人工智能实践：TENSORFLOW2（五）

循环神经网络（Recurrent Neural Network， RNN）

循环神经网络（RNN）

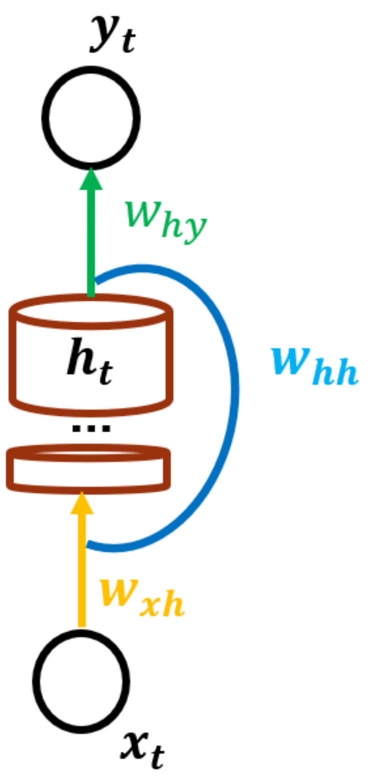
CNN: 借助卷积核(kernel)提取特征后，送入后续网络(如全连接网络Dense)进行分类、目标检测等操作。CNN借助卷积核从空间维度提取信息，卷积核参数空间共享。元素之间是相互独立的，**输入与输出也是独立的。**

RNN: 借助循环核(cell)提取特征后，送入后续网络(如全连接网络Dense)进行预测等操作。RNN借助循环核从时间维度提取信息，循环核参数时间共享。循环神经网络本质是：**像人一样拥有记忆的能力。**因此，他的输出就依赖于当前的输入和记忆。

