# 人工智能实践：TensorFlow 笔记

## 第六讲

## 循环神经网络

**本节目标：学习循环神经网络，用 RNN、LSTM、GRU 实现连续数据的预测（以股票预测为例）。**

### 一，循环神经网络（Recurrent Neural Network，RNN）

1，卷积神经网络与循环神经网络简单对比

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。

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。

2，详解 RNN

**• 循环核**

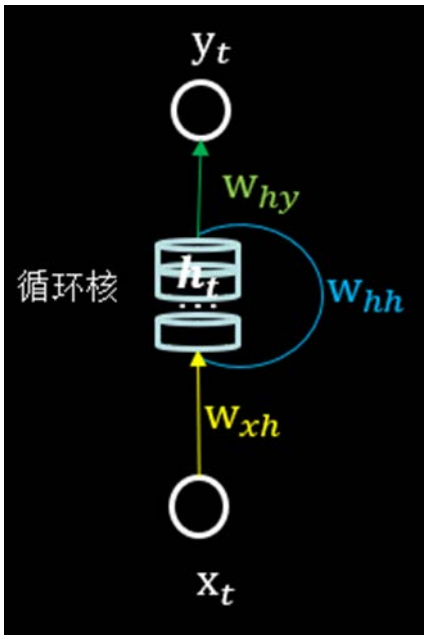


图 1.2.1 RNN 循环核

循环核具有记忆力，通过不同时刻的参数共享，实现了对时间序列的信息提取。每个循环核有多个记忆体，对应图 1.2.1 中的多个小圆柱。记忆体内存储着

每个时刻的状态信息$h_t$，这里$h_t = \tanh(x_t w_{xh} + h_{t-1} w_{hh} + bh)$。其中，$w_{xh}$、$w_{hh}$为权重矩阵，$bh$为偏置，$x_t$为当前时刻的输入特征，$h_{t-1}$为记忆体上一时刻存储的状态信息，tanh 为激活函数。

yt是整个循环网络的末层

当前时刻循环核的输出特征$y_t = \text{softmax}(h_t w_{hy} + by)$，其中$w_{hy}$为权重矩阵、$by$为偏置、softmax 为激活函数，其实就相当于一层全连接层。我们可以设定记忆体的个数从而改变记忆容量，当记忆体个数被指定、输入$x_t$输出$y_t$维度被指定，周围这些待训练参数的维度也就被限定了。在前向传播时，记忆体内存储的状态信息$h_t$在每个时刻都被刷新，而三个参数矩阵$w_{xh}$、$w_{hh}$、$w_{hy}$和两个偏置项$bh$、$by$自始至终都是固定不变的。在反向传播时，三个参数矩阵和两个偏置项由梯度下降法更新。

**• 循环核按时间步展开**

将循环核按时间步展开，就是把循环核按照时间轴方向展开，可以得到如图 1.2.2 的形式。每个时刻记忆体状态信息$h_t$被刷新，记忆体周围的参数矩阵和两个偏置项是固定不变的，我们训练优化的就是这些参数矩阵。训练完成后，使用效果最好的参数矩阵执行前向传播，然后输出预测结果。其实这和我们人类的预测是一致的：我们脑中的记忆体每个时刻都根据当前的输入而更新；当前的预测推理是根据我们以往的知识积累用固化下来的“参数矩阵”进行的推理判断。

Wxh、Whh、Why

bh、by

可以看出，循环神经网络就是借助循环核实现时间特征提取后把提取到的信息送入全连接网络，从而实现连续数据的预测。

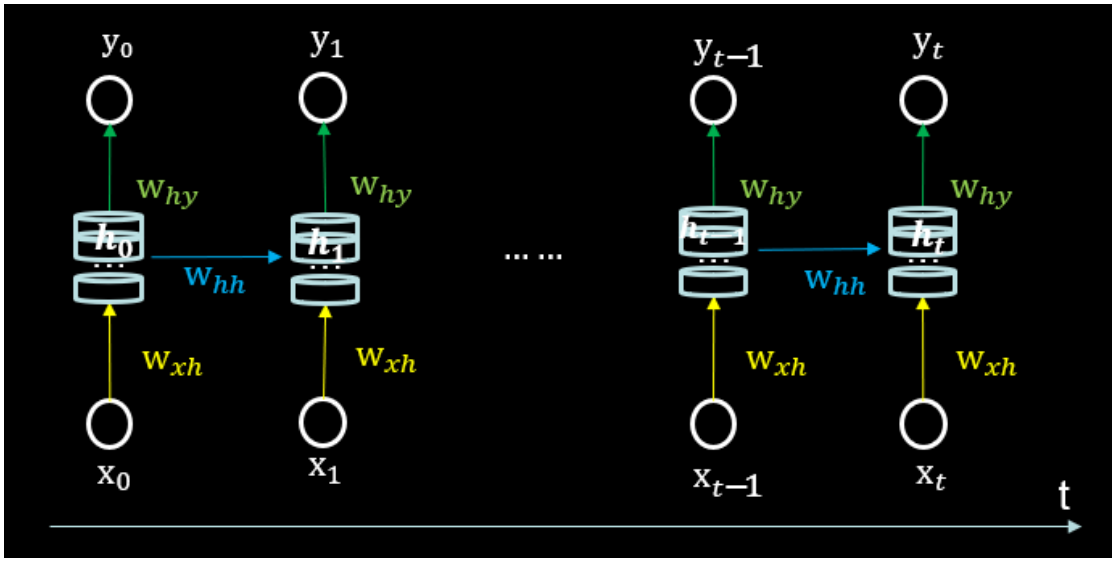


图 1.2.2 RNN 循环核按时间步展开

• **循环计算层：向输出方向生长**

在 RNN 中，每个循环核构成一层循环计算层，循环计算层的层数是向输出方向增长的。如图 1.2.3 所示，左图的网络有一个循环核，构成了一层循环计算层；中图的网络有两个循环核，构成了两层循环计算层；右图的网络有三个循环核，构成了三层循环计算层。其中，三个网络中每个循环核中记忆体的个数可以根据我们的需求任意指定。

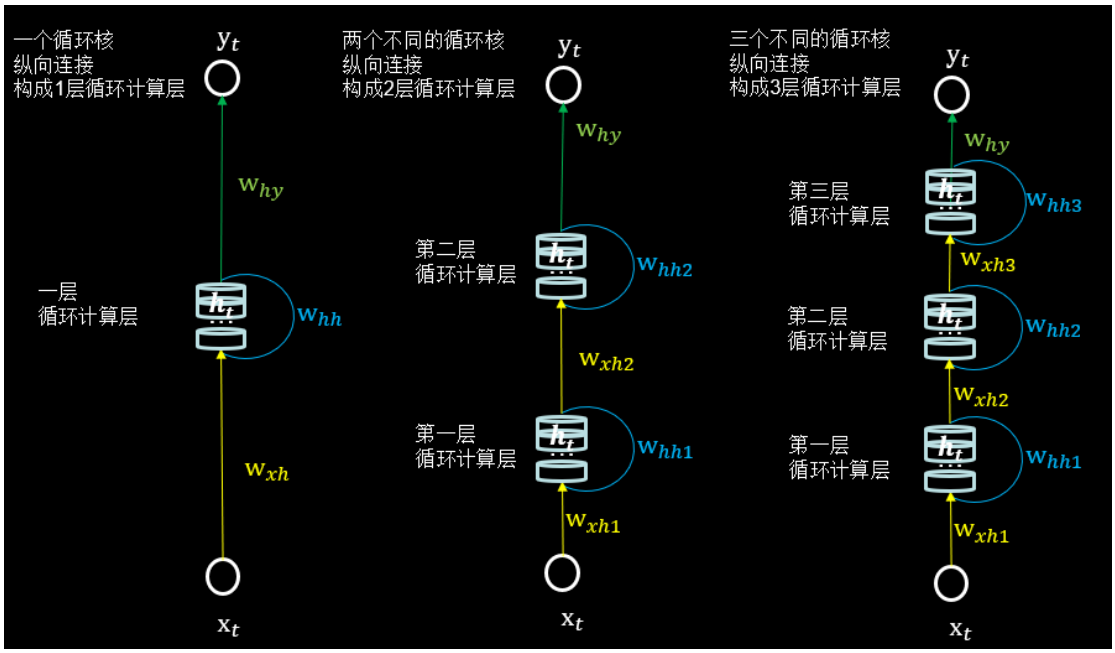


图 1.2.3 循环计算层

• **RNN 训练**

得到 RNN 的前向传播结果之后，和其他神经网络类似，我们会定义损失函数，使用反向传播梯度下降算法训练模型。RNN 唯一的区别在于：由于它每个时刻的节点都可能有一个输出，所以 RNN 的总损失为所有时刻（或部分时刻）上的损失和。

• **Tensorflow2 描述循环计算层**

记忆体个数　　使用什么激活函数计算ht，若不写，默认tanh

tf.keras.layers.SimpleRNN(神经元个数，activation=‘激活函数’，return_sequences=是否每个时刻输出$h_t$到下一层)

