

PHÁT TRIỂN HỆ THỐNG MINH

Môn học Phát triển Hệ thống minh gồm 3 phần:

Phần I: Machine Learning cơ bản

Phần này tập trung xem xét những vấn đề liên quan đến ML bao gồm các kỹ thuật học có giám sát (supervised learning) và học không có giám sát (unsupervised learning). Chúng ta đã trải nghiệm sử dụng các thư viện *numpy*, *pandas*, *matplotlib* để cài đặt và triển khai một số ví dụ áp dụng. Các chủ đề sinh viên cần nắm vững:

- Các kỹ thuật xử lý dữ liệu
- Các kỹ thuật thử nghiệm, đánh giá kết quả
- Các kỹ thuật học có giám sát
- Các kỹ thuật học không có giám sát
- Sử dụng các thư viện để thử nghiệm và đánh giá

Phần II Phát triển ứng dụng

- E-commerce
- Medicine (diagnose, assist,...)
- Management Systems (Library, Hospital Appointment, Tourist...)
- ...

Phần III Deep learning

Phần này bao gồm 2 phần:

- Tensorflow
- Các kỹ thuật trong deep learning với keras

=====

DEEP LEARNING

Tensorflow

Cài đặt tensorflow: **`pip install tensorflow`** hay **`conda install tensorflow`**

Cài đặt keras: **`pip install keras`**

Introduction to Tensorflow

<https://www.geeksforgeeks.org/introduction-to-tensorflow/>

Kỹ thuật trong deep Learning với keras

https://keras.io/getting_started/

<https://www.tensorflow.org/tutorials/quickstart/beginner>

https://keras.io/getting_started/intro_to_keras_for_engineers/

=====

BÀI TẬP 3: TENSORFLOW

Due date: 18-19/09/2024

3.1. Chạy và giải thích (hằng **Constant** trong tensorflow)

```
import tensorflow as tf
h = tf.constant("Hello")
w = tf.constant(" World!")
hw = h + w
with tf.Session() as sess:
    ans = sess.run(hw)
print (ans)
=====

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(v)
```

3.2. Chạy và giải thích (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)

```

```

import tensorflow.compat.v1 as tf

# creating nodes in computation graph
node = tf.Variable(tf.zeros([2,2]))

# running computation graph
with tf.Session() as sess:

    # initialize all global variables
    sess.run(tf.global_variables_initializer())

    # evaluating node
    print("Tensor value before addition:\n",sess.run(node))

    # elementwise addition to tensor
    node = node.assign(node + tf.ones([2,2]))

    # evaluate node again
    print("Tensor value after addition:\n", sess.run(node))
    sess.close()

```

3.3. Chạy và giải thích (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)
=====

```

3.4. 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 numpy as np
import matplotlib.pyplot as plt

number_of_points = 500
x_point = []
y_point = []
a = 0.22
b = 0.78

for i in range(number_of_points):
    x = np.random.normal(0.0,0.5)
    y = a*x + b +np.random.normal(0.0,0.1)
    x_point.append([x])
    y_point.append([y])

plt.plot(x_point,y_point, 'o', label = 'Input Data')
plt.legend()
plt.show()

===

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)
====
import tensorflow as tf
import numpy as np

matrix1 = np.array([(2,2,2),(2,2,2),(2,2,2)],dtype = 'int32')
matrix2 = np.array([(1,1,1),(1,1,1),(1,1,1)],dtype = 'int32')

print (matrix1)
print (matrix2)

matrix1 = tf.constant(matrix1)
matrix2 = tf.constant(matrix2)
matrix_product = tf.matmul(matrix1, matrix2)
matrix_sum = tf.add(matrix1,matrix2)
matrix_3 = np.array([(2,7,2),(1,4,2),(9,0,2)],dtype = 'float32')
print (matrix_3)

matrix_det = tf.matrix_determinant(matrix_3)
with tf.Session() as sess:
    result1 = sess.run(matrix_product)
    result2 = sess.run(matrix_sum)
    result3 = sess.run(matrix_det)

print (result1)
print (result2)
print (result3)

```

3.5. Chạy ví dụ và giải thích **Linear Regression model** using TensorFlow Core API.