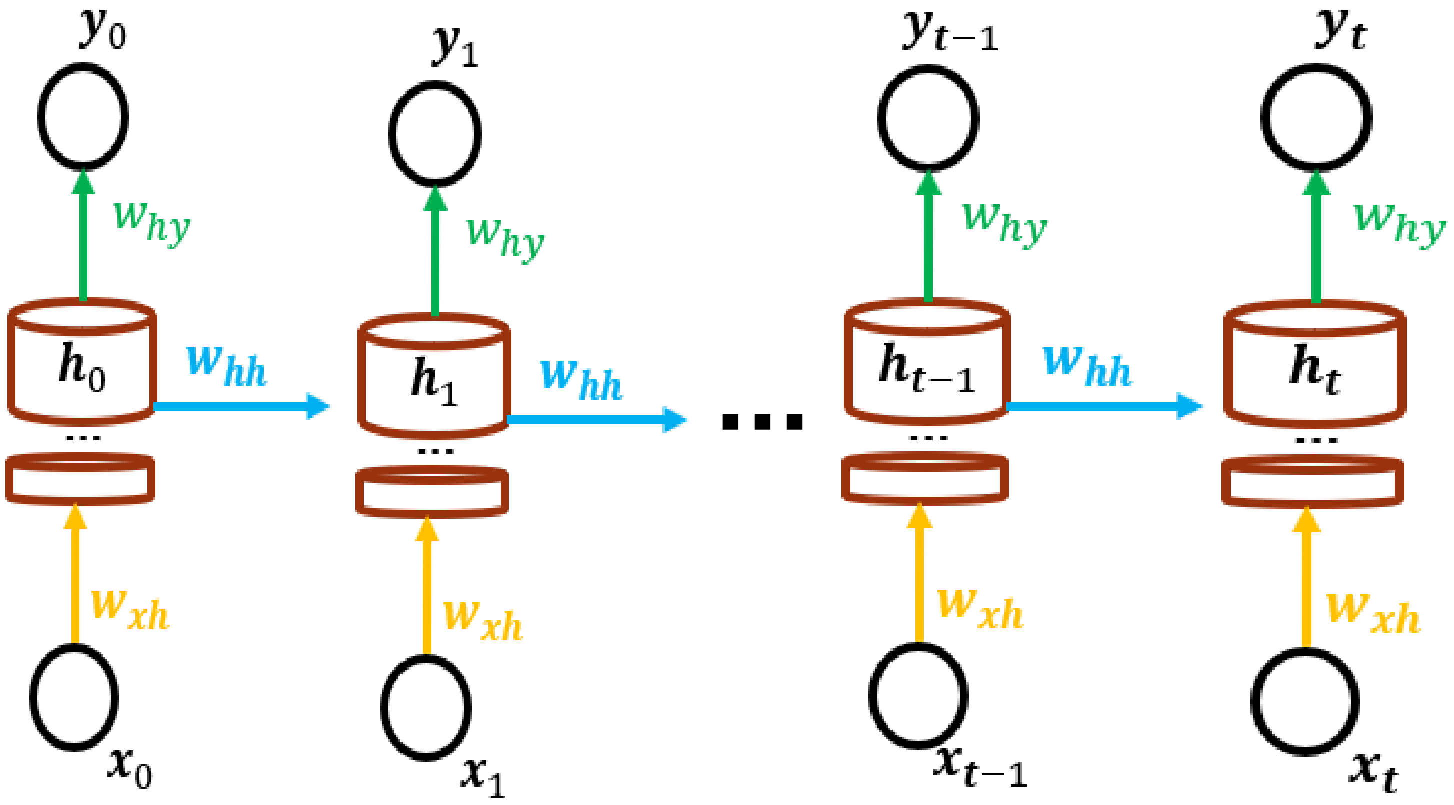
循环神经网络

- 循环核
- 循环核时间步展开
- 循环计算层
- TF描述循环计算层
- 循环计算过程

循环核



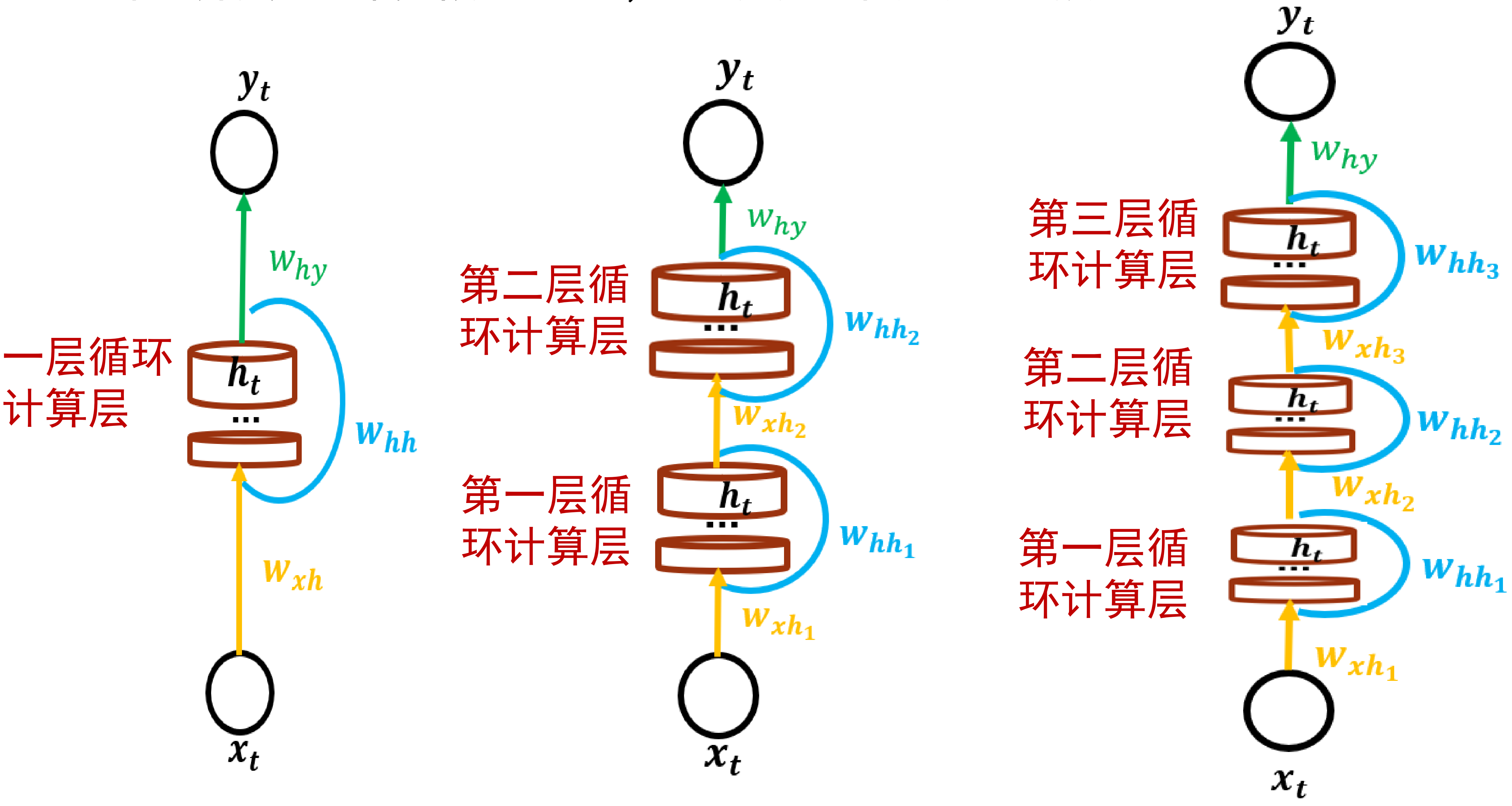
循环核按时间步展开



$$y_t = softmax(h_t w_{hy} + b_y)$$
$$h_t = tanh(x_t w_{xh} + h_{t-1} w_{hh} + b_h)$$

循环神经网络：借助循环核提取时间特征后，送入全连接网络。

循环计算层：向输出方向生长，不同的循环核纵向连接。



TF描述循环计算层

tf.keras.layers.SimpleRNN(记忆体个数， activation=‘激活函数’ ，
return_sequences=是否每个时刻输出ht到下一层)

其中 activation=‘激活函数’ （不写，默认使用tanh）

return_sequences=True #循环核各时刻会把ht推送到下一层

return_sequences=False #循环核仅最后一个时刻把ht推送到下一层

例： SimpleRNN(3, return_sequences=True)

定义了一个具有三个记忆体的循环核， 这个循环核会在每个时间步输出h_t

return_sequences： 在输出序列中，返回最后时刻的输出值h_t，还是返回全部时刻h_t的输出。 False 返回最后时刻(图 1)， True 返回全部时刻(图2)。

当下一层依然是 RNN 层，通常为 True；如果后面是 Dense 层，通常为 Fasle。

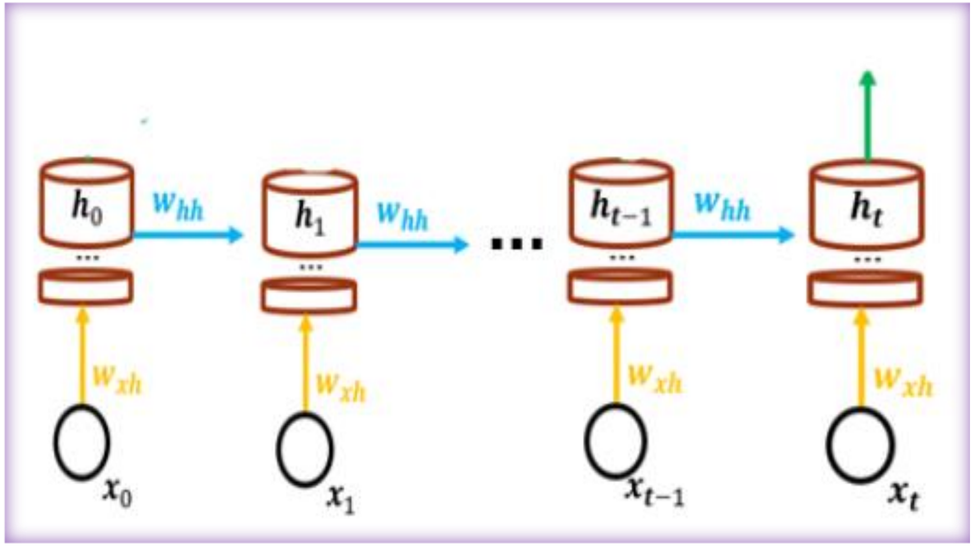


图 1

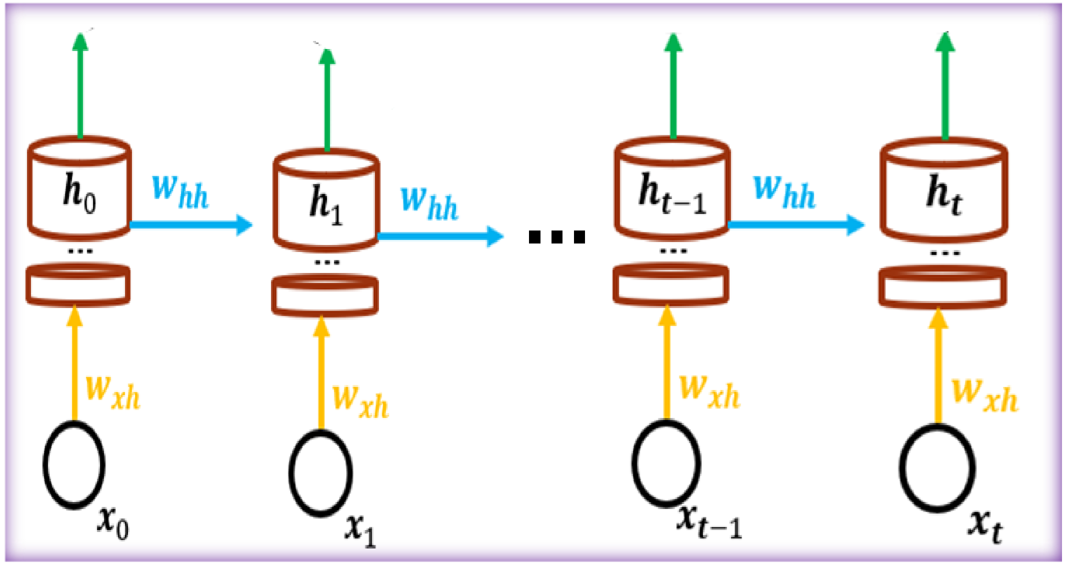
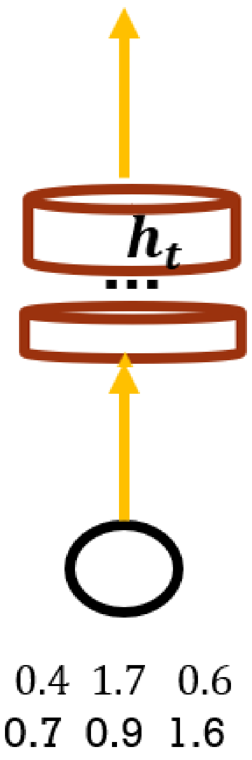
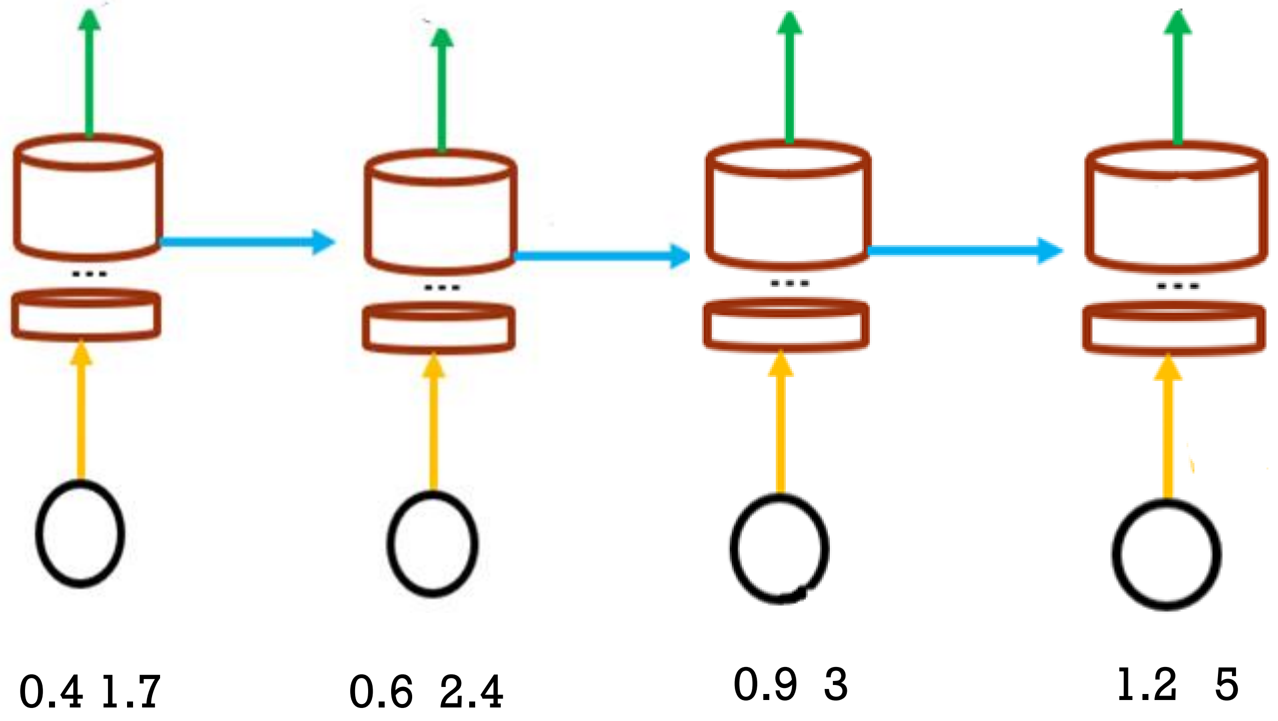