(1)神经元个数：即循环核中记忆体的个数

(2) return_sequences：在输出序列中，返回最后时间步的输出值$h_t$还是返回全部时间步的输出。False 返回最后时刻(图 1.2.5)，True 返回全部时刻(图 1.2.4)。当下一层依然是 RNN 层，通常为 True，反之如果后面是 Dense 层，通

一般，最后一层的循环核用False，仅在最后一个时间步输出ht；
中间层的循环核用True，每个时间步都把ht输出给下一层
默认为False

常为 Fasle。

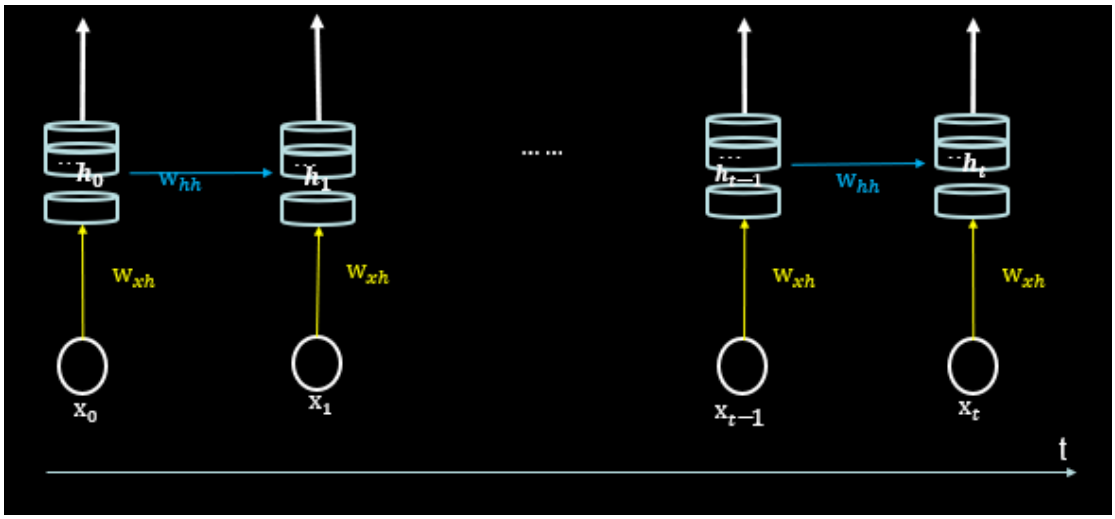


图 1.2.4 return_sequences = True 循环核在每个时间步输出ht

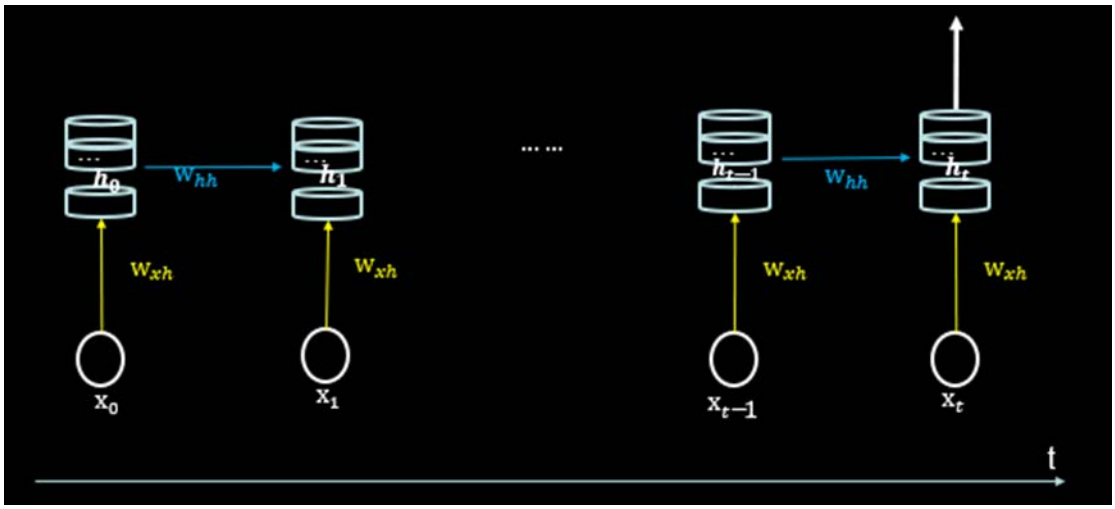


图 1.2.5 return_sequences = False 循环核仅在最后一个时间步输出ht

API对送入循环层的数据维度是有要求的

(3)输入维度：三维张量(输入样本数，循环核时间展开步数， 每个时间步输入特征个数)。

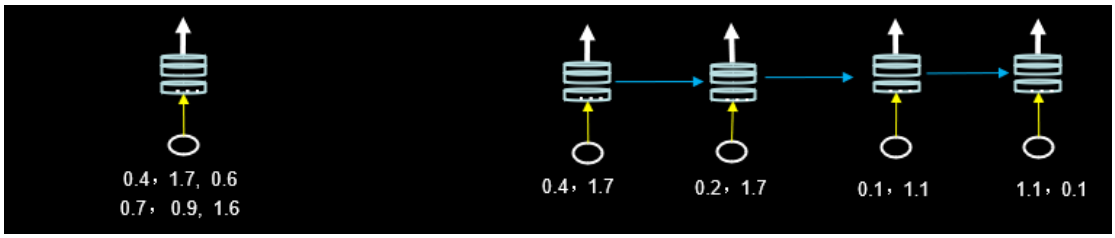


图 1.2.6 RNN 层输入维度

如图 1.2.6 所示，左图一共要送入 RNN 层两组数据，每组数据经过一个时间步就会得到输出结果，每个时间步送入三个数值，则输入循环层的数据维度就是

[2, 1, 3]；右图输入只有一组数据，分四个时间步送入循环层，每个时间步送入两个数值 ，则输入循环层的数据维度就是 [1，4， 2]。

(4)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)

(5) activation：‘激活函数’(不写默认使用 tanh)

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$h_t$。

**• 循环计算过程之 lpre1**

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。举一个简单的字母预测例子体会一下循环网络的计算过程：输入一个字母预测下一个字母---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。计算机不认识字母，只能处理数字。所以需要我们对字母进行编码。这里假设使用独热编码（实际中可使用其他编码方式），编码结果如图 1.2.7 所示。

| | |
|---|---|
| 10000 | a |
| 01000 | b |
| 00100 | c |
| 00010 | d |
| 00001 | e |

图 1.2.7 字母独热编码

假设使用一层 RNN 网络，记忆体的个数选取 3，则字母预测的网络如图 1.2.8 所示。假设输入字母 b，即输入$x_t$为[0,1,0,0,0]，这时上一时刻的记忆体状态信息$h_{t-1}$为 0。由上文理论知识不难得到：$h_t = \tanh(x_t w_{xh} + h_{t-1} w_{hh} + b_h) =$ $\tanh([-2.3 \quad 0.8 \quad 1.1] + 0 + [0.5 \quad 0.3 \quad -0.2]) =$ $\tanh[-1.8 \quad 1.1 \quad 0.9] = [-0.9 \quad 0.8 \quad 0.7]$，这个过程可以理解为脑中的记忆因为当前输入的事物而更新了。

输出$y_t$是把提取到的时间信息通过全连接进行识别预测的过程，是整个网络的输出层。不难知道，$y_t = \text{softmax}(h_t w_{hy} + by) = \text{softmax}([-0.7 \quad -0.6 \quad 2.9 \quad 0.7 \quad -0.8] + [0.0 \quad 0.1 \quad 0.4 \quad -0.7 \quad 0.1]) = \text{softmax}([-0.7 \quad -0.5 \quad 3.3 \quad 0.0 \quad -0.7]) =$

当前时刻的记忆体状态信息

随机生成了Wxh、Whh和Why三个参数矩阵

[0.02 0.02 **0.91** 0.03 0.02 ] 。可见模型认为有 91%的可能性输出字母 c ，所以循环网络输出了预测结果 c。

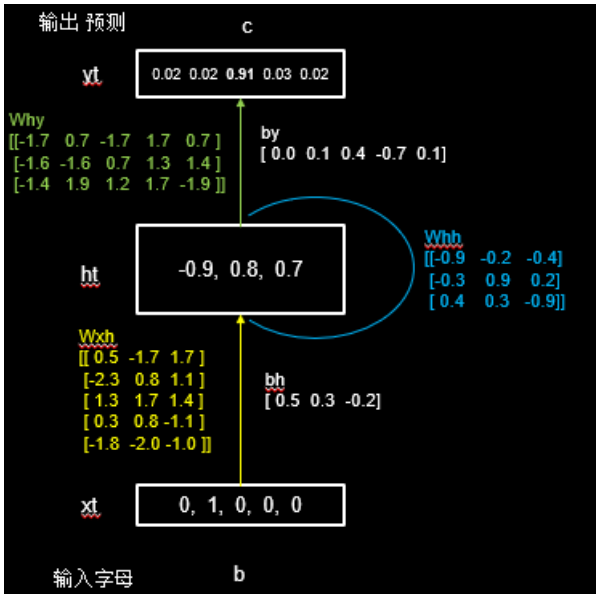


图 1.2.8 字母预测网络 1pre1

**• 实践：字母预测 1pre1**

(1)用独热编码的方式实现：

按照六步法八股套路进行编码：

源码：p15_rnn_onehot_1pre1.py

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

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

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']]

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所以送入，送入样本数为len(x_train)；输入1个字母出结果，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
x_train = np.reshape(x_train, (len(x_train), 1, 5))
y_train = np.array(y_train)
```

图 1.2.9 p15_rnn_onehot_1pre1.py(1)

如图 1.2.9 所示，import 相关模块→生成训练用的输入特征 x_train 和标签 y_train(输入特征 a 对应的标签是 b、输入特征 b 对应的标签是 c、依次类

推)，打乱顺序后变形成 RNN 输入需要的维度。

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

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

checkpoint_save_path = "./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

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

summary
model.summary()
```

$$y_t = \text{softmax}(h_t w_{hy} + by)$$

图 1.2.10 p15_rnn_onehot_1pre1.py(2)

如图 1.2.10 所示，构建模型:一个具有 3 个记忆体的循环层+一层全连接→Compile→fit→summary。

```
参数提取
file = open('./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()

acc/loss可视化
############################################    show    ############################################

# 显示训练集和验证集的acc和loss曲线
acc = history.history['sparse_categorical_accuracy']
loss = history.history['loss']

plt.subplot(1, 2, 1)
plt.plot(acc, label='Training Accuracy')
plt.title('Training Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(loss, label='Training Loss')
plt.title('Training Loss')
plt.legend()
plt.show()
```

图 1.2.11 p15_rnn_onehot_1pre1.py(3)

如图 1.2.11 所示，提取参数和 acc、loss 可视化。

```
############## predict #############

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])
```

图 1.2.12 p15_rnn_onehot_1pre1.py(4)

如图 1.2.12 所示为展示预测效果的应用程序,将其写到了这段代码的最后：首先输入要执行几次预测任务;随后等待输入一个字母，将这个字母转换为独热码形式后调整为 RNN 层希望的形状;然后通过 predict 得到预测结果，选出预测结果中最大的一个即为预测结果。运行结果如图 1.2.13 所示。

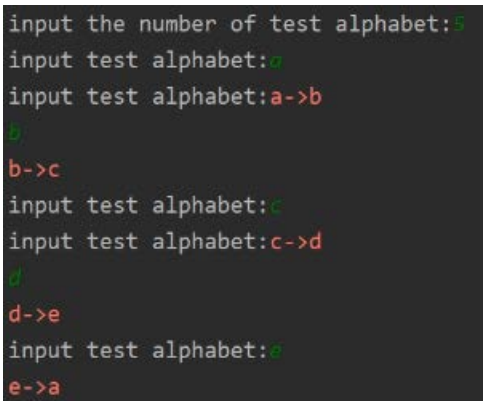


图 1.2.13 p15_rnn_onehot_1pre1.py 预测结果

独热码的位宽要与词汇量一致
若词汇量增大时，(2)用 Embedding 编码的方式实现:
非常浪费资源

*为什么使用 Embedding?

独热码：数据量大、过于稀疏，映射之间是独立的，没有表现出关联性。

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。

*Tensorflow2 中的词向量空间编码层：

tf.keras.layers.Embedding(词汇表大小，编码维度)

词汇表大小：编码一共要表示多少个单词；

编码维度：用几个数字表达一个单词；

输入维度：二维张量[送入样本数，循环核时间展开步数]

输出维度：三维张量[送入样本数，循环核时间展开步数，编码维度]

例 ：tf.keras.layers.Embedding(100, 3)。对数字 1-100 进行编码，词汇表大小就是 100 ;每个自然数用三个数字表示,编码维度就是 3；所以 Embedding 层的参数是 100 和 3。比如数字[4] embedding 为 [0.25, 0.1, 0.11]。

*Embedding 实现 1pre1:

如图 1.2.14 所示，浅蓝色框框住的区域为与独热编码不同的地方。不同是因为需要把输入特征变成 Embedding 层期待的形状：第一个维度是送入样本数、第二个维度是循环核时间展开步数。

源码：p27_rnn_embedding_1pre1.py

```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Dense, SimpleRNN, Embedding
import matplotlib.pyplot as plt
import os

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

x_train = [w_to_id['a'], w_to_id['b'], w_to_id['c'], w_to_id['d'], 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']]

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1个字母出结果，循环核时间展开步数为1。
x_train = np.reshape(x_train, (len(x_train), 1))
y_train = np.array(y_train)
```

图 1.2.14 p27_rnn_embedding_1pre1.py(1)

如图 1.2.15 所示，在模型部分相比于独热编码形式多了一个 Embedding 层对输入数据进行编码，这一层会生成一个五行两列的可训练参数矩阵，实现编码可训练。

```python
model = tf.keras.Sequential([
    Embedding(5, 2),
    SimpleRNN(3),
    Dense(5, activation='softmax')
])

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

checkpoint_save_path = "./checkpoint/run_embedding_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()
```

图 1.2.15 p27_rnn_embedding_1pre1.py(2)

参数提取和 acc/loss 可视化和 p15_rnn_onehot_1pre1.py 代码完全一样。在结果预测时，如图 1.2.16 所示，只需要将读到的输入字母直接查找表示它的 ID 值，然后调整为 Embedding 层希望的形状输入网络进行预测即可。

```
74  ############## predict ############
75
76  preNum = int(input("input the number of test alphabet:"))
77  for i in range(preNum):
78      alphabet1 = input("input test alphabet:")
79      alphabet = [w_to_id[alphabet1]]
80      # 使alphabet符合Embedding输入要求：[送入样本数， 循环核时间展开步数]。
81      # 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，循环核时间展开步数为1。
82      alphabet = np.reshape(alphabet, (1, 1))
83      result = model.predict(alphabet)
84      pred = tf.argmax(result, axis=1)
85      pred = int(pred)
86      tf.print(alphabet1 + '->' + input_word[pred])
```

图 1.2.16 p27_rnn_embedding_1pre1.py(3)

• 循环计算过程之 4pre1

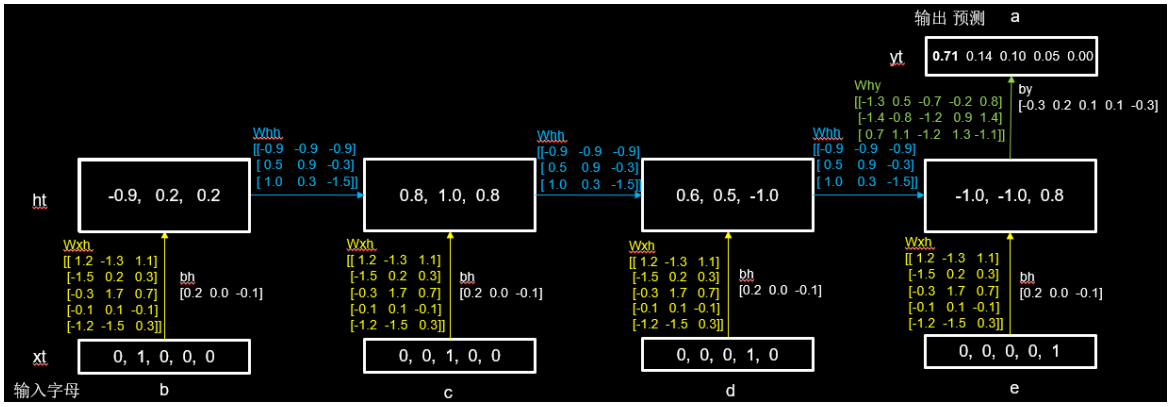


图 1.2.17 字母预测网络 4pre1

把时间核按时间步展开

1pre1 是输入一个字母预测下一个字母的例子，4pre1 是连续输入多(这里取 4)个字母预测下一个字母的例子。这里仍然使用三个记忆体，初始时刻记忆体内的记忆是 0。接下来用一套训练好的参数矩阵感受循环计算的前向传播过程，在这个过程中的每个时刻参数矩阵是固定的，记忆体会在每个时刻被更新。下面以输入 bcde 预测 a 为例：

在第一个时刻，b 的独热码[0, 1, 0, 0, 0]输入，记忆体根据更新公式$h_t = \tanh(x_t w_{xh} + h_{t-1} w_{hh} + bh)$ = tanh([−1.5 0.2 0.3] + [0.0 0.0 0.0] + [0.2 0.0 −0.1]) = tanh([−1.3 0.2 0.2 ]) = [−0.9 0.2 0.2]刷新。

这四个时间步所用到的参数矩阵Wxh和偏置项bh的数值是相同的

在第二个时刻，c 的独热码[0, 0, 1, 0, 0]输入，记忆体根据更新公式$h_t = \tanh(x_t w_{xh} + h_{t-1} w_{hh} + bh)$ = tanh([−0.3 1.7 0.7] + [1.1 1.1 0.5] + [0.2 0.0 −0.1]) = tanh([ 1.0 2.8 1.1]) = [0.8 1.0 0.8]刷新。

四个时间步用到的三个Whh矩阵也是一样的

这里的ht-1就是第一个时刻算出来的值

在第三个时刻，d 的独热码[0, 0, 0, 1, 0]输入，记忆体根据更新公式$h_t = \tanh(x_t w_{xh} + h_{t-1} w_{hh} + bh)$= tanh([-0.1 0.1 -0.1] + [0.6 0.4 -2.2] + [0.2 0.0 -0.1])= tanh([ 0.7

0.5　-2.4 ] = [0.6　0.5　-1.0]刷新。

在第四个时刻，e 的独热码[0, 0, 0, 0, 1]输入，记忆体根据更新公式$h_t = \tanh(x_t w_{xh} + h_{t-1} w_{hh} + bh)$= tanh([-1.2　-1.5　0.3] + [-1.3　-0.4　0.8] + [0.2 0.0 -0.1])= tanh([-2.3　-1.9　1.0] = [-1.0　-1.0　0.8]刷新。

输出预测通过全连接完成，由下式求得最终输出：ht：[1 3]
Why：[3 5]

$$y_t = \text{softmax}(h_t w_{hy} + by)$$

$$= softmax([3.3 \quad 1.2 \quad 0.9 \quad 0.3 \quad -3.1]$$

$$+ [-0.3 \quad 0.2 \quad 0.1 \quad 0.1 \quad -0.3])$$

$$= softmax([3.0 \quad 1.4 \quad 1.0 \quad 0.4 \quad -3.4])$$

$$= [\mathbf{0.71} \quad 0.14 \quad 0.10 \quad 0.05 \quad 0.00]$$

说明有 71%的可能是字母 a。观察输出结果，模型不仅成功预测出了下一个字母是 a，还可以从神经网络输出的概率发现：因为输入序列的最后一个字母是 e，所以模型理应也确实认为下一个字母还是 e 的可能性最小，可能性最大的是 a，其次分别是 b、c、d 。

**• 实践：字母预测 4pre1**

(1)用独热编码的方式实现：

源码：p21_rnn_onehot_4pre1.py

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

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

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']]

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

图 1.2.18 p21_rnn_onehot_4pre1.py(1)

如图 1.2.18 所示，浅蓝色框框住的区域为与 p15_rnn_onehot_1pre1.py 不同的地方，即 x_train、y_train 变成了四个字母预测一个字母的形式(输入连续的 abcd 对应的标签是 e、输入连续的 bcde 对应的标签是 a、依此类推)。

源码：p21_rnn_onehot_4pre1.py

```
26
27  # 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]。
28  # 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；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
29  x_train = np.reshape(x_train, (len(x_train), 4, 5))
30  y_train = np.array(y_train)
31
32  model = tf.keras.Sequential([
33      SimpleRNN(3),
34      Dense(5, activation='softmax')
35  ])
36
37  model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
38                loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
39                metrics=['sparse_categorical_accuracy'])
40
41  checkpoint_save_path = "./checkpoint/rnn_onehot_4pre1.ckpt"
42
43  if os.path.exists(checkpoint_save_path + '.index'):
44      print('-------------load the model-----------------')
45      model.load_weights(checkpoint_save_path)
46
47  cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_save_path,
48                                                   save_weights_only=True,
49                                                   save_best_only=True,
50                                                   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
51
52  history = model.fit(x_train, y_train, batch_size=32, epochs=100, callbacks=[cp_callback])
53
54  model.summary()
```

sequential　compile　fit　断点续训　summary

图 1.2.19 p21_rnn_onehot_4pre1.py(2)

如图 1.2.19 所示，与 p15_rnn_onehot_1pre1.py 不同，这里的循环核展开步数为 4。

```
81  ############## predict #############
82
83  preNum = int(input("input the number of test alphabet:"))
84  for i in range(preNum):
85      alphabet1 = input("input test alphabet:")
86      alphabet = [id_to_onehot[w_to_id[a]] for a in alphabet1]
87      # 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]。
88      # 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
89      alphabet = np.reshape(alphabet, (1, 4, 5))
90      result = model.predict([alphabet])
91      pred = tf.argmax(result, axis=1)
92      pred = int(pred)
93      tf.print(alphabet1 + '->' + input_word[pred])
```

图 1.2.20 p21_rnn_onehot_4pre1.py(3)

参数提取和 acc/loss 可视化和 p15_rnn_onehot_1pre1.py 代码完全一样。在结果预测时，如图 1.2.20 所示，需要等待连续输入四个字母，然后把这四个字母转换为独热码，然后调整为 RNN 层希望的形状输入网络进行预测即可。运行结果如图 1.2.21 所示。

```
input the number of test alphabet:5
input test alphabet:abcd
input test alphabet:abcd->e
bcde
bcde->a
input test alphabet:cdea
input test alphabet:cdea->b
deab
deab->c
input test alphabet:eabc
eabc->d
```

图 1.2.21 p21_rnn_onehot_4pre1.py 预测结果

(2)用 Embedding 编码的方式实现:

这次将词汇量扩充到 26 个(即字母从 a 到 z)。如图 1.2.22 所示，首先建立一个映射表，把字母用数字表示为 0 到 25；然后建立两个空列表，一个用于存放训练用的输入特征 x_train，另一个用于存放训练用的标签 y_train；接下来用 for 循环从数字列表中把连续 4 个数作为输入特征添加到 x_train 中，第 5 个数作为标签添加到 y_train 中，这就构建了训练用的输入特征 x_train 和标签 y_train。

源码：p32_rnn_embedding_4pre1.py

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

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]

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)
```

图 1.2.22 p31_rnn_embedding_4pre1.py(1)

如图 1.2.23，把输入特征变成 Embedding 层期待的形状才能输入网络；在 sequntial 搭建网络时，相比于 one_hot 形式增加了一层 Embedding 层，先对输入数据进行编码，这里的 26 表示词汇量是 26，这里的 2 表示每个单词用 2 个数值编码，这一层会生成一个 26 行 2 列的可训练参数矩阵，实现编码可训练。随后设定具有十个记忆体的循环层和一个全连接层（输出会是 26 个字母之一，所以这里是 26)；后边进行 compile、fit、summary、参数提取和 acc/loss 可视化和 p21_rnn_onehot_4pre1.py 代码完全一样。


图 1.2.23 p31_rnn_embedding_4pre1.py(2)

在验证环节，如图 1.2.24 所示，同样使用了 for 循环先输入要执行几次检测，随后等待连续输入四个字母，待输入结束后把它们转换为 Embedding 层希望的形状，然后输入网络进行预测，选出预测结果最大的一个。运行结果如图 1.2.25 所示。

```
################## predict ##################

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])
```

图 1.2.24 p31_rnn_embedding_4pre1.py(3)

```
input the number of test alphabet:3
input test alphabet:abcd
input test alphabet:abcd->e
bcde
bcde->f
input test alphabet:opqr
opqr->s
```

图 1.2.25 p31_rnn_embedding_4pre1.py 预测结果

• **实践：RNN 实现股票预测**

(1)数据源

SH600519.csv 是用 tushare 模块下载的 SH600519 贵州茅台的日 k 线数据，本次例子中只用它的 C 列数据(如图 1.2.26 所示)：用连续 60 天的开盘价，预测第 61 天的开盘价。这个 excel 表格是使用源码 p37_tushare.py(如图 1.2.27)直接下载的真实数据，可以在这里写出我们需要的六位股票代码，下载需要的股票历史数据。

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 |

图 1.2.26 股票数据格式

```
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 = "./SH600519.csv"
df1.to_csv(datapath1)
```

图 1.2.27 p37_tushare.py

(2)代码实现

```
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

maotai = pd.read_csv('./SH600519.csv')  # 读取股票文件

training_set = maotai.iloc[0:2426 - 300, 2:3].values  # 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
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

x_train = []
y_train = []

x_test = []
y_test = []
```

图 1.2.28 p38_rnn_stock.py(1)

如图 1.2.28 所示，按照六步法： import 相关模块→读取贵州茅台日 k 线数据到变量 maotai,把变量 maotai 中前 2126 天数据中的开盘价作为训练数据，把变量 maotai 中后 300 天数据中的开盘价作为测试数据；然后对开盘价进行归一化，使送入神经网络的数据分布在 0 到 1 之间；接下来建立空列表分别用于接收训练集输入特征、训练集标签、测试集输入特征、测试集标签；

```
# 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])
# 打乱训练集顺序
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)

# 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))
```

图 1.2.29 p38_rnn_stock.py(2)

如图 1.2.29 所示，继续构造数据。用 for 循环遍历整个训练数据，每连续 60 天数据作为输入特征 x_train，第 61 天数据作为对应的标签 y_train ，一共生成 2066 组训练数据，然后打乱训练数据的顺序并转变为 array 格式继而转变为 RNN 输入要求的维度；同理，利用 for 循环遍历整个测试数据，一共生成 240 组测试数据，测试集不需要打乱顺序，但需转变为 array 格式继而转变为 RNN 输入要求的维度。

源码：p38_rnn_stock.py

```
52
53  model = tf.keras.Sequential([                                    # sequential
54      SimpleRNN(80, return_sequences=True),
55      Dropout(0.2),
56      SimpleRNN(100),
57      Dropout(0.2),
58      Dense(1)
59  ])
60
61  model.compile(optimizer=tf.keras.optimizers.Adam(0.001),         # compile
62                loss='mean_squared_error')  # 损失函数用均方误差
63  # 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
64
65  checkpoint_save_path = "./checkpoint/stock.ckpt"
66
67  if os.path.exists(checkpoint_save_path + '.index'):
68      print('-------------load the model-----------------')
69      model.load_weights(checkpoint_save_path)
70
71  cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_save_path,
72                                                   save_weights_only=True,
73                                                   save_best_only=True,
74                                                   monitor='val_loss')
75
76  history = model.fit(x_train, y_train, batch_size=64, epochs=50, validation_data=(x_test, y_test), validation_freq=1,   # fit 断点续训
77                      callbacks=[cp_callback])
78
79  model.summary()                                                  # summary
```

图 1.2.30 p38_rnn_stock.py(3)

如图 1.2.30 所示，用 sequntial 搭建神经网络：第一层循环计算层记忆体设定 80 个，每个时间步推送$\mathbf{h}_t$给下一层，使用 0.2 的 Dropout；第二层循环计算层设定记忆体有 100 个，仅最后的时间步推送$\mathbf{h}_t$给下一层，使用 0.2 的 Dropout；由于输出值是第 61 天的开盘价只有一个数，所以全连接 Dense 是 1→compile 配置训练方法使用 adam 优化器，使用均方误差损失函数。在股票预测代码中，只需观测 loss，训练迭代打印的时候也只打印 loss，所以这里就无需给 metrics 赋值→设置断点续训，fit 执行训练过程→summary 打印出网络结构和参数统计。

```
81  file = open('./weights.txt', 'w')   # 参数提取                    # 参数提取
82  for v in model.trainable_variables:
83      file.write(str(v.name) + '\n')
84      file.write(str(v.shape) + '\n')
85      file.write(str(v.numpy()) + '\n')
86  file.close()
87
88  loss = history.history['loss']                                   # loss可视化
89  val_loss = history.history['val_loss']
90
91  plt.plot(loss, label='Training Loss')
92  plt.plot(val_loss, label='Validation Loss')
93  plt.title('Training and Validation Loss')
94  plt.legend()
95  plt.show()
```

图 1.2.31 p38_rnn_stock.py(4)

如图 1.2.31，进行 loss 可视化与参数报错操作。

源码：p38_rnn_stock.py

```
应用：股票预测
91  ################# predict ####################
92  # 测试集输入模型进行预测
93  predicted_stock_price = model.predict(x_test)
94  # 对预测数据还原---从（0，1）反归一化到原始范围
95  predicted_stock_price = sc.inverse_transform(predicted_stock_price)
96  # 对真实数据还原---从（0，1）反归一化到原始范围
97  real_stock_price = sc.inverse_transform(test_set[60:])
98  # 画出真实数据和预测数据的对比曲线
预测效果可视化
99  plt.plot(real_stock_price, color='red', label='MaoTai Stock Price')
100 plt.plot(predicted_stock_price, color='blue', label='Predicted MaoTai Stock Price')
101 plt.title('MaoTai Stock Price Prediction')
102 plt.xlabel('Time')
103 plt.ylabel('MaoTai Stock Price')
104 plt.legend()
105 plt.show()
```

图 1.2.32 p38_rnn_stock.py(5)

如图 1.2.32 所示，进行股票预测。用 predict 预测测试集数据，然后将预测值和真实值从归一化的数值变换到真实数值，最后用红色线画出真实值曲线 、用蓝色线画出预测值曲线。

