# HW3

## 一、环境配置

本机上已经配置过了

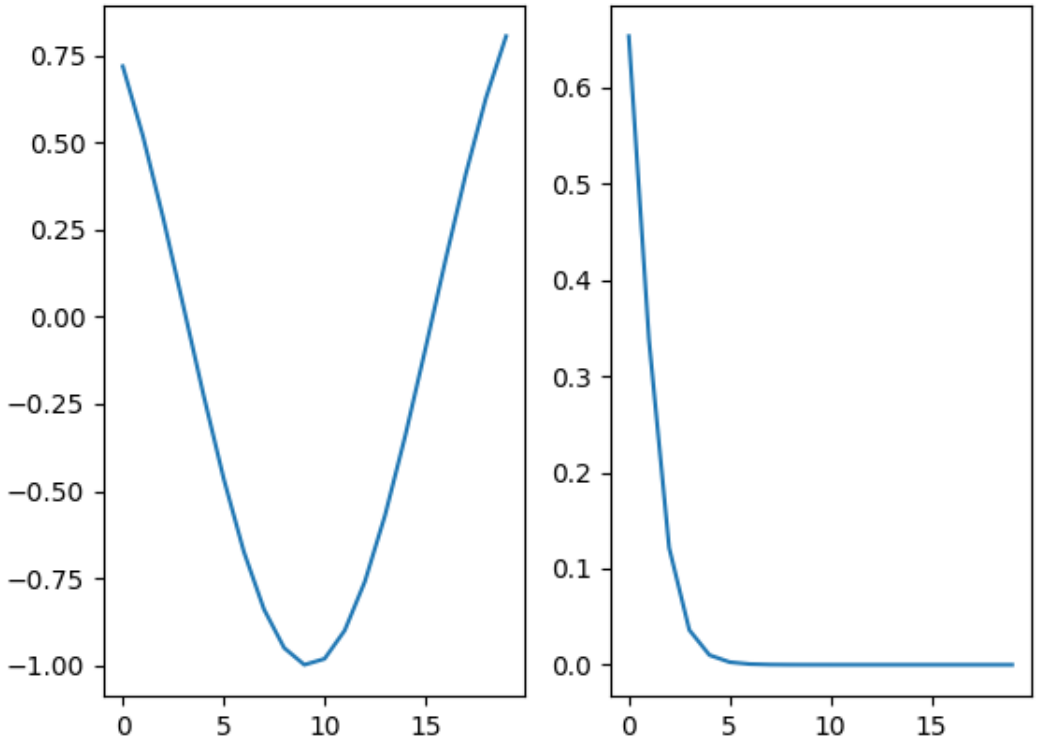
## 二、Code Complete

### 2.1 baseline

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt

DATA_SIZE = 1000

sine_data_size = np.random.randint(int(0.3 * DATA_SIZE), int(0.7 * DATA_SIZE))
sigmoid_data_size = DATA_SIZE - sine_data_size

steps = np.arange(0, 10, 0.5)

# generate sine-like function samples
sine_init = np.random.uniform(-3, 3, (sine_data_size, 2))  # randomize a and b
for sin(ax+b)
sine_data = np.sin(sine_init[:, :1] * steps + sine_init[:, 1:])

# generate sigmoid-like function samples
sigmoid_init = np.random.uniform(-3, 3, (sigmoid_data_size, 2)) # randomize a and
b for 1/(1+e^(-ax+b))
sigmoid_data = 1 / (1 + np.exp(0 - sigmoid_init[:, :1] * steps + sigmoid_init[:,
1:]))
fig, axs = plt.subplots(1, 2)
axs[0].plot(sine_data[0])
axs[1].plot(sigmoid_data[1])
plt.show()
# mix data
sine_data = np.concatenate((sine_data, np.ones((sine_data_size, 1))), axis=1)
sigmoid_data = np.concatenate((sigmoid_data, np.zeros((sigmoid_data_size, 1))),
axis=1)
data = np.concatenate((sine_data, sigmoid_data), axis=0)
data = torch.Tensor(data)

# split two datasets
from torch.utils.data import random_split
train_set, test_set = random_split(data, [0.8, 0.2])

# define network
class SimpleClassificationRNN(nn.Module):
    def __init__(self, hidden_size):
        super(SimpleClassificationRNN, self).__init__()
```