图 2

输入RNN时, x_train维度：[送入样本数，循环核时间展开步数，每个时间步输入特征个数]



x_train维度：[2,1,3]



x_train维度：[1,4,2]

输出维度：当 return_sequenc=True，三维张量(输入样本数, 循环核时间展开步数,本层的神经元个数)；当 return_sequenc=False，二维张量(输入样本数,本层的神经元个数)

RNN 最典型的应用就是利用历史数据预测下一时刻将发生什么，即根据以前见过的历史规律做预测。

循环计算过程

字母预测：输入a预测出b，输入b预测出c，输入c预测出d，输入d预测出e，输入e预测出a

独热码编码

10000	a
01000	b
00100	c
00010	d
00001	e

输出

y_t: 0.02 0.02 0.91 0.03 0.02

w_{hy}

$$[[-1.7\ 0.7\ -1.7\ 1.7\ 0.7]\ [-1.6\ -1.6\ 0.7\ 1.3\ 1.4]\ [-1.4\ 1.9\ 1.2\ 1.7\ -1.9]]$$

b_y

$$[0.0\ 0.1\ 0.4\ -0.7\ 0.1]$$

h_t: -0.9，0.8，0.7

w_{hh}

$$[[-0.9\ -0.2\ -0.4]\ [-0.3\ 0.9\ 0.2]\ [0.4\ 0.3\ -0.9]]$$

w_{xh}

$$[[0.5\ -1.7\ 1.7]\ [-2.3\ 0.8\ 1.1]\ [1.3\ 1.7\ 1.4]\ [0.3\ 0.8\ -1.1]\ [-1.8\ -2.0\ -1.0]]$$

b_h

$$[0.5\ 0.3\ -0.2]$$

x_t: 0，1，0，0，0

输入

$$y_t = softmax(h_t w_{hy} + b_y)$$
$$= softmax([-0.7\ -0.6\ 2.9\ 0.7\ -0.8] + [0.0\ 0.1\ 0.4\ -0.7\ 0.1])$$
$$= softmax([-0.7\ -0.5\ 3.3\ 0.0\ -0.7])$$
$$= [0.02\ 0.02\ \color{red}{0.91}\ 0.03\ 0.02]$$

$$h_t = tanh(x_t w_{xh} + h_{t-1} w_{hh} + b_h)$$
$$= tanh([-2.3\ 0.8\ 1.1] + 0 + [0.5\ 0.3\ -0.2])$$
$$= [-0.9\ 0.8\ 0.7]$$

用RNN实现输入一个字母，预测下一个字母（One hot编码）

[1]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Dense, SimpleRNN
import matplotlib.pyplot as plt
import os
```

[2]:
```
input_word = "abcde"
w_to_id = {'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4}  # 单词映射到数值id的词典
id_to_onehot = {0: [1., 0., 0., 0., 0.], 1: [0., 1., 0., 0., 0.], 2: [0., 0., 1., 0., 0.], 3: [0., 0., 0., 1., 0.],
                4: [0., 0., 0., 0., 1.]}  # id编码为one-hot
```

[3]:
```
x_train = [id_to_onehot[w_to_id['a']], id_to_onehot[w_to_id['b']], id_to_onehot[w_to_id['c']],
           id_to_onehot[w_to_id['d']], id_to_onehot[w_to_id['e']]]
y_train = [w_to_id['b'], w_to_id['c'], w_to_id['d'], w_to_id['e'], w_to_id['a']]
```

[4]:
```
np.random.seed(7)
np.random.shuffle(x_train)
np.random.seed(7)
np.random.shuffle(y_train)
```

```
[5]: x_train = np.reshape(x_train, (len(x_train), 1, 5))
     y_train = np.array(y_train)
```

```
[6]: model = tf.keras.Sequential([
         SimpleRNN(3),
         Dense(5, activation='softmax')
     ])
```

```
[7]: model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
                   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
                   metrics=['sparse_categorical_accuracy'])
```

```
[8]: checkpoint_save_path = "G:/教学/生产实习/资料/RNN/checkpoint/rnn_onehot_1pre1.ckpt"

if os.path.exists(checkpoint_save_path + '.index'):
    print('-------------load the model-----------------')
    model.load_weights(checkpoint_save_path)

cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_save_path,
                                                 save_weights_only=True,
                                                 save_best_only=True,
                                                 monitor='loss')  # 由于fit没有给出测试集，不计算测试集准确率，根据loss，保存最优模型

history = model.fit(x_train, y_train, batch_size=32, epochs=100, callbacks=[cp_callback])

model.summary()
```

```
Epoch 97/100
5/5 [==============================] - 0s 5ms/sample - loss: 0.2741 - sparse_categorical_accuracy: 1.0000
Epoch 98/100
5/5 [==============================] - 0s 5ms/sample - loss: 0.2684 - sparse_categorical_accuracy: 1.0000
Epoch 99/100
5/5 [==============================] - 0s 5ms/sample - loss: 0.2628 - sparse_categorical_accuracy: 1.0000
Epoch 100/100
5/5 [==============================] - 0s 5ms/sample - loss: 0.2574 - sparse_categorical_accuracy: 1.0000
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       multiple                  27
_________________________________________________________________
dense (Dense)                multiple                  20
=================================================================
Total params: 47
Trainable params: 47
```

[9]:
```python
# print(model.trainable_variables)
file = open('G:/教学/生产实习/资料/RNN/weights.txt', 'w')  # 参数提取
for v in model.trainable_variables:
    file.write(str(v.name) + '\n')
    file.write(str(v.shape) + '\n')
    file.write(str(v.numpy()) + '\n')
file.close()
```

[10]:
```python
preNum = int(input("input the number of test alphabet:"))
for i in range(preNum):
    alphabet1 = input("input test alphabet:")
    alphabet = [id_to_onehot[w_to_id[alphabet1]]]
    # 使alphabet符合SimpleRNN输入要求：[送入样本数， 循环核时间展开步数，
    #每个时间步输入特征个数]。此处验证效果送入了1个样本，送入样本数为1;
    #输入1个字母出结果，所以循环核时间展开步数为1; 表示为独热码有5个输入特征，每个时间步输入特征个数为5
    alphabet = np.reshape(alphabet, (1, 1, 5))
    result = model.predict([alphabet])
    pred = tf.argmax(result, axis=1)
    pred = int(pred)
    tf.print(alphabet1 + '->' + input_word[pred])
```

```
input the number of test alphabet:3
input test alphabet:a
a->b
input test alphabet:c
c->d
input test alphabet:e
e->a
```

循环计算过程

连续输入四个字母，预测下一个字母

输出

y_t: 0.02 0.02 0.91 0.03 0.02

w_{hy}

$$[[-1.7\ 0.7\ -1.7\ 1.7\ 0.7]\ [-1.6\ -1.6\ 0.7\ 1.3\ 1.4]\ [-1.4\ 1.9\ 1.2\ 1.7\ -1.9]]$$

b_y

$$[0.0\ 0.1\ 0.4\ -0.7\ 0.1]$$

h_t: -0.9，0.8，0.7

w_{hh}

$$[[-0.9\ -0.2\ -0.4]\ [-0.3\ 0.9\ 0.2]\ [0.4\ 0.3\ -0.9]]$$

w_{xh}

$$[[0.5\ -1.7\ 1.7]\ [-2.3\ 0.8\ 1.1]\ [1.3\ 1.7\ 1.4]\ [0.3\ 0.8\ -1.1]\ [-1.8\ -2.0\ -1.0]]$$

b_h

$$[0.5\ 0.3\ -0.2]$$

x_t: 0，1，0，0，0

输入

独热码编码

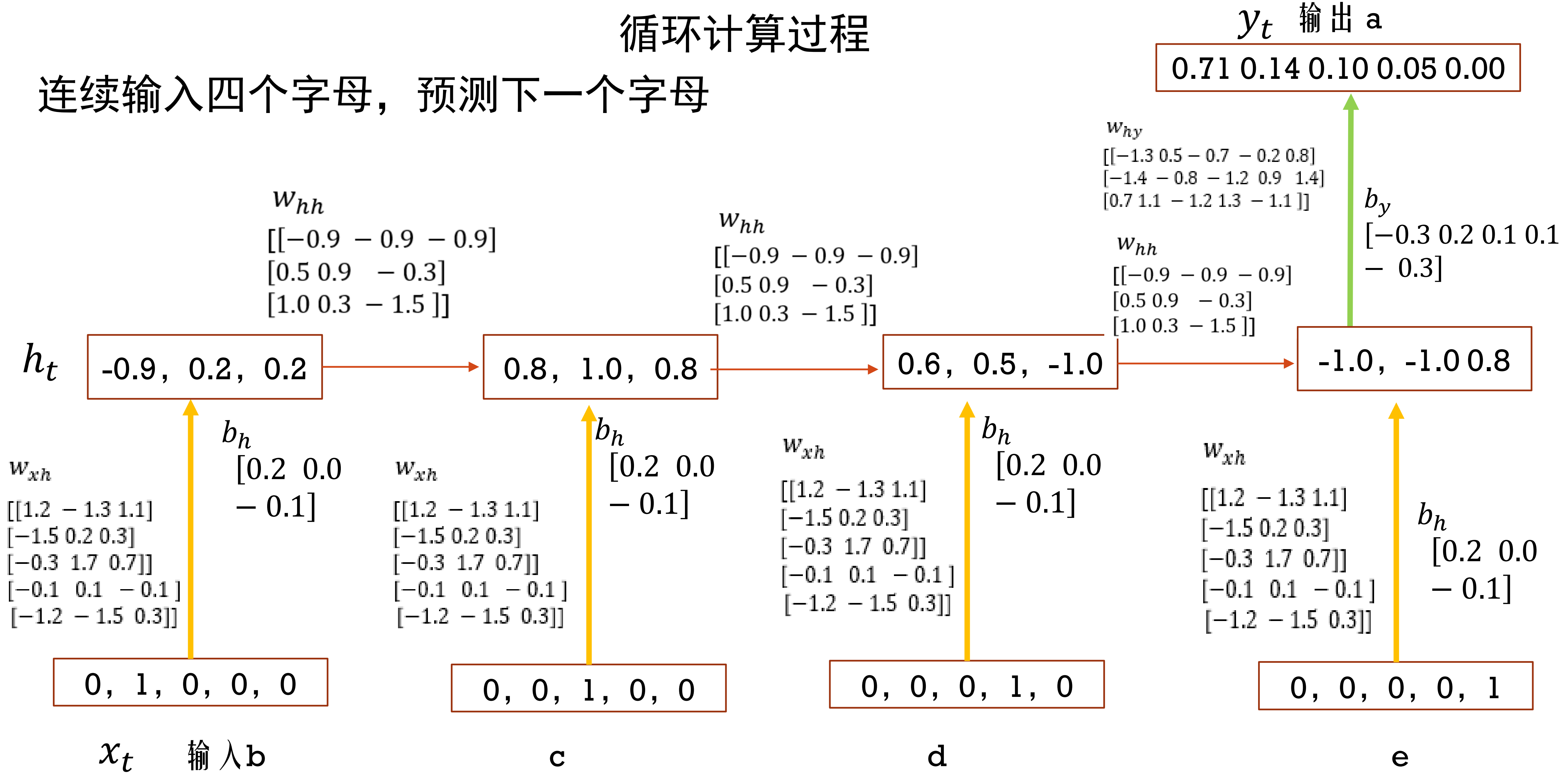
10000	a
01000	b
00100	c
00010	d
00001	e

$$y_t = softmax(h_t w_{hy} + b_y)$$
$$= softmax([-0.7\ -0.6\ 2.9\ 0.7\ -0.8] + [0.0\ 0.1\ 0.4\ -0.7\ 0.1])$$
$$= softmax([-0.7\ -0.5\ 3.3\ 0.0\ -0.7])$$
$$= [0.02\ 0.02\ \color{red}{0.91}\ 0.03\ 0.02]$$

$$h_t = tanh(x_t w_{xh} + h_{t-1} w_{hh} + b_h)$$
$$= tanh([-2.3\ 0.8\ 1.1] + 0 + [0.5\ 0.3\ -0.2])$$
$$= [-0.9\ 0.8\ 0.7]$$

循环计算过程

连续输入四个字母，预测下一个字母



$$y_t = softmax(h_t w_{hy} + b_y) \qquad h_t = tanh(x_t w_{xh} + h_{t-1} w_{hh} + b_h)$$

用RNN实现输入连续四个字母，预测下一个字母（One hot 编码）

(输入abcd输出e;输入bcde输出a;输入cdea输出b;

输入deab输出c;输入eabc输出d)

[1]:
```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Dense, SimpleRNN
import matplotlib.pyplot as plt
import os
```

[2]:
```python
input_word = "abcde"
w_to_id = {'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4}  # 单词映射到数值id的词典
id_to_onehot = {0: [1., 0., 0., 0., 0.], 1: [0., 1., 0., 0., 0.], 2: [0., 0., 1., 0., 0.], 3: [0., 0., 0., 1., 0.],
                4: [0., 0., 0., 0., 1.]}  # id编码为one-hot
```

[3]:
```python
x_train = [
    [id_to_onehot[w_to_id['a']], id_to_onehot[w_to_id['b']], id_to_onehot[w_to_id['c']], id_to_onehot[w_to_id['d']]],
    [id_to_onehot[w_to_id['b']], id_to_onehot[w_to_id['c']], id_to_onehot[w_to_id['d']], id_to_onehot[w_to_id['e']]],
    [id_to_onehot[w_to_id['c']], id_to_onehot[w_to_id['d']], id_to_onehot[w_to_id['e']], id_to_onehot[w_to_id['a']]],
    [id_to_onehot[w_to_id['d']], id_to_onehot[w_to_id['e']], id_to_onehot[w_to_id['a']], id_to_onehot[w_to_id['b']]],
    [id_to_onehot[w_to_id['e']], id_to_onehot[w_to_id['a']], id_to_onehot[w_to_id['b']], id_to_onehot[w_to_id['c']]],
]
y_train = [w_to_id['e'], w_to_id['a'], w_to_id['b'], w_to_id['c'], w_to_id['d']]
```

[4]:
```
np.random.seed(7)
np.random.shuffle(x_train)
np.random.seed(7)
np.random.shuffle(y_train)
```

[5]:
```
# 使x_train符合SimpleRNN输入要求：[送入样本数， 循环核时间展开步数， 每个时间步输入特征个数]。
# 此处整个数据集送入，送入样本数为len(x_train)；输入4个字母出结果，循环核时间展开步数为4; 表示
x_train = np.reshape(x_train, (len(x_train), 4, 5))
y_train = np.array(y_train)
```

[6]:
```
model = tf.keras.Sequential([
    SimpleRNN(3),
    Dense(5, activation='softmax')
])
```

[7]:
```
model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['sparse_categorical_accuracy'])
```

[7]:

```python
model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['sparse_categorical_accuracy'])
```

[8]:

```python
checkpoint_save_path = "G:/教学/生产实习/资料/RNN/checkpoint/rnn_onehot_4pre1.ckpt"

if os.path.exists(checkpoint_save_path + '.index'):
    print('-------------load the model-----------------')
    model.load_weights(checkpoint_save_path)

cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_save_path,
                                                 save_weights_only=True,
                                                 save_best_only=True,
                                                 monitor='loss')  # 由于fit没有给出测试集，不计算测试集准确率，根据loss，保存最优模

history = model.fit(x_train, y_train, batch_size=32, epochs=100, callbacks=[cp_callback])

model.summary()
```

```
Epoch 17/100
5/5 [==============================] - 0s 6ms/sample - loss: 1.2473 - sparse_categorical_accuracy: 0.8000
Epoch 18/100
5/5 [==============================] - 0s 5ms/sample - loss: 1.2290 - sparse_categorical_accuracy: 0.8000
```

[9]:
```python
# print(model.trainable_variables)
file = open('G:/教学/生产实习/资料/RNN/weights.txt', 'w')  # 参数提取
for v in model.trainable_variables:
    file.write(str(v.name) + '\n')
    file.write(str(v.shape) + '\n')
    file.write(str(v.numpy()) + '\n')
file.close()
```

[10]:
```python
preNum = int(input("input the number of test alphabet:"))
for i in range(preNum):
    alphabet1 = input("input test alphabet:")
    alphabet = [id_to_onehot[w_to_id[a]] for a in alphabet1]
    # 使alphabet符合SimpleRNN输入要求：[送入样本数， 循环核时间展开步数，
    #每个时间步输入特征个数]。此处验证效果送入了1个样本，送入样本数为1；输入4个字母出结果，
    #所以循环核时间展开步数为4；表示为独热码有5个输入特征，每个时间步输入特征个数为5
    alphabet = np.reshape(alphabet, (1, 4, 5))
    result = model.predict([alphabet])
    pred = tf.argmax(result, axis=1)
    pred = int(pred)
    tf.print(alphabet1 + '->' + input_word[pred])
```

```
input the number of test alphabet:1
input test alphabet:bcde
bcde->a
```

Embedding —— 一种编码方法

独热码：数据量大 过于稀疏，映射之间是独立的，没有表现出关联

Embedding：是一种单词编码方法，用低维向量实现了编码，这种编码通过神经网络训练优化，能表达出单词间的相关性。

tf.keras.layers.Embedding(词汇表大小，编码维度**)**

编码维度就是用几个数字表达一个单词

例 对1-100进行编码， [4] 编码为 [0.25, 0.1, 0.11]

tf.keras.layers.Embedding(100, 3)

输入**Embedding**时， **x_train**维度：**[**送入样本数， 循环核时间展开步数**]**

用RNN实现输入连续四个字母，预测下一个字母（Embedding 编码）

[1]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Dense, SimpleRNN, Embedding
import matplotlib.pyplot as plt
import os
```

[2]:
```
input_word = "abcdefghijklmnopqrstuvwxyz"
w_to_id = {'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4,
           'f': 5, 'g': 6, 'h': 7, 'i': 8, 'j': 9,
           'k': 10, 'l': 11, 'm': 12, 'n': 13, 'o': 14,
           'p': 15, 'q': 16, 'r': 17, 's': 18, 't': 19,
           'u': 20, 'v': 21, 'w': 22, 'x': 23, 'y': 24, 'z': 25}  # 单词映射到数值id的词典

training_set_scaled = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                       11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
                       21, 22, 23, 24, 25]
```

[3]:
```
x_train = []
y_train = []

for i in range(4, 26):
    x_train.append(training_set_scaled[i - 4:i])
    y_train.append(training_set_scaled[i])

np.random.seed(7)
np.random.shuffle(x_train)
np.random.seed(7)
np.random.shuffle(y_train)
tf.random.set_seed(7)

# 使x_train符合Embedding输入要求：[送入样本数， 循环核时间展开步数] ，
# 此处整个数据集送入所以送入，送入样本数为len(x_train)；输入4个字母出结果，循环核时间展开步数为4。
x_train = np.reshape(x_train, (len(x_train), 4))
y_train = np.array(y_train)
```

[4]:
```
model = tf.keras.Sequential([
    Embedding(26, 2),
    SimpleRNN(10),
    Dense(26, activation='softmax')
])
```

[5]:
```
model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['sparse_categorical_accuracy'])
```

[6]:
```
checkpoint_save_path = "G:/教学/生产实习/资料/checkpoint/rnn_embedding_4pre1.ckpt"

if os.path.exists(checkpoint_save_path + '.index'):
    print('-------------load the model-----------------')
    model.load_weights(checkpoint_save_path)

cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_save_path,
                                                 save_weights_only=True,
                                                 save_best_only=True,
                                                 monitor='loss')  # 由于fit没有给出测试集，不计算测试
```

[7]:
```
history = model.fit(x_train, y_train, batch_size=32, epochs=100, callbacks=[cp_callback])

model.summary()

file = open('G:/教学/生产实习/资料/weights.txt', 'w')  # 参数提取
for v in model.trainable_variables:
    file.write(str(v.name) + '\n')
    file.write(str(v.shape) + '\n')
    file.write(str(v.numpy()) + '\n')
file.close()
```

[8]:

```python
preNum = int(input("input the number of test alphabet:"))
for i in range(preNum):
    alphabet1 = input("input test alphabet:")
    alphabet = [w_to_id[a] for a in alphabet1]
    # 使alphabet符合Embedding输入要求：[送入样本数， 时间展开步数]。
    # 此处验证效果送入了1个样本，送入样本数为1；输入4个字母出结果，循环核时间展开步数为4。
    alphabet = np.reshape(alphabet, (1, 4))
    result = model.predict([alphabet])
    pred = tf.argmax(result, axis=1)
    pred = int(pred)
    tf.print(alphabet1 + '->' + input_word[pred])
```

```
input the number of test alphabet:1
input test alphabet:abcd
abcd->e
```

用RNN实现股票预测

SH600519.csv

A	B	C	D	E	F	G	H
	date	open	close	high	low	volume	code
74	2010/4/26	88.702	87.381	89.072	87.362	107036.1	600519
75	2010/4/27	87.355	84.841	87.355	84.681	58234.48	600519
76	2010/4/28	84.235	84.318	85.128	83.597	26287.43	600519
77	2010/4/29	84.592	85.671	86.315	84.592	34501.2	600519
78	2010/4/30	83.871	82.34	83.871	81.523	85566.7	600519
79	2010/5/4	81.676	82.091	82.678	80.974	23975.16	600519
80	2010/5/5	81.555	83.463	83.75	81.236	33838.78	600519
81	2010/5/6	83.597	81.267	83.597	81.236	28240.34	600519
82	2010/5/7	80.406	82.965	83.195	80.087	31254.76	600519
83	2010/5/10	84.235	85.218	86.787	84.235	66192.58	600519
84	2010/5/11	86.526	85.294	86.787	84.764	33632.41	600519

下载股票数据的代码

[1]:

```
import tushare as ts
import matplotlib.pyplot as plt

df1 = ts.get_k_data('600519', ktype='D', start='2010-04-26', end='2020-04-26')

datapath1 = "G:/教学/生产实习/资料/数据/SH600519.csv"
df1.to_csv(datapath1)
```

