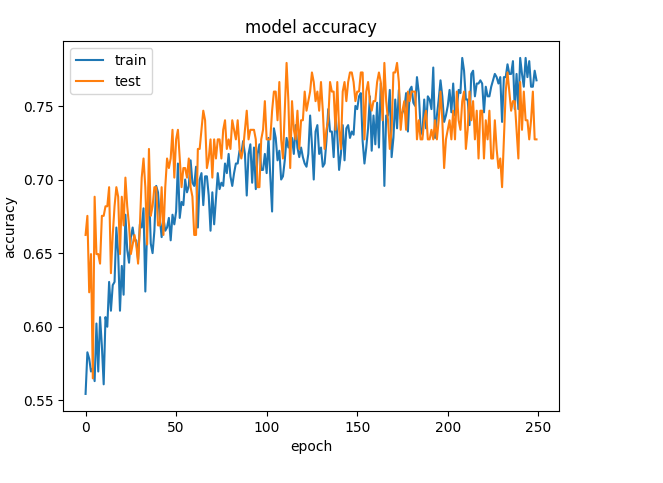
model.fit(X\_train, y\_train, epochs=250, batch\_size=10, validation\_data=(X\_val, y\_val))

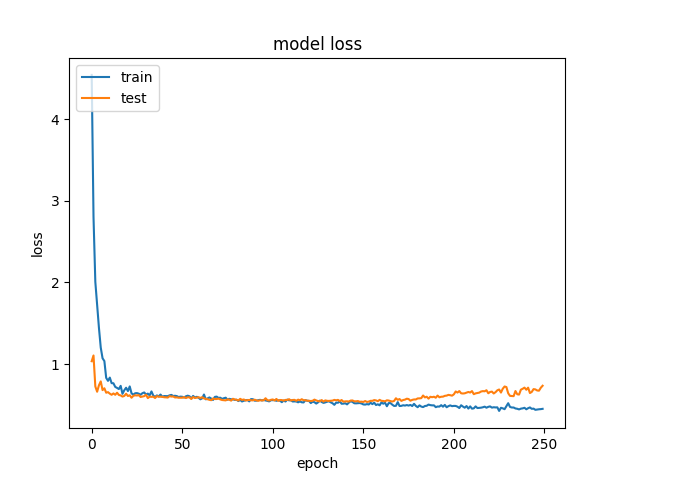




* Nhìn thấy từ epoch 140 trở đi mô hình bị overfiting
* 
* Độ chính xác của model đã tăng lên từ 0.69 ->0.73

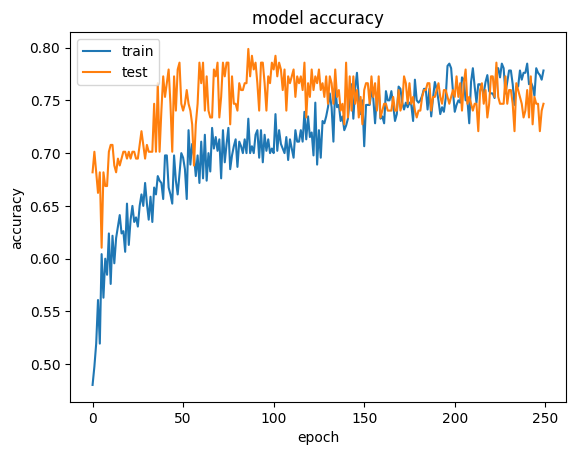
##### Model 250 thêm dropout 0.1 và thêm layer, số neural:

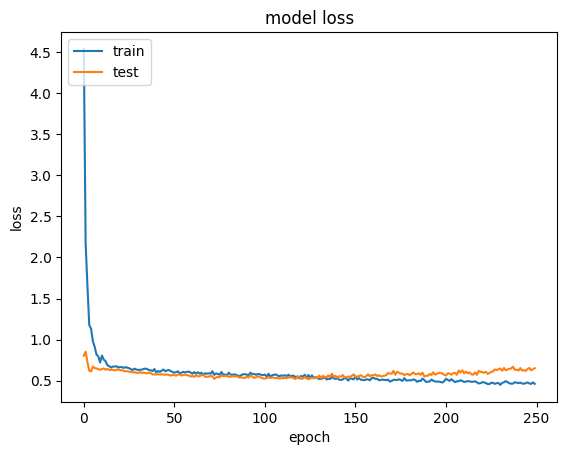




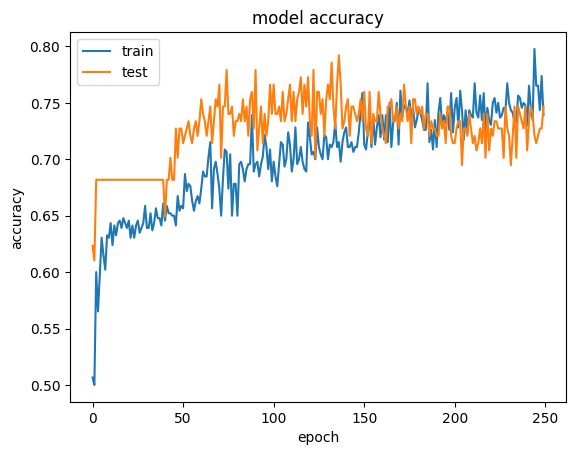
* Loss đã được khắc phục nhưng acc giảm đi

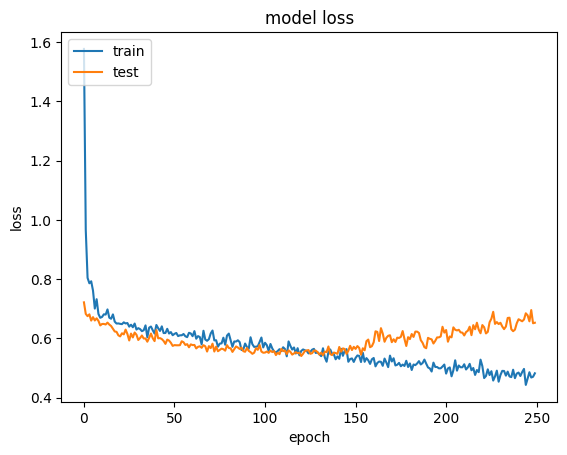
##### Model 250 giảm thêm neural:





##### Model 250 thêm neroun, dropout 0.2:





Model 250 them neroun, dropout 0.2:

#### • So sánh, đánh giá kết quả dự đoán deep learning và các kỹ thuật cớ bản

##### Các kỹ thuật cơ bản:

    Logistic Regression: 0.765704032809296

    K-Nearest Neighbors: 0.7669514695830485

    Support Vector Machines: 0.7617908407382091

##### Deep learning:

Model chưa cải thiện:

    y\_test với y\_pred = 0.7597402597402597

    evaluate the model = [0.5268608927726746, 0.7597402334213257]

                         ['loss', 'accuracy']

Model đã cải thiện với dropout:

    y\_test với y\_pred = 0.6948051948051948

    evaluate the model = [0.5477756857872009, 0.6948052048683167]

                         ['loss', 'accuracy']