```
        '''
        task 1: write network structure here using nn.RNN
        '''
        self.rnn = nn.RNN(input_size=1, hidden_size=hidden_size,
batch_first=True, num_layers=1)
        self.linear = nn.Linear(hidden_size, 1)

    def forward(self, seq, hc=None):
        '''
        task 2: write forward process
        '''
        tmp, hc = self.rnn(seq, hc)
        out = torch.sigmoid(self.linear(hc[-1, ... ,:]))
        return out, hc

hidden_size = 16
learning_rate = 0.01

model = SimpleClassificationRNN(hidden_size)

'''
task 3: select appropriate criterion and optimizer
'''
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), learning_rate)

def cal_accuracy(preds, true_values):
    preds = torch.where(preds>0.5, 1, 0)
    acc = torch.sum(1-torch.abs(preds-true_values)) / preds.shape[0]
    return acc
# training ...

epochs = 500
loss_log = []
for epoch in range(epochs):
    optimizer.zero_grad()
    output, _ = model(train_set[:][:, :-1, np.newaxis])
    loss = criterion(output.view(-1), train_set[:][:, -1])
    acc = cal_accuracy(output.view(-1), train_set[:][:, -1])
    loss.backward()
    optimizer.step()
    if epoch % 10 == 0:
        print("Epoch {}: loss {} acc {}".format(epoch, loss.item(), acc))
# performance on test set

output, _ = model(test_set[:][:, :-1, np.newaxis])
loss = criterion(output.view(-1), test_set[:][:, -1])
acc = cal_accuracy(output.view(-1), test_set[:][:, -1])

print("Test set: loss {} acc {}".format(loss.item(), acc))
```

运行如下：




```
Epoch 360: loss 0.05446205288171768 acc 0.9887499809265137
Epoch 370: loss 0.05335211753845215 acc 0.9887499809265137
Epoch 380: loss 0.05263228341937065 acc 0.9887499809265137
Epoch 390: loss 0.05194520950317383 acc 0.9887499809265137
Epoch 400: loss 0.05124456435441971 acc 0.9887499809265137
Epoch 410: loss 0.050512686371803284 acc 0.9887499809265137
Epoch 420: loss 0.04977012053132057 acc 0.9887499809265137
Epoch 430: loss 0.04906436428427696 acc 0.9887499809265137
Epoch 440: loss 0.04844847694039345 acc 0.9887499809265137
Epoch 450: loss 0.047916460782289505 acc 0.9887499809265137
Epoch 460: loss 0.047403693199157715 acc 0.9887499809265137
Epoch 470: loss 0.04686314985156059 acc 0.9887499809265137
Epoch 480: loss 0.04626360908150673 acc 0.9887499809265137
Epoch 490: loss 0.04558933153748512 acc 0.9900000095367432
Test set: loss 0.037420582026243210 acc 0.9900000095367432
```

## 2.2 early-stopping

注意到对于有的数据，baseline的方法可能会出现过拟合：