本接口即将停止更新，请尽快使用Pro版接口：https://tushare.pro/document/2

[1]:
```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Dropout, Dense, SimpleRNN
import matplotlib.pyplot as plt
import os
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
import math
```

[2]:
```python
maotai = pd.read_csv('G:/教学/生产实习/资料/SH600519.csv')  # 读取股票文件
# 前(2426-300=2126)天的开盘价作为训练集,表格从0开始计数, 2:3 是提取[2:3)列, 前闭后开,故提取出C列开盘价
training_set = maotai.iloc[0:2426 - 300, 2:3].values
test_set = maotai.iloc[2426 - 300:, 2:3].values  # 后300天的开盘价作为测试集
```

[3]:
```python
# 归一化
sc = MinMaxScaler(feature_range=(0, 1))  # 定义归一化: 归一化到(0, 1)之间
training_set_scaled = sc.fit_transform(training_set)  # 求得训练集的最大值, 最小值这些训练集固有的属性, 并在训练集上进行归一化
test_set = sc.transform(test_set)  # 利用训练集的属性对测试集进行归一化
```

[4]:
```python
x_train = []
y_train = []

x_test = []
y_test = []

# 测试集：csv表格中前2426-300=2126天数据
# 利用for循环，遍历整个训练集，提取训练集中连续60天的开盘价作为输入特征x_train,
#第61天的数据作为标签，for循环共构建2426-300-60=2066组数据。
for i in range(60, len(training_set_scaled)):
    x_train.append(training_set_scaled[i - 60:i, 0])
    y_train.append(training_set_scaled[i, 0])
# 对训练集进行打乱
np.random.seed(7)
np.random.shuffle(x_train)
np.random.seed(7)
np.random.shuffle(y_train)
# 将训练集由list格式变为array格式
x_train, y_train = np.array(x_train), np.array(y_train)
```

```
[5]: # 使x_train符合RNN输入要求：[送入样本数， 循环核时间展开步数， 每个时间步输入特征个数]。
# 此处整个数据集送入，送入样本数为x_train.shape[0]即2066组数据；输入60个开盘价，
# 预测出第61天的开盘价，循环核时间展开步数为60; 每个时间步送入的特征是某一天的开盘价，
# 只有1个数据，故每个时间步输入特征个数为1
x_train = np.reshape(x_train, (x_train.shape[0], 60, 1))
# 测试集：csv表格中后300天数据
# 利用for循环，遍历整个测试集，提取测试集中连续60天的开盘价作为输入特征x_train，第61天的数据作为标签，for循环共构建300-60=240组数据。
for i in range(60, len(test_set)):
    x_test.append(test_set[i - 60:i, 0])
    y_test.append(test_set[i, 0])
# 测试集变array并reshape为符合RNN输入要求：[送入样本数， 循环核时间展开步数， 每个时间步输入特征个数]
x_test, y_test = np.array(x_test), np.array(y_test)
x_test = np.reshape(x_test, (x_test.shape[0], 60, 1))
```

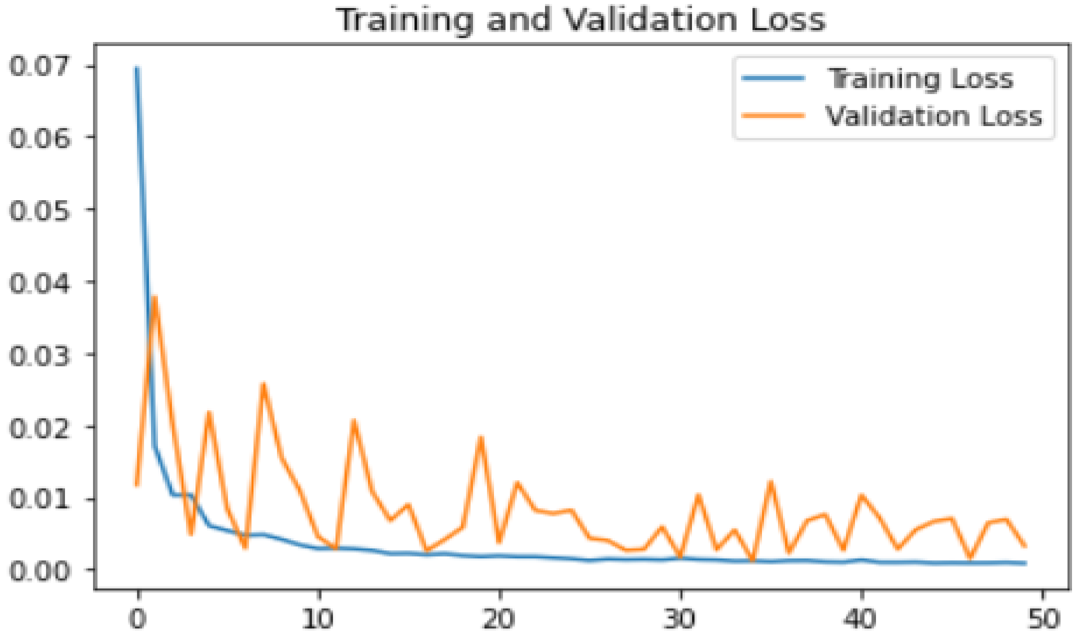
```
[6]: model = tf.keras.Sequential([
    SimpleRNN(80, return_sequences=True),
    Dropout(0.2),
    SimpleRNN(100),
    Dropout(0.2),
    Dense(1)
])

model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss='mean_squared_error')  # 损失函数用均方误差
# 该应用只观测loss数值，不观测准确率，所以删去metrics选项，一会在每个epoch迭代显示时只显示loss值
```

```
[7]: checkpoint_save_path = "G:/教学/生产实习/资料/checkpoint/rnn_stock.ckpt"

if os.path.exists(checkpoint_save_path + '.index'):
    print('-------------load the model-----------------')
    model.load_weights(checkpoint_save_path)

cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_save_path,
                                                 save_weights_only=True,
                                                 save_best_only=True,
                                                 monitor='val_loss')

history = model.fit(x_train, y_train, batch_size=64, epochs=50, validation_data=(x_test, y_test), validation_freq=1,
                    callbacks=[cp_callback])

model.summary()
```

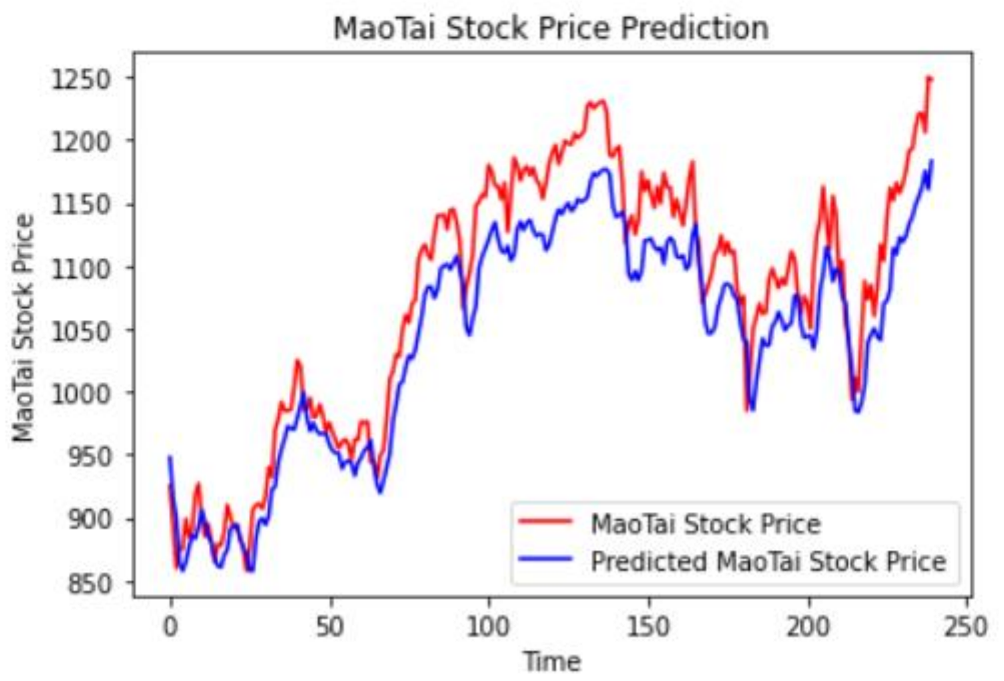
```
Train on 2066 samples, validate on 240 samples
Epoch 1/50
2066/2066 [==============================] - 2s 1ms/sample - loss: 0.0694 - val_loss: 0.0118
Epoch 2/50
2066/2066 [==============================] - 1s 596us/sample - loss: 0.0171 - val_loss: 0.0378
```

