Chapter 3: ANN

Ex2: Medical records for Pima Indians

Hãy áp dụng ANN cho bài toán xác định một người có bị tiểu đường hay không?

- https://www.kaggle.com/kumargh/pimaindiansdiabetescsv (https://www.kaggle.com/kumargh/pimaindiansdiabetescsv)

This dataset describes the medical records for Pima Indians and whether or not each patient will have an onset of diabetes within five years.

Fields description follow:

- preg = Number of times pregnant
- plas = Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- pres = Diastolic blood pressure (mm Hg)
- skin = Triceps skin fold thickness (mm)
- test = 2-Hour serum insulin (mu U/ml)
- mass = Body mass index (weight in kg/(height in m)^2)
- pedi = Diabetes pedigree function
- age = Age (years)

class = Class variable (1:tested positive for diabetes, 0: tested negative for diabetes)

In [1]:

```
# from google.colab import drive
# drive.mount("/content/gdrive", force_remount=True)
```

In [2]:

```
# %cd '/content/gdrive/My Drive/LDS8_DeepLearning/Practice/Chapter3/'
```

In [3]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [4]:

```
# Import
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
import numpy
```

In [5]:

```
print(tf.__version__)
print(keras.__version__)
```

```
2.5.0
2.5.0
```

In [6]:

```
# fix random seed for reproducibility
numpy.random.seed(7)
# load pima indians dataset
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
dataset[5]
```

Out[6]:

```
array([  5.   , 116.   ,  74.   ,   0.   ,   0.   ,  25.6  ,   0.201,
        30.   ,   0.   ])
```

In [7]:

```
dataset.size
```

Out[7]:

```
6912
```

In [8]:

```
# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]
```

In [9]:

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

In [10]:

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

In [11]:

```
# Fit the model
history = model.fit(X, Y, epochs=200,
          batch_size=32,
          verbose=1,
          validation_split=0.3)
```

```
y: 0.6108 - val_loss: 0.9599 - val_accuracy: 0.5974
Epoch 7/200
17/17 [==============================] - 0s 2ms/step - loss: 1.0455 - accurac
y: 0.6443 - val_loss: 0.9170 - val_accuracy: 0.6277
Epoch 8/200
17/17 [==============================] - 0s 2ms/step - loss: 0.9850 - accurac
y: 0.6387 - val_loss: 0.8651 - val_accuracy: 0.6147
Epoch 9/200
17/17 [==============================] - 0s 2ms/step - loss: 0.9385 - accurac
y: 0.6555 - val_loss: 0.8296 - val_accuracy: 0.6190
Epoch 10/200
17/17 [==============================] - 0s 2ms/step - loss: 0.8996 - accurac
y: 0.6369 - val_loss: 0.8106 - val_accuracy: 0.6234
Epoch 11/200
17/17 [==============================] - 0s 2ms/step - loss: 0.8752 - accurac
y: 0.6499 - val_loss: 0.7966 - val_accuracy: 0.6190
Epoch 12/200
17/17 [==============================] - 0s 2ms/step - loss: 0.8465 - accurac
y: 0.6462 - val_loss: 0.7838 - val_accuracy: 0.6494
Epoch 13/200
```

In [12]:

```
# evaluate the model
scores = model.evaluate(X, Y)
print(scores)
```

```
24/24 [==============================] - 0s 1ms/step - loss: 0.5422 - accuracy:
0.7344
[0.5422187447547913, 0.734375]
```

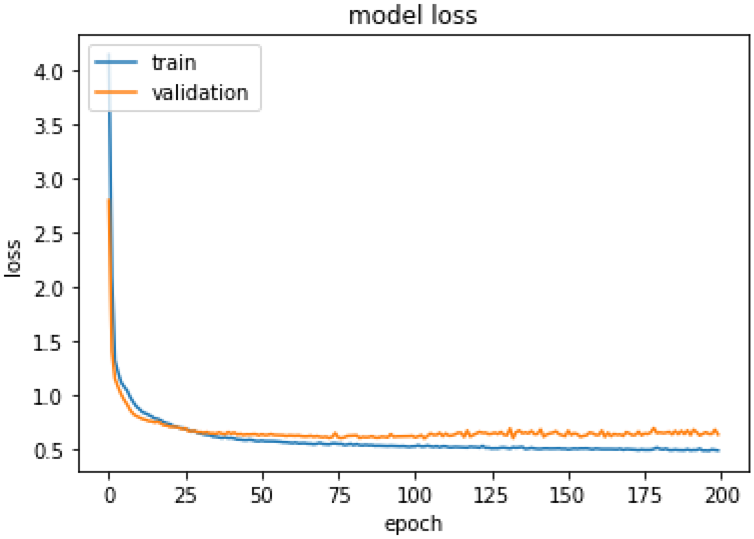
In [13]:

```
import matplotlib.pyplot as plt
```

In [14]:

```
print(history.history.keys())
# "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', 'validation'], loc='upper left')
plt.show()
```

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



In [15]:

```
predictions = model.predict(X)
# round predictions
rounded = [round(x[0]) for x in predictions]
print(rounded[:5])
```

```
[1, 0, 1, 0, 1]
```

In [16]:

```
X_new= X[0:5, :]
X_new
```

Out[16]:

```
array([[6.000e+00, 1.480e+02, 7.200e+01, 3.500e+01, 0.000e+00, 3.360e+01,
        6.270e-01, 5.000e+01],
       [1.000e+00, 8.500e+01, 6.600e+01, 2.900e+01, 0.000e+00, 2.660e+01,
        3.510e-01, 3.100e+01],
       [8.000e+00, 1.830e+02, 6.400e+01, 0.000e+00, 0.000e+00, 2.330e+01,
        6.720e-01, 3.200e+01],
       [1.000e+00, 8.900e+01, 6.600e+01, 2.300e+01, 9.400e+01, 2.810e+01,
        1.670e-01, 2.100e+01],
       [0.000e+00, 1.370e+02, 4.000e+01, 3.500e+01, 1.680e+02, 4.310e+01,
        2.288e+00, 3.300e+01]])
```

In [17]:

```
y_new = model.predict(X_new)
# round predictions
rounded = [round(x[0]) for x in y_new]
print(rounded)
```

```
[1, 0, 1, 0, 1]
```

In [18]:

```
y = Y[0:5]
y
```

Out[18]:

```
array([1., 0., 1., 0., 1.])
```

In [ ]: