DEEP LEARNING
BÀI TẬP
//Tuần lễ: 03/10-------09/10

Tensorflow

1. Constant trong tensorflow

```
import tensorflow.compat.v1 as tf
x = tf.constant(5,tf.float32)
y = tf.constant([5], tf.float32)
z = tf.constant([5,3,4], tf.float32)
t = tf.constant([[5,3,4,6],[2,3,4,7]], tf.float32)
u = tf.constant([[[5,3,4,6],[2,3,4,0]]], tf.float32)
v = tf.constant([[[5,3,4,6],[2,3,4,0]],
                 [[5,3,4,6],[2,3,4,0]],
                 [[5,3,4,6],[2,3,4,0]]
                ], tf.float32)
print(y)
……
```

2. Variable trong tensorflow

```
==================================
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x1 = tf.Variable(5.3, tf.float32)
x2 = tf.Variable(4.3, tf.float32)
x  = tf.multiply(x1,x2)

init = tf.global_variables_initializer()
with tf.Session() as sess:
     sess.run(init)
     t = sess.run(x)
     print(t)
==================================
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x1 = tf.Variable([[5.3,4.5,6.0],
                  [4.3,4.3,7.0]
                 ], tf.float32)
x2 = tf.Variable([[4.3,4.3,7.0],
                  [5.3,4.5,6.0]
                 ], tf.float32)
x  = tf.multiply(x1,x2)
```

```
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    t = sess.run(x)
    print(t)
```

3. Placeholder

```
=======
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x = tf.placeholder(tf.float32,None)
y = tf.add(x,x)

with tf.Session() as sess:
    x_data= 5
    result = sess.run(y,feed_dict={x:x_data})
    print(result)
=========
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x = tf.placeholder(tf.float32,[None,3])
y = tf.add(x,x)

with tf.Session() as sess:
    x_data= [[1.5, 2.0, 3.3]]
    result = sess.run(y,feed_dict={x:x_data})
    print(result)
========
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x = tf.placeholder(tf.float32,[None,None,3])
y = tf.add(x,x)

with tf.Session() as sess:
    x_data= [[[1,2,3]]]
    result = sess.run(y,feed_dict={x:x_data})
    print(result)
======
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x = tf.placeholder(tf.float32,[None,4,3])
y = tf.add(x,x)

with tf.Session() as sess:
    x_data= [[[1,2,3],
```

```
              [2,3,4],
              [2,3,5],
              [0,1,2]
            ]]
    result = sess.run(y,feed_dict={x:x_data})
    print(result)
========
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x = tf.placeholder(tf.float32,[2,4,3])
y = tf.add(x,x)

with tf.Session() as sess:
    x_data= [[[1,2,3],
              [2,3,4],
              [2,3,5],
              [0,1,2]
            ],
            [[1,2,3],
              [2,3,4],
              [2,3,5],
              [0,1,2]
            ]]
    result = sess.run(y,feed_dict={x:x_data})
    print(result)
========
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x = tf.placeholder(tf.float32,[2,4,3])
y = tf.placeholder(tf.float32,[2,4,3])

z = tf.add(x,y)
u = tf.multiply(x,y)

with tf.Session() as sess:
    x_data= [[[1,2,3],
              [2,3,4],
              [2,3,5],
              [0,1,2]
            ],
            [[1,2,3],
              [2,3,4],
              [2,3,5],
              [0,1,2]
            ]]
    y_data= [[[1,2,3],
```

```
                  [2,3,4],
                  [2,3,5],
                  [0,1,2]
                ],
                [[1,2,3],
                  [2,3,4],
                  [2,3,5],
                  [0,1,2]
                ]]
        result1 = sess.run(z,feed_dict={x:x_data, y:y_data})
        result2 = sess.run(u,feed_dict={x:x_data, y:y_data})
        print("result1 =", result1)
        print("result2 =", result2)
    ========
```