[8]:

```
file = open('G:/教学/生产实习/资料/weights.txt', 'w')  # 参数提取
for v in model.trainable_variables:
    file.write(str(v.name) + '\n')
    file.write(str(v.shape) + '\n')
    file.write(str(v.numpy()) + '\n')
file.close()

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```



```python
################### predict ######################
# 测试集输入模型进行预测
predicted_stock_price = model.predict(x_test)
# 对预测数据还原---从（0，1）反归一化到原始范围
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
# 对真实数据还原---从（0，1）反归一化到原始范围
real_stock_price = sc.inverse_transform(test_set[60:])
# 画出真实数据和预测数据的对比曲线
plt.plot(real_stock_price, color='red', label='MaoTai Stock Price')
plt.plot(predicted_stock_price, color='blue', label='Predicted MaoTai Stock Price')
plt.title('MaoTai Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('MaoTai Stock Price')
plt.legend()
plt.show()
```



[10]:

```python
##########evaluate##############
# calculate MSE 均方误差 ---> E[(预测值-真实值)^2] (预测值减真实值求平方后求均值)
mse = mean_squared_error(predicted_stock_price, real_stock_price)
# calculate RMSE 均方根误差--->sqrt[MSE]    (对均方误差开方)
rmse = math.sqrt(mean_squared_error(predicted_stock_price, real_stock_price))
# calculate MAE 平均绝对误差----->E[|预测值-真实值|](预测值减真实值求绝对值后求均值)
mae = mean_absolute_error(predicted_stock_price, real_stock_price)
print('均方误差: %.6f' % mse)
print('均方根误差: %.6f' % rmse)
print('平均绝对误差: %.6f' % mae)
```

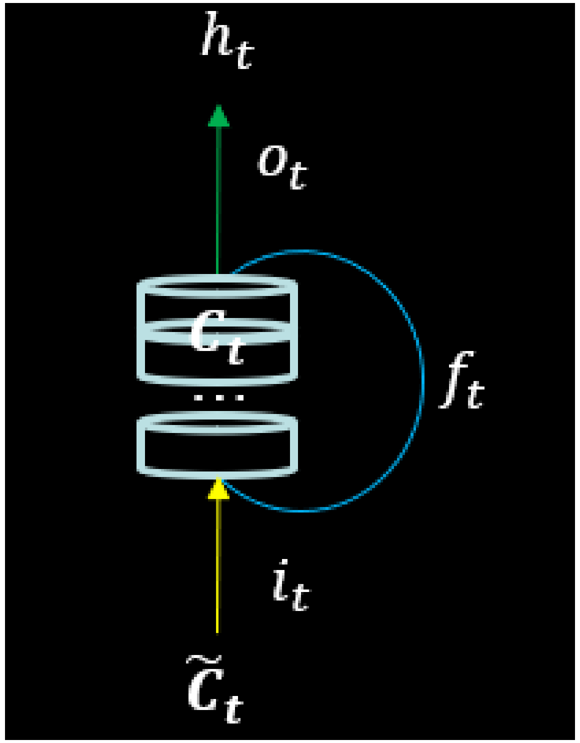
```
均方误差: 1652.593133
均方根误差: 40.652099
平均绝对误差: 35.488483
```

用长短记忆网络（LSTM）实现股票预测

RNN 面临的较大问题是无法解决长跨度依赖问题，即后面节点相对于跨度很大 的 前 面 时 间 节 点 的 信 息 感 知 能 力 太 弱 。

LSTM(LONG SHORT-TERM MEMORY) 是1997年提出的，通过门控单元机制对信息的流通和损失进行控制，改善了RNN长期依赖问题。

LSTM的计算过程



输入门（门限）: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

遗忘门（门限）: $f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$

输出门（门限）: $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$

细胞态（长期记忆）: $C_t = \underbrace{f_t * C_{t-1}}_{\text{过去}} + \underbrace{i_t * \widetilde{C}_t}_{\text{现在}}$

记忆体（短期记忆）: $h_t = o_t * \tanh(C_t)$

候选态（归纳出的新知识）: $\widetilde{C}_t = tanh(W_c \cdot [\underbrace{h_{t-1}}_{\text{上一页 ppt}}, \underbrace{x_t}_{\text{当前页 ppt}}] + b_c)$

TF描述LSTM层

tf.keras.layers.LSTM(记忆体个数， return_sequences=是否返回输出)

#return_sequences=True 各时间步输出ht

return_sequences=False 仅最后时间步输出ht（默任）

例：

```
model = tf.keras.Sequential([
    LSTM(80, return_sequences=True),
    Dropout(0.2),
    LSTM(100),
    Dropout(0.2),
    Dense(1)
])
```

[1]:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Dropout, Dense, LSTM
import matplotlib.pyplot as plt
import os
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
import math
```

[2]:

```
maotai = pd.read_csv('G:/教学/生产实习/资料/SH600519.csv')  # 读取股票文件
# 前(2426-300=2126)天的开盘价作为训练集,表格从0开始计数, 2:3 是提取[2:3)列, 前闭后开,故提取出C列开盘价
training_set = maotai.iloc[0:2426 - 300, 2:3].values
test_set = maotai.iloc[2426 - 300:, 2:3].values  # 后300天的开盘价作为测试集

# 归一化
sc = MinMaxScaler(feature_range=(0, 1))  # 定义归一化: 归一化到(0, 1)之间
training_set_scaled = sc.fit_transform(training_set)  # 求得训练集的最大值, 最小值这些训练集固有的属性, 并在训练集上
test_set = sc.transform(test_set)  # 利用训练集的属性对测试集进行归一化
```

[3]:

```python
x_train = []
y_train = []

x_test = []
y_test = []

# 测试集：csv表格中前2426-300=2126天数据
# 利用for循环，遍历整个训练集，提取训练集中连续60天的开盘价作为输入特征x_train,
#第61天的数据作为标签，for循环共构建2426-300-60=2066组数据。
for i in range(60, len(training_set_scaled)):
    x_train.append(training_set_scaled[i - 60:i, 0])
    y_train.append(training_set_scaled[i, 0])
# 对训练集进行打乱
np.random.seed(7)
np.random.shuffle(x_train)
np.random.seed(7)
np.random.shuffle(y_train)
tf.random.set_seed(7)
# 将训练集由list格式变为array格式
x_train, y_train = np.array(x_train), np.array(y_train)
```

[4]:
```
# 使x_train符合RNN输入要求：[送入样本数， 循环核时间展开步数， 每个时间步输入特征个数]。
# 此处整个数据集送入，送入样本数为x_train.shape[0]即2066组数据；输入60个开盘价，预测出第61天的开盘价，
#循环核时间展开步数为60; 每个时间步送入的特征是某一天的开盘价，只有1个数据，故每个时间步输入特征个数为1
x_train = np.reshape(x_train, (x_train.shape[0], 60, 1))
# 测试集：csv表格中后300天数据
# 利用for循环，遍历整个测试集，提取测试集中连续60天的开盘价作为输入特征x_train，第61天的数据作为标签，
#for循环共构建300-60=240组数据。
for i in range(60, len(test_set)):
    x_test.append(test_set[i - 60:i, 0])
    y_test.append(test_set[i, 0])
# 测试集变array并reshape为符合RNN输入要求：[送入样本数， 循环核时间展开步数， 每个时间步输入特征个数]
x_test, y_test = np.array(x_test), np.array(y_test)
x_test = np.reshape(x_test, (x_test.shape[0], 60, 1))
```

[5]:
```
model = tf.keras.Sequential([
    LSTM(80, return_sequences=True),
    Dropout(0.2),
    LSTM(100),
    Dropout(0.2),
    Dense(1)
])
```