```

# importing the dependencies
import tensorflow.compat.v1 as tf
import numpy as np
import matplotlib.pyplot as plt

```

```

# Model Parameters
learning_rate = 0.01
training_epochs = 2000
display_step = 200

# Training Data
train_X =
np.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,2.167,

7.042,10.791,5.313,7.997,5.654,9.27,3.1])
train_y =
np.asarray([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.22
1,
2.827,3.465,1.65,2.904,2.42,2.94,1.3])
n_samples = train_X.shape[0]

# Test Data
test_X = np.asarray([6.83, 4.668, 8.9, 7.91, 5.7, 8.7, 3.1,
2.1])

# Set placeholders for feature and target vectors
X = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

# Set model weights and bias)
test_y = np.asarray([1.84, 2.273, 3.2, 2.831, 2.92, 3.24, 1.35,
1.03])
W = tf.Variable(np.random.randn(), name="weight")
b = tf.Variable(np.random.randn(), name="bias")

# Construct a linear model
linear_model = W*X + b

# Mean squared error
cost = tf.reduce_sum(tf.square(linear_model - y)) /
(2*n_samples)

# Gradient descent
optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Initializing the variables
init = tf.global_variables_initializer()

# Launch the graph
with tf.Session() as sess:
    # Load initialized variables in current session
    sess.run(init)

```



```

# Fit all training data
for epoch in range(training_epochs):

    # perform gradient descent step
    sess.run(optimizer, feed_dict={X: train_X, y: train_y})

    # Display logs per epoch step
    if (epoch+1) % display_step == 0:
        c = sess.run(cost, feed_dict={X: train_X, y:
train_y})
        print("Epoch:{0:6} \t Cost:{1:10.4} \t W:{2:6.4} \t
b:{3:6.4}".
              format(epoch+1, c, sess.run(W), sess.run(b)))

    # Print final parameter values
    print("Optimization Finished!")
    training_cost = sess.run(cost, feed_dict={X: train_X, y:
train_y})
    print("Final training cost:", training_cost, "W:",
sess.run(W), "b:",
        sess.run(b), '\n')

    # Graphic display
    plt.plot(train_X, train_y, 'ro', label='Original data')
    plt.plot(train_X, sess.run(W) * train_X + sess.run(b),
label='Fitted line')
    plt.legend()
    plt.show()

    # Testing the model
    testing_cost = sess.run(tf.reduce_sum(tf.square(linear_model
- y)) / (2 * test_X.shape[0]),
                          feed_dict={X: test_X, y: test_y})

    print("Final testing cost:", testing_cost)
    print("Absolute mean square loss difference:",
abs(training_cost - testing_cost))

    # Display fitted line on test data
    plt.plot(test_X, test_y, 'bo', label='Testing data')
    plt.plot(train_X, sess.run(W) * train_X + sess.run(b),
label='Fitted line')
    plt.legend()
    plt.show()

```

3.6. Model Neuron Network

a. Hãy xem và hiểu [1]

Demonstration of Activation Function

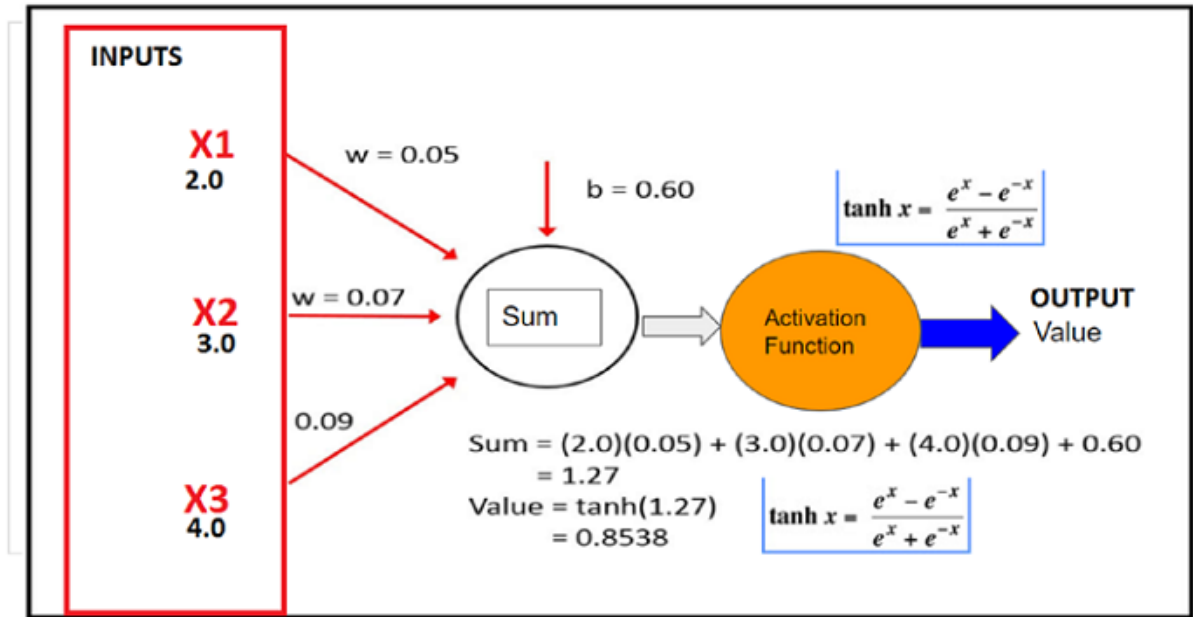
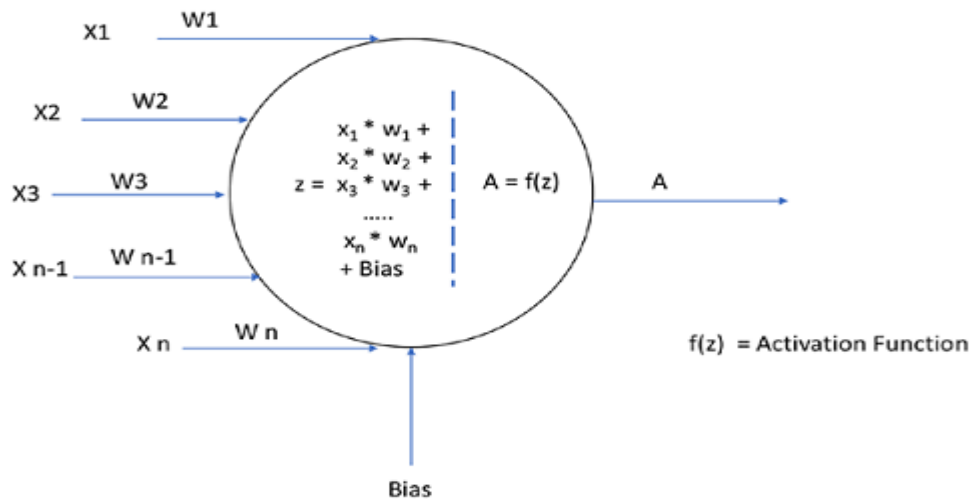


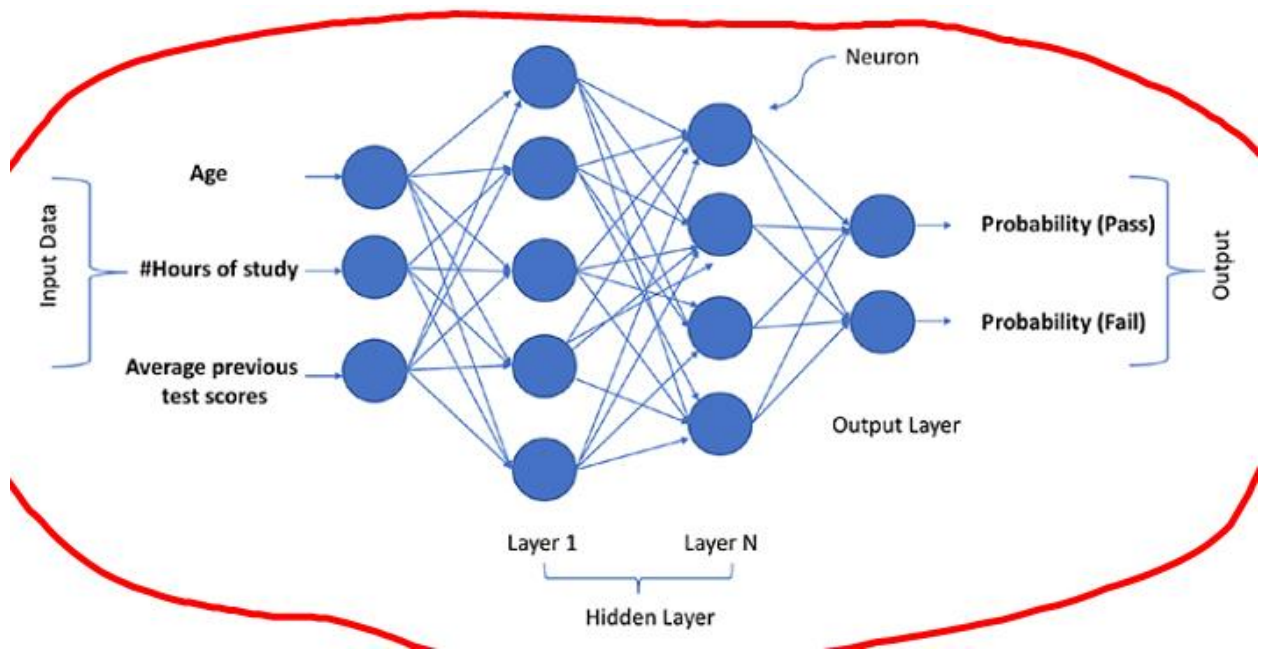
Figure 1-1. An activation function

- b. Hãy thiết kế mạng neuron đơn giản như trình bày trong biểu đồ. Có thể 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

3.7 Sử dụng tensorflow để thiết kế với cấu trúc phức tạp hơn trong Hình và giải thích

A Single Neuron





=====

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

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