4. Operation

```
    import tensorflow.compat.v1 as tf
    tf.compat.v1.disable_eager_execution()

    x1 = tf.constant(5.3, tf.float32)
    x2 = tf.constant(1.5, tf.float32)
    w1 = tf.Variable(0.7, tf.float32)
    w2 = tf.Variable(0.5, tf.float32)
    u = tf.multiply(x1,w1)
    v = tf.multiply(x2,w2)
    z = tf.add(u,v)
    result = tf.sigmoid(z)
    init = tf.global_variables_initializer()

    with tf.Session() as sess:
        sess.run(init)
        print(sess.run(result))

===
```

```
===
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x1 = tf.placeholder(tf.float32,[None,3])
x2 = tf.placeholder(tf.float32,[None,3])

w1 = tf.Variable([0.5,0.4,0.7],tf.float32)
w2 = tf.Variable([0.8,0.5,0.6], tf.float32)
```

```
u1 = tf.multiply(w1,x1)
u2 = tf.multiply(w2,x2)
v = tf.add(u1,u2)
z = tf.sigmoid(v)

init = tf.global_variables_initializer()

with tf.Session() as sess:
    x1_data= [[1,2,3]]
    x2_data= [[1,2,3]]
    sess.run(init)
    result = sess.run(z,feed_dict={x1:x1_data, x2:x2_data})
    print(result)
====
```

TensorFlow operator	Shortcut	Description
`tf.add()`	`a + b`	Adds a and b, element-wise.
`tf.multiply()`	`a * b`	Multiplies a and b, element-wise.
`tf.subtract()`	`a - b`	Subtracts a from b, element-wise.
`tf.divide()`	`a / b`	Computes Python-style division of a by b.
`tf.pow()`	`a ** b`	Returns the result of raising each element in a to its corresponding element b, element-wise.
`tf.mod()`	`a % b`	Returns the element-wise modulo.
`tf.logical_and()`	`a & b`	Returns the truth table of `a & b`, element-wise. `dtype` must be `tf.bool`.
`tf.greater()`	`a > b`	Returns the truth table of `a > b`, element-wise.
`tf.greater_equal()`	`a >= b`	Returns the truth table of `a >= b`, element-wise.
`tf.less_equal()`	`a <= b`	Returns the truth table of `a <= b`, element-wise.
`tf.less()`	`a < b`	Returns the truth table of `a < b`, element-wise.
`tf.negative()`	`-a`	Returns the negative value of each element in a.
`tf.logical_not()`	`~a`	Returns the logical NOT of each element in a. Only compatible with Tensor objects with `dtype` of `tf.bool`.
`tf.abs()`	`abs(a)`	Returns the absolute value of each element in a.
`tf.logical_or()`	`a | b`	Returns the truth table of `a | b`, element-wise. `dtype` must be `tf.bool`.

Data type	Python type	Description
DT_FLOAT	tf.float32	32-bit floating point.
DT_DOUBLE	tf.float64	64-bit floating point.
DT_INT8	tf.int8	8-bit signed integer.
DT_INT16	tf.int16	16-bit signed integer.
DT_INT32	tf.int32	32-bit signed integer.
DT_INT64	tf.int64	64-bit signed integer.
DT_UINT8	tf.uint8	8-bit unsigned integer.
DT_UINT16	tf.uint16	16-bit unsigned integer.
DT_STRING	tf.string	Variable-length byte array. Each element of a Tensor is a byte array.
DT_BOOL	tf.bool	Boolean.
DT_COMPLEX64	tf.complex64	Complex number made of two 32-bit floating points: real and imaginary parts.
DT_COMPLEX128	tf.complex128	Complex number made of two 64-bit floating points: real and imaginary parts.
DT_QINT8	tf.qint8	8-bit signed integer used in quantized ops.
DT_QINT32	tf.qint32	32-bit signed integer used in quantized ops.
DT_QUINT8	tf.quint8	8-bit unsigned integer used in quantized ops.

DEEP LEARNING & KERAS

1. Model Neuron Network
   a. Hãy xem và hiểu (tham khảo [1], trg 18)

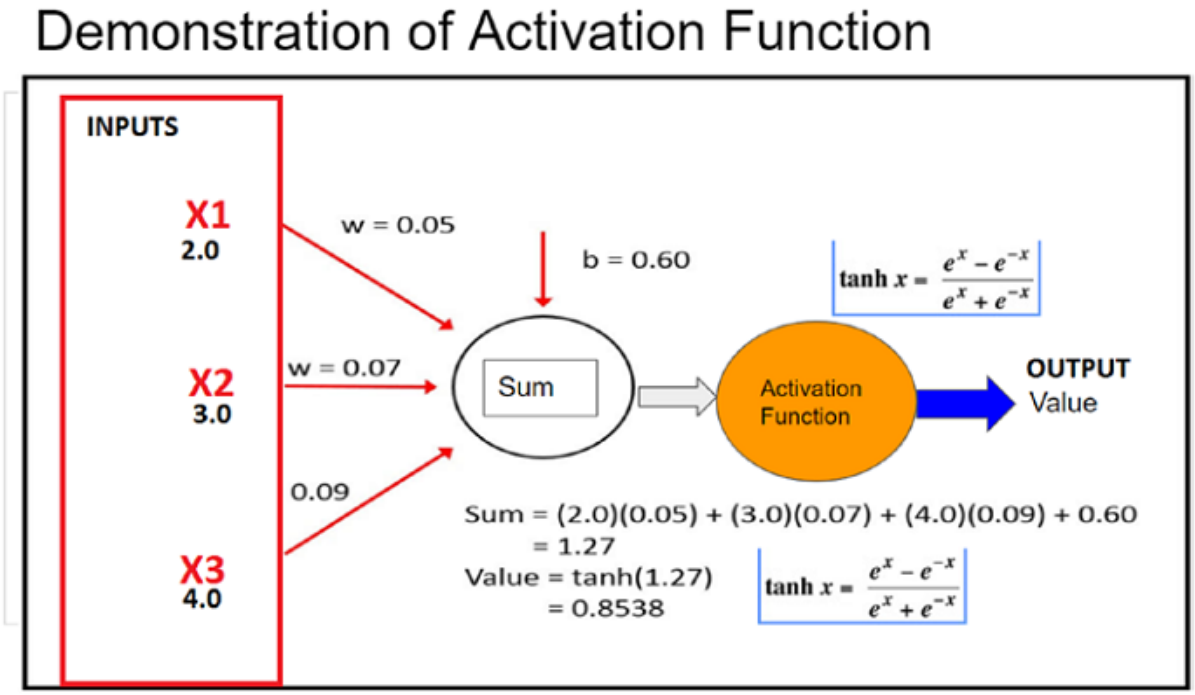
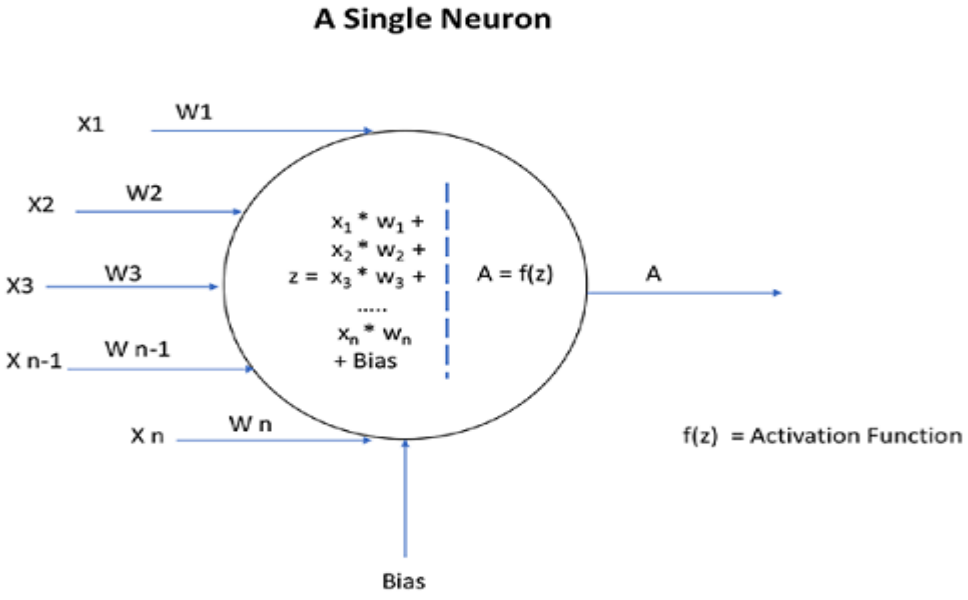
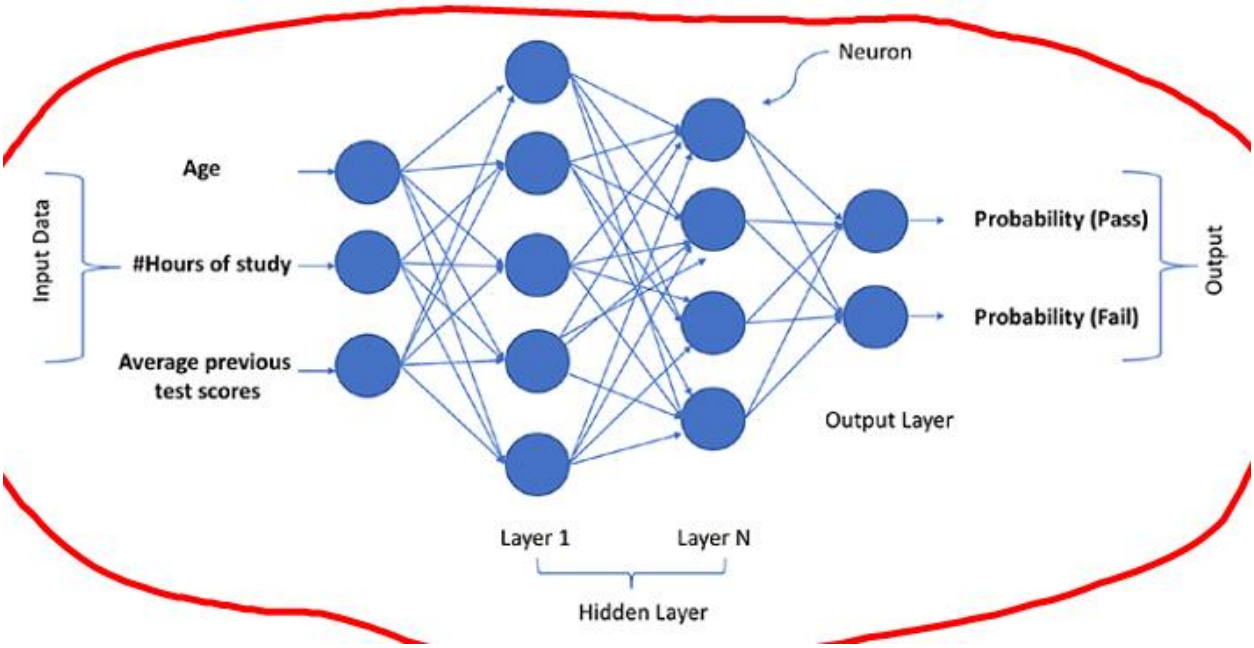


Figure 1-1. *An activation function*

   b. Hãy code mô tả trên và thay hàm tanh bởi các hàm khác nhau ([1] trang 18-19) như sigmoid, relu… Sử dụng và hiểu **tf.nn.tanh(x).** Copy code và ảnh chương trình chạy

2. Xem cấu trúc phức tạp hơn trong Hình và giải thích





3. Chạy ví dụ sau và Giải thích (tài liệu [0.2] trg 13)

```
======
#Import required packages
from keras.models import Sequential
from keras.layers import Dense
import numpy as np

# Getting the data ready
# Generate train dummy data for 1000 Students and dummy testfor
500
#Columns :Age, Hours of Study &Avg Previous test scores
np.random.seed(2018) #Setting seed for reproducibility
train_data, test_data = np.random.random((1000, 3)),
np.random.random((500, 3))
#Generate dummy results for 1000 students : Whether Passed (1) or
Failed (0)
labels = np.random.randint(2, size=(1000, 1))
#Defining the model structure with the required layers,
# ofneurons, activation function and optimizers
model = Sequential()
model.add(Dense(5, input_dim=3, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
#Train the model and make predictions
model.fit(train_data, labels, epochs=10, batch_size=32)
#Make predictions from the trained model
```

```
predictions = model.predict(test_data)
======
```

4. Chạy và giải thích (tham khảo [0.2])

```
====
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation

# Generate dummy training dataset
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))
#Define the model architecture
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'])
#Train the model
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)
print(model.evaluate(x_test,y_test))
print(model.metrics_names)
#print 10 predictions
pred = model.predict(x_test)
pred[:10]
=======
```

5. Chạy ví dụ trong Chap 3 ([0.2])
6. Chạy ví dụ trong Chap 4 ([0.2])