```
Epoch 360: loss 0.6518135070800781 acc 0.8737499713897705
Epoch 370: loss 0.29629650712013245 acc 0.9137499928474426
Epoch 380: loss 0.27731063961982727 acc 0.9312499761581421
Epoch 390: loss 0.1594288945198059 acc 0.9524999856948853
Epoch 400: loss 0.26239970326423645 acc 0.9624999761581421
Epoch 410: loss 0.17269426584243774 acc 0.9725000262260437
Epoch 420: loss 0.6180518865585327 acc 0.7662500143051147
Epoch 430: loss 0.5115599632263184 acc 0.8587499856948853
Epoch 440: loss 0.4566730260848999 acc 0.8399999737739563
Epoch 450: loss 0.44526684284210205 acc 0.8262500166893005
Epoch 460: loss 0.4132120609283447 acc 0.8525000214576721
Epoch 470: loss 0.3864530324935913 acc 0.862500011920929
Epoch 480: loss 0.3603284955024719 acc 0.8662499785423279
Epoch 490: loss 0.3358425796031952 acc 0.8824999928474426
Test set: loss 0.36604297161102295 acc 0.8650000095367432
```

、

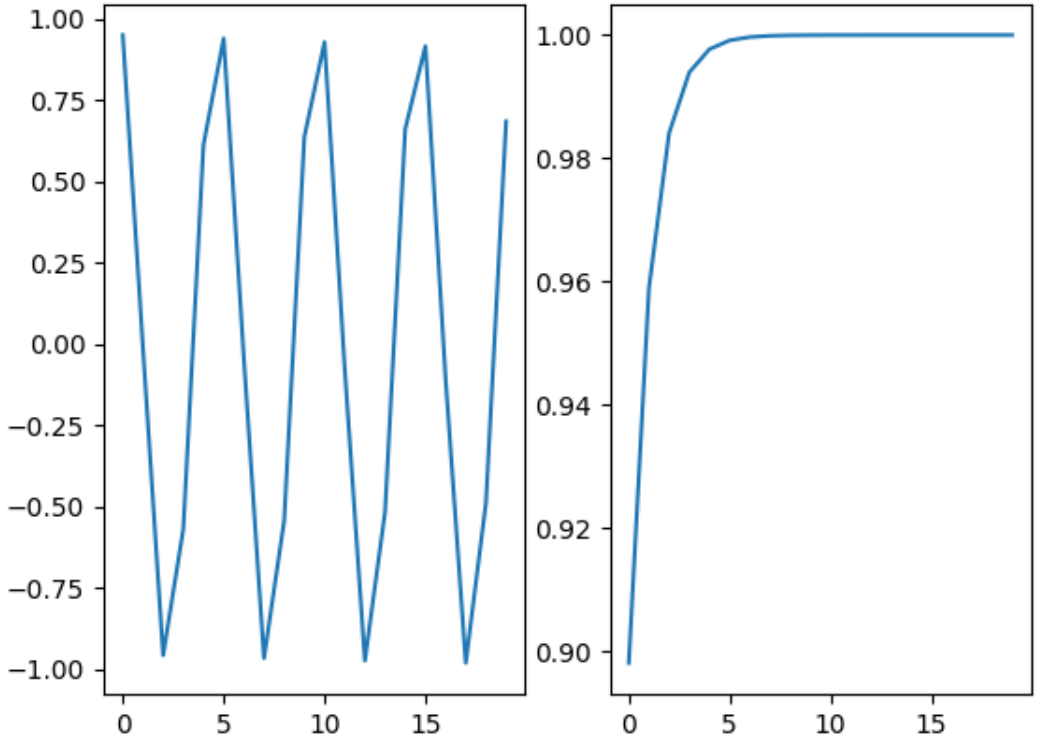
采用early-stopping的方法防止过拟合：

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score

DATA_SIZE = 1000

sine_data_size = np.random.randint(int(0.3 * DATA_SIZE), int(0.7 * DATA_SIZE))
sigmoid_data_size = DATA_SIZE - sine_data_size

steps = np.arange(0, 10, 0.5)

# generate sine-like function samples
sine_init = np.random.uniform(-3, 3, (sine_data_size, 2))  # randomize a and b
for sin(ax+b)
sine_data = np.sin(sine_init[:, :1] * steps + sine_init[:, 1:])

# generate sigmoid-like function samples
sigmoid_init = np.random.uniform(-3, 3, (sigmoid_data_size, 2)) # randomize a and
b for 1/(1+e^(-ax+b))
sigmoid_data = 1 / (1 + np.exp(0 - sigmoid_init[:, :1] * steps + sigmoid_init[:,
1:]))
fig, axs = plt.subplots(1, 2)
axs[0].plot(sine_data[0])
axs[1].plot(sigmoid_data[1])
plt.show()

# mix data
sine_data = np.concatenate((sine_data, np.ones((sine_data_size, 1))), axis=1)
sigmoid_data = np.concatenate((sigmoid_data, np.zeros((sigmoid_data_size, 1))),
axis=1)
```

```
data = np.concatenate((sine_data, sigmoid_data), axis=0)
data = torch.Tensor(data)

# split two datasets
from torch.utils.data import random_split
train_set, test_set = random_split(data, [0.8, 0.2])

# define network
class SimpleClassificationRNN(nn.Module):
    def __init__(self, hidden_size):
        super(SimpleClassificationRNN, self).__init__()
        self.rnn = nn.RNN(input_size=1, hidden_size=hidden_size,
batch_first=True, num_layers=1)
        self.linear = nn.Linear(hidden_size, 1)

    def forward(self, seq, hc=None):
        tmp, hc = self.rnn(seq, hc)
        out = torch.sigmoid(self.linear(hc[-1, :, :]))
        return out, hc

hidden_size = 16
learning_rate = 0.01

model = SimpleClassificationRNN(hidden_size)
criterion = nn.BCELoss()

# Test different optimizers and learning rates
optimizers = {
    'Adam': optim.Adam(model.parameters(), lr=0.01),
    'SGD': optim.SGD(model.parameters(), lr=0.1),
    'RMSprop': optim.RMSprop(model.parameters(), lr=0.01)
}

# Use Adam optimizer in this example