```
模型预测效果量化
106
107 ##########evaluate##############
108 # 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)
109 mse = mean_squared_error(predicted_stock_price, real_stock_price)
110 # calculate RMSE 均方根误差--->sqrt[MSE]    (对均方误差开方)
111 rmse = math.sqrt(mean_squared_error(predicted_stock_price, real_stock_price))
112 # 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)
113 mae = mean_absolute_error(predicted_stock_price, real_stock_price)
114 print('均方误差: %.6f' % mse)
115 print('均方根误差: %.6f' % rmse)
116 print('平均绝对误差: %.6f' % mae)
```

图 1.2.33 p38_rnn_stock.py(6)

如图 1.2.33 所示，为了评价模型优劣，给出了三个评判指标：均方误差、均方根误差和平均绝对误差，这些误差越小说明预测的数值与真实值越接近。

图 1.2.34 为 loss 值曲线、图 1.2.35 为股票预测曲线、图 1.2.36 为三个评价指标值。

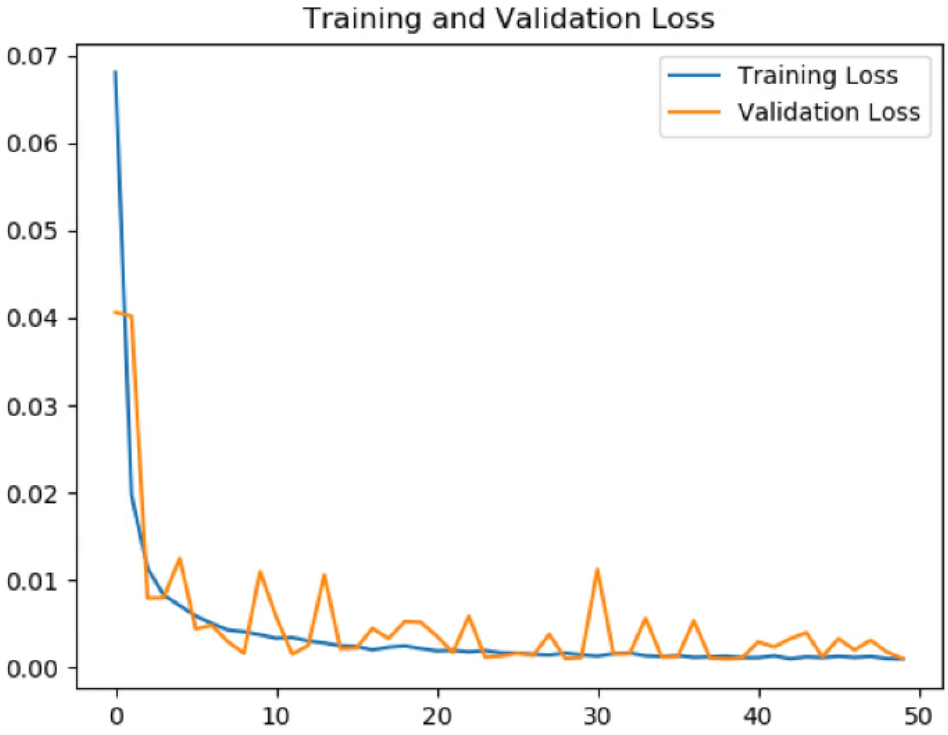


图 1.2.34 RNN 股票预测 loss 曲线

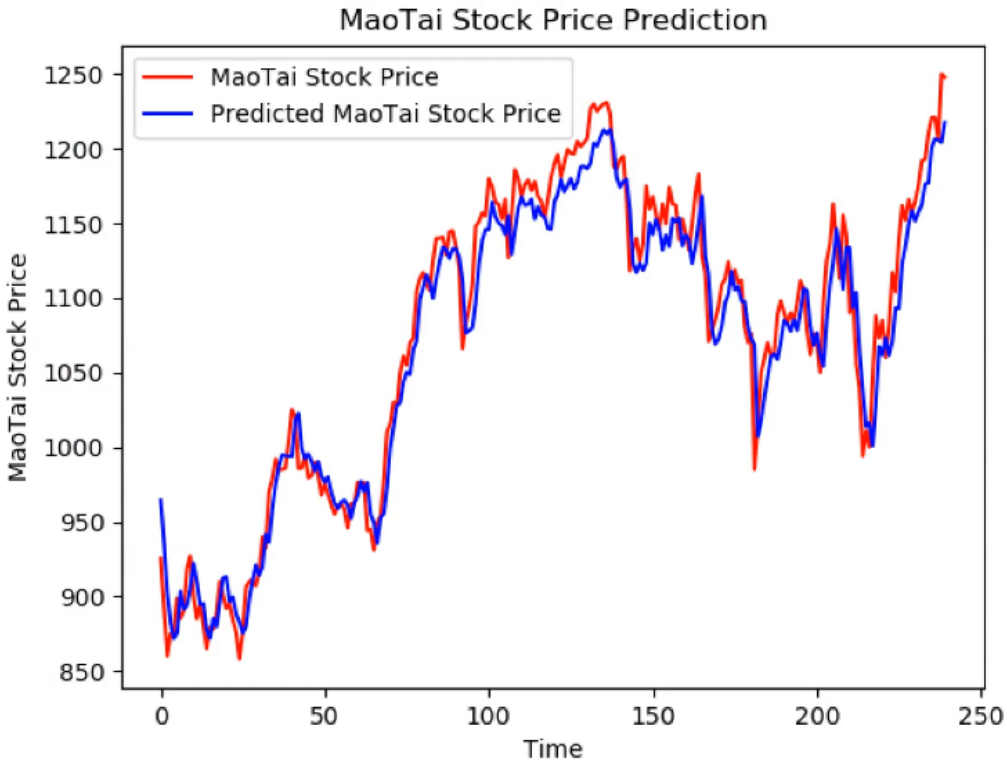


图 1.2.35 RNN 股票预测曲线

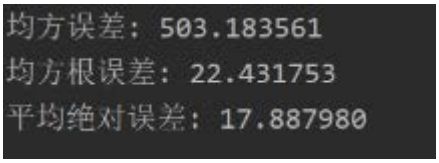


图 1.2.36 RNN 股票预测评价指标

## 二，LSTM、GRU

传统循环网络RNN可以通过记忆体实现短期记忆，实现连续数据的预测

但是当连续数据的序列变长时，会使展开时间步过长，在反向传播更新参数时，梯度要按照时间步连续相乘，会导致梯度消失


1，原始 RNN 的问题：

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。如图 2.1.1(图片来源：https://www.jianshu.com/p/9dc9f41f0b29)中的两句话：左上角的句子中 sky 可以由较短跨度的词预测出来，而右下角句子中的 French 与较长跨度之前的 France 有关系，即长跨度依赖，比较难预测。

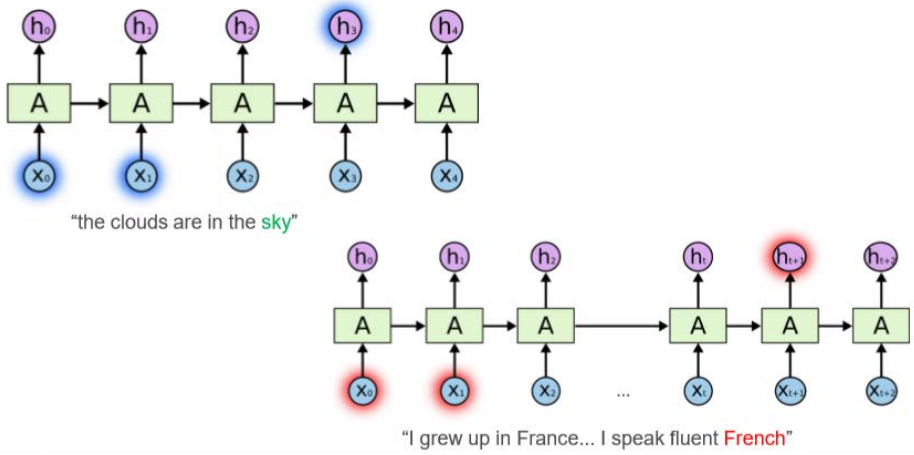


图 2.1.1 RNN 长跨度依赖问题的影响

长跨度依赖的根本问题在于，多阶段的反向传播后会导致梯度消失、梯度爆炸。可以使用梯度截断去解决梯度爆炸问题，但无法轻易解决梯度消失问题。

下面举一个例子来解释 RNN 梯度消失和爆炸的原因（例子来源：https://zhuanlan.zhihu.com/p/28687529）：

假设时间序列有三段，$h_0$为给定值，且为了简便假设没有激活函数和偏置，则 RNN 的前向传播过程如下：

$$h_1 = w_{xh}x_1 + w_{hh}h_0$$

$$h_2 = w_{xh}x_2 + w_{hh}h_1$$

$$h_3 = w_{xh}x_3 + w_{hh}h_2$$

$$y_1 = w_{hy}h_1$$

$$y_2 = w_{hy}h_2$$

$$y_3 = w_{hy}h_3$$

假设在 t=3 时刻，损失函数为$loss_3 = \frac{1}{2}(y_3 - y_true_3)^2$, 其余时刻类似。则$total_loss = \frac{1}{2}[(y_1 - y_true_1)^2 + (y_2 - y_true_2)^2 + (y_3 - y_true_3)^2]$。梯度下降法训练就是对参数分别求偏导，然后按照梯度反方向调整它们使 loss 值变小的过程。假设只考虑 t=3 时刻的$loss = loss_3$，这里考虑$w_{hh}$的偏导：

$$\frac{\partial_{loss}}{\partial_{w_{hh}}} = \frac{\partial_{loss}}{\partial_{y_3}} * \frac{\partial_{y_3}}{\partial_{h_3}} * \frac{\partial_{h_3}}{\partial_{w_{hh}}}$$

$$= \frac{\partial_{loss}}{\partial_{y_3}} * w_{hy} * \left(h_2 + w_{hh}\frac{\partial_{h_2}}{\partial_{w_{hh}}}\right)$$