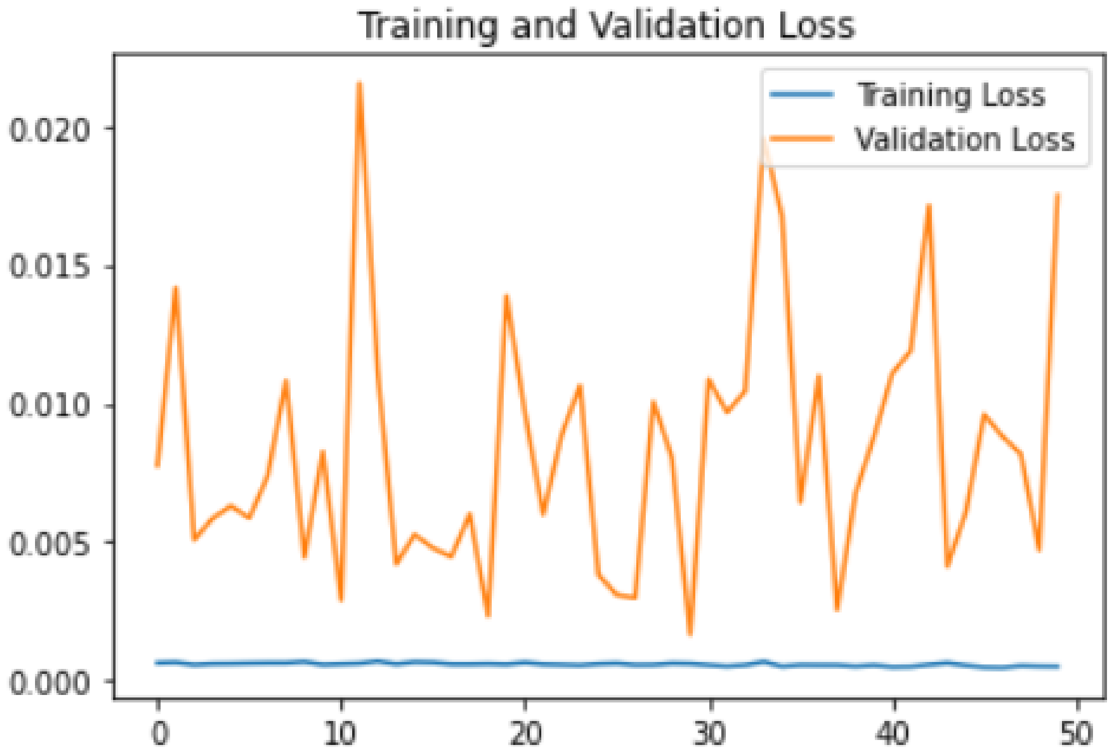
[6]:
```
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss='mean_squared_error')  # 损失函数用均方误差
# 该应用只观测loss数值，不观测准确率，所以删去metrics选项，一会在每个epoch迭代显示时只显示loss值
```

[7]:
```
checkpoint_save_path = "G:/教学/生产实习/资料/checkpoint/LSTM_stock.ckpt"

if os.path.exists(checkpoint_save_path + '.index'):
    print('-------------load the model-----------------')
    model.load_weights(checkpoint_save_path)

cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_save_path,
                                                 save_weights_only=True,
                                                 save_best_only=True,
                                                 monitor='val_loss')
```

```
[8]: history = model.fit(x_train, y_train, batch_size=64, epochs=50, validation_data=(x_test, y_test), validation_freq=1,
                    callbacks=[cp_callback])


model.summary()


file = open('G:/教学/生产实习/资料/weights.txt', 'w')  # 参数提取
for v in model.trainable_variables:
    file.write(str(v.name) + '\n')
    file.write(str(v.shape) + '\n')
    file.write(str(v.numpy()) + '\n')
file.close()
```

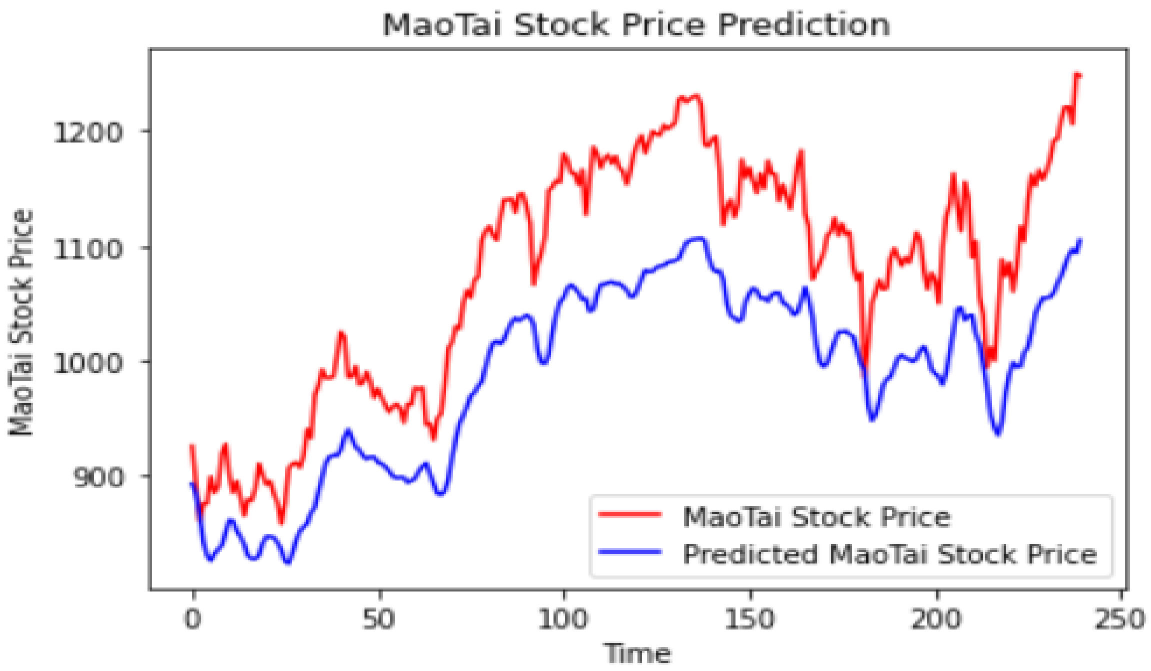
```
2066/2066 [==============================] - 4s 2ms/sample - loss: 6.1338e-04 - val_loss: 0.0023
Epoch 20/50
2066/2066 [==============================] - 3s 1ms/sample - loss: 5.9359e-04 - val_loss: 0.0139
```

[9]:
```
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```



[10]:

```
#################### predict ########################
# 测试集输入模型进行预测
predicted_stock_price = model.predict(x_test)
# 对预测数据还原---从（0，1）反归一化到原始范围
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
# 对真实数据还原---从（0，1）反归一化到原始范围
real_stock_price = sc.inverse_transform(test_set[60:])
# 画出真实数据和预测数据的对比曲线
plt.plot(real_stock_price, color='red', label='MaoTai Stock Price')
plt.plot(predicted_stock_price, color='blue', label='Predicted MaoTai Stock Price')
plt.title('MaoTai Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('MaoTai Stock Price')
plt.legend()
plt.show()
```



[11]:

```python
##########evaluate##############
# calculate MSE 均方误差 ---> E[(预测值-真实值)^2] (预测值减真实值求平方后求均值)
mse = mean_squared_error(predicted_stock_price, real_stock_price)
# calculate RMSE 均方根误差--->sqrt[MSE]    (对均方误差开方)
rmse = math.sqrt(mean_squared_error(predicted_stock_price, real_stock_price))
# calculate MAE 平均绝对误差----->E[|预测值-真实值|](预测值减真实值求绝对值后求均值)
mae = mean_absolute_error(predicted_stock_price, real_stock_price)
print('均方误差: %.6f' % mse)
print('均方根误差: %.6f' % rmse)
print('平均绝对误差: %.6f' % mae)
```

```
均方误差: 8779.581307
均方根误差: 93.699420
平均绝对误差: 88.640324
```