$$= \frac{\partial_{loss}}{\partial_{y_3}} * w_{hy} * \left(h_2 + w_{hh}\left(h_1 + w_{hh}\frac{\partial_{h_1}}{\partial_{w_{hh}}}\right)\right)$$

$$= \frac{\partial_{loss}}{\partial_{y_3}} * w_{hy} * \left(h_2 + w_{hh}\left(h_1 + w_{hh}h_0\right)\right)$$

可以看出，只有三个时间点时，$w_{hh}$的偏导与$w_{hh}$的平方成比例。传统循环网络 RNN 可以通过记忆体实现短期记忆进行连续数据的预测，但是当连续数据的序列变长时会使展开时间步过长。当时间跨度变长时，幂次将变大。所以，如果$w_{hh}$为一个大于 0 小于 1 的数，随着时间跨度的增长，偏导值将会趋于 0；同理，当$w_{hh}$较大时，偏导值将趋于无穷。这就是梯度消失和爆炸的原因。

2，LSTM：

LSTM 由 Hochreiter & Schmidhuber 于 1997 年提出，通过门控单元很好的解

决了 RNN 长期依赖问题。Sepp Hochreiter,Jurgen Schmidhuber.LONG SHORT-TERM MEMORY.Neural Computation,December 1997.

（1） 原理：

为了解决长期依赖问题，长短记忆网络（Long Short Term Memory,LSTM）应运而生。之所以 LSTM 能解决 RNN 的长期依赖问题，是因为 LSTM 使用门（gate）机制对信息的流通和损失进行控制。

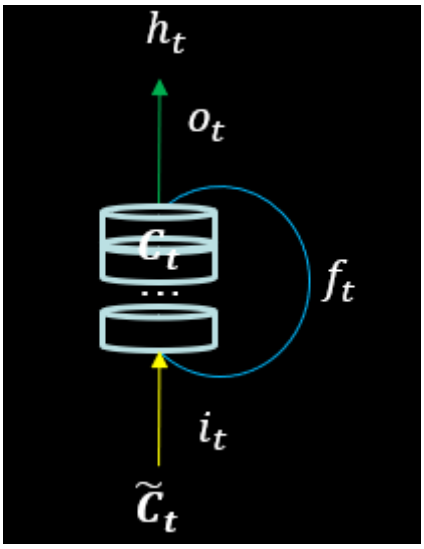


图 2.2.1 LSTM 计算过程

如图 2.2.1 所示，LSTM 引入了三个门限：输入门$i_t$、遗忘门$f_t$、输出门$o_t$；引入了表征长期记忆的细胞态$C_t$；引入了等待存入长期记忆的候选态$\tilde{C}_t$：

*三个门限都是当前时刻的输入特征$x_t$和上个时刻的短期记忆$h_{t-1}$的函数，分别表示为：

输入门（门限）：$i_t = \sigma(\mathrm{W}_i.[h_{t-1}, x_t] + \mathrm{b}_i)$，决定了多少比例的信息会被存入当前细胞态；

遗忘门（门限）：$f_t = \sigma(\mathrm{W}_f.[h_{t-1}, x_t] + \mathrm{b}_f)$，将细胞态中的信息选择性的遗忘；

输出门（门限）：$o_t = \sigma(\mathrm{W}_o.[h_{t-1}, x_t] + \mathrm{b}_o)$，将细胞态中的信息选择性的进行输出；

三个公式中$\mathrm{W}_i$、$\mathrm{W}_f$和$\mathrm{W}_o$是待训练参数矩阵，$\mathrm{b}_i$、$\mathrm{b}_f$和$\mathrm{b}_o$是待训练偏置项。$\sigma$为 sigmoid 激活函数，它可以使门限的范围在 0 到 1 之间。

*定义$h_t$为记忆体，它表征短期记忆，是当前细胞态经过输出门得到的：

属于长期记忆的一部分

记忆体（短期记忆）：$h_t = o_t * tanh(C_t)$

*候选态表示归纳出的待存入细胞态的新知识，是当前时刻的输入特征$x_t$和上个时刻的短期记忆$h_{t-1}$的函数：

Wc是待训练参数矩阵，bc是待训练偏置项

候选态（归纳出的新知识）：$\tilde{C}_t = tanh(W_c.[h_{t-1}, x_t] + b_c)$

拼接

*细胞态$C_t$表示长期记忆，它等于上个时刻的长期记忆$C_{t-1}$通过遗忘门的值和当前时刻归纳出的新知识$\tilde{C}_t$通过输入门的值之和：

细胞态（长期记忆）：$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

当明确了这些概念，这里举一个简单的例子理解一下 LSTM:

假设 LSTM 就是我们听老师讲课的过程，目前老师讲到了第 45 页 PPT。我们的脑袋里记住的内容，是 PPT 第 1 页到第 45 页的长期记忆$C_t$。它由两部分组成：一部分是 PPT 第 1 页到第 44 页的内容，也就是上一时刻的长期记忆$C_{t-1}$。我们不可能一字不差的记住全部内容，会不自觉地忘记了一些，所以上个时刻的长期记忆$C_{t-1}$要乘以遗忘门ft，这个乘积项就表示留存在我们脑中的对过去的记忆；另一部分是当前我们归纳出的新知识$\tilde{C}_t$（即候选态），它由老师正在讲的第 45 页 PPT（当前时刻的输入$x_t$）和第 44 页 PPT 的短期记忆留存（上一时刻的短期记忆$h_{t-1}$）组成。将现在的记忆$\tilde{C}_t$乘以输入门it后与过去的记忆一同存储为当前的长期记忆$C_t$。接下来，如果我们想把我们学到的知识（当前的长期记忆$C_t$）复述给朋友，我们不可能一字不落的讲出来，所以$C_t$需要经过输出门ot筛选后才成为了输出$h_t$。

当有多层循环网络时，第二层循环网络的输入$x_t$就是第一层循环网络的输出$h_t$，即输入第二层网络的是第一层网络提取出的精华。可以这么想，老师现在扮演的就是第一层循环网络，每一页 PPT 都是老师从一篇一篇论文中提取出的精华，输出给我们。作为第二层循环网络的我们，接收到的数据就是老师的长期记忆$C_t$过 tanh 激活函数后乘以输出门提取出的短期记忆$h_t$。

（2）Tensorflow2 描述 LSTM 层

记忆体个数　　是否每个时间步都输出ht

tf.keras.layers.LSTM(神经元个数，return_sequences=是否返回输出)

神经元个数和 return_sequences 的含义与 SimpleRNN 相同。

True表示每个时间步都输出ht
False表示仅最后一个时间步输出ht
一般最后一层用False，中间层用True
默认是False

例：LSTM(8, return_sequences=True)

（3）LSTM 股票预测

我们只需要将 RNN 预测股票中的模型更换为如图 2.2.2 所示即可。

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

仅在最后一个时间步输出ht

图 2.2.2 p47_LSTM_stock.py

图 2.2.3 为 loss 值曲线、图 2.2.4 为股票预测曲线、图 2.2.5 为三个评价指标值。

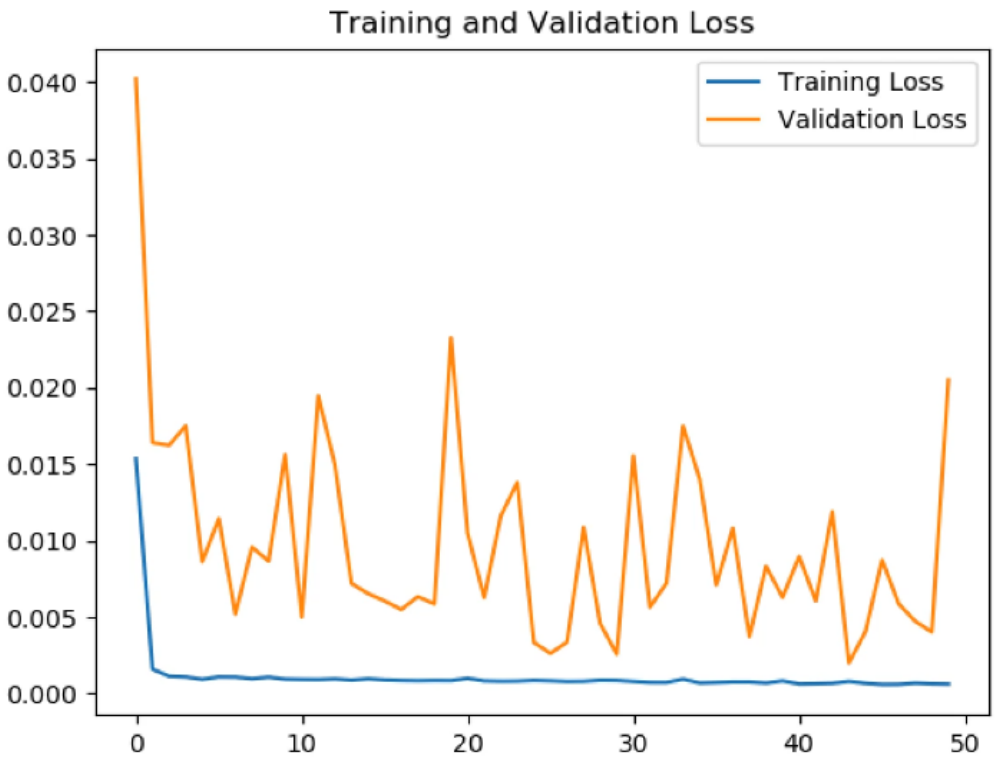


图 2.2.3 LSTM 股票预测 loss 曲线

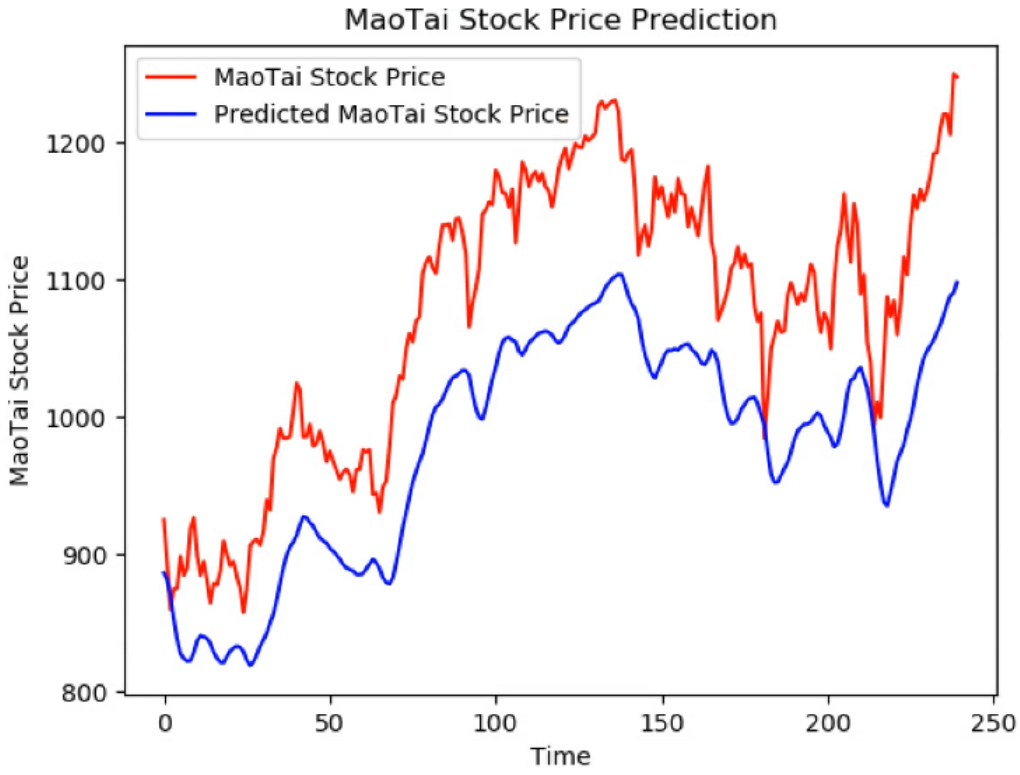


图 2.2.4 LSTM 股票预测曲线


图 2.2.5 LSTM 股票预测评价指标

3，GRU

GRU 由 Cho 等人于 2014 年提出，优化 LSTM 结构。Kyunghyun Cho,Bart van Merrienboer,Caglar Gulcehre,Dzmitry Bahdanau,Fethi Bougares,Holger Schwenk,Yoshua Bengio.Learning Phrase Representations using RNN Encoder－Decoder for Statistical Machine Translation.Computer ence, 2014.

（1）原理：

门控循环单元(Gated Recurrent Unit，GRU)是 LSTM 的一种变体，将 LSTM 中遗忘门与输入门合二为一为更新门，模型比 LSTM 模型更简单。

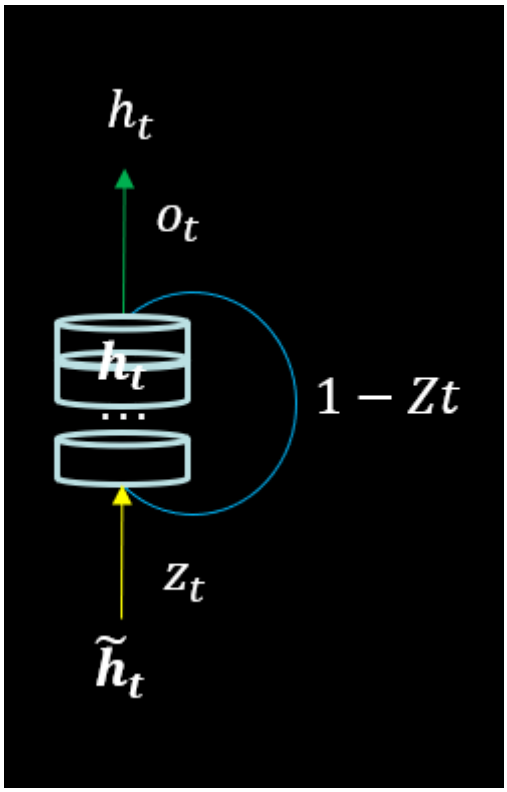


图 2.3.1 GRU 计算过程

如图 2.3.1 所示，GRU 使记忆体$h_t$融合了长期记忆和短期记忆。$h_t$包含了过去信息$h_{t-1}$和现在信息(候选隐藏层)$\tilde{h}_t$，由更新门$z_t$分配重要性：

记忆体：$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$ 前向传播时，直接使用这个记忆体更新公式，就可以算出每个时刻的ht值了

现在信息是过去信息$h_{t-1}$过重置门$r_t$与当前输入$x_t$共同决定的：

候选隐藏层：$\tilde{h}_t = tanh(W.[r_t * h_{t-1}, x_t])$

更新门和重置门的取值范围也是 0 到 1 之间：

更新门：$z_t = \sigma(\mathrm{W}_z.[h_{t-1}, x_t])$

重置门：$r_t = \sigma(\mathrm{W}_r.[h_{t-1}, x_t])$

（2）Tensorflow2 描述 GRU 层

tf.keras.layers.GRU(神经元个数， return_sequences=是否返回输出)

神经元个数和 return_sequences 的含义与 SimpleRNN 相同。

例：GRU(8, return_sequences=True)

（3）GRU 股票预测

我们只需要将 RNN 预测股票中的模型更换为如图 2.3.2 所示即可。

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

图 2.3.2 p56_GRU_stock.py

图 2.3.3 为 loss 值曲线、图 2.3.4 为股票预测曲线、图 2.3.5 为三个评价指标值。

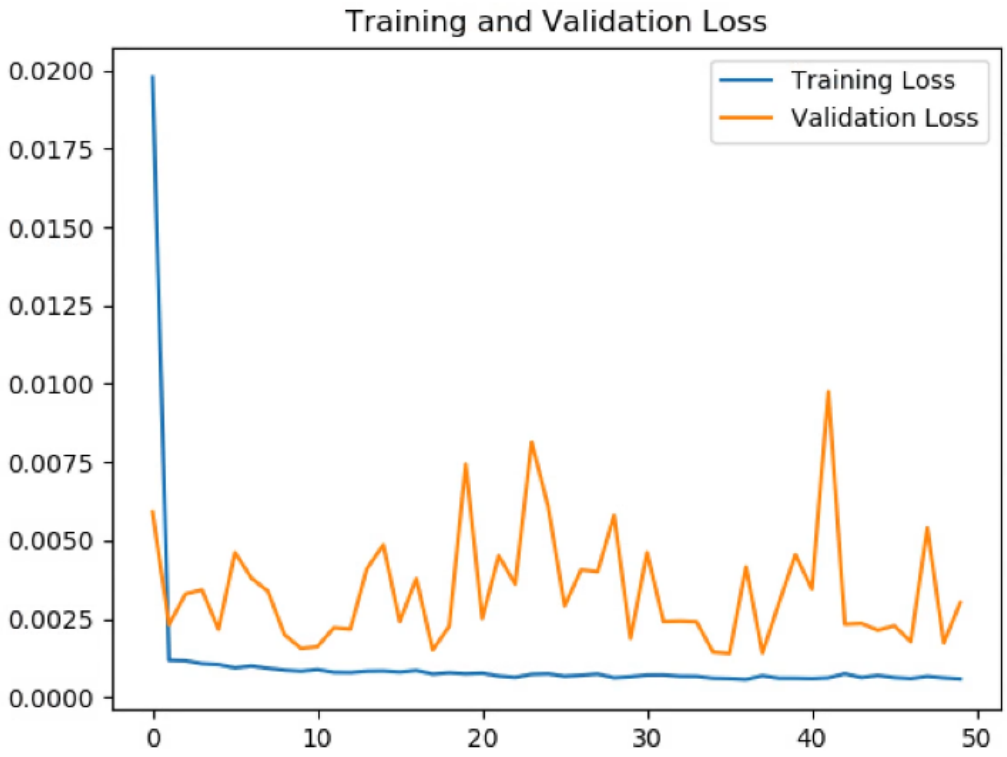


图 2.3.3 GRU 股票预测 loss 曲线

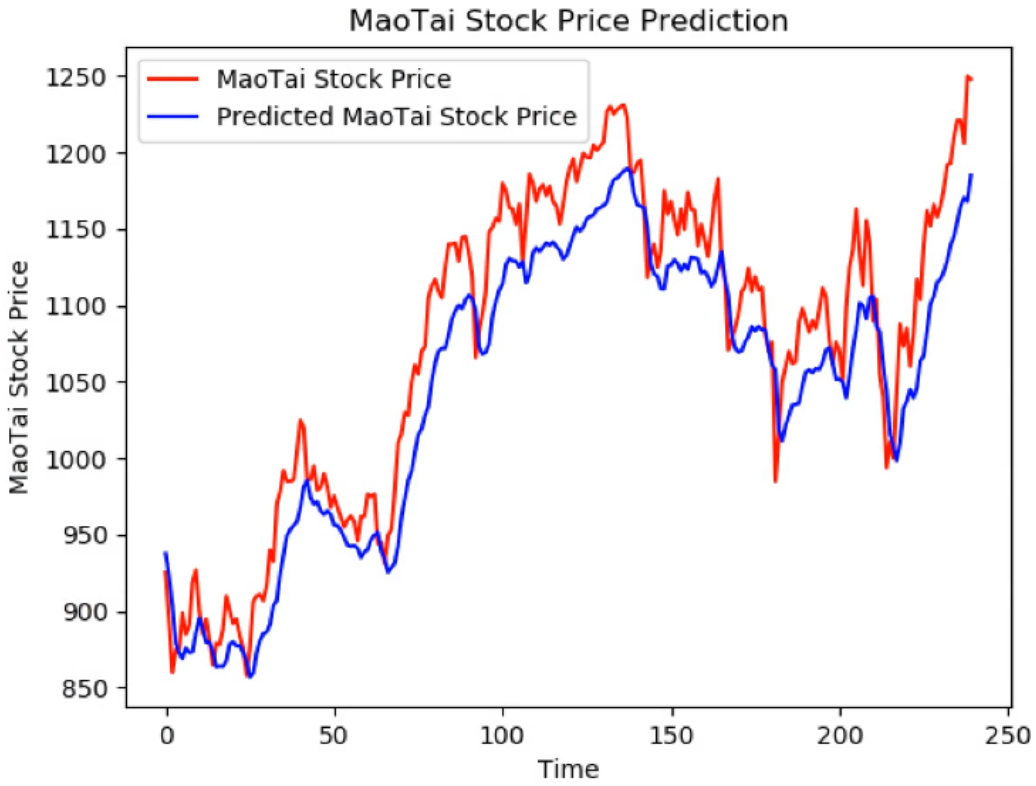


图 2.3.4 GRU 股票预测曲线

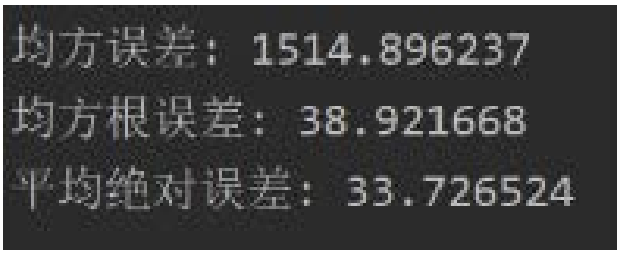


图 2.3.5 GRU 股票预测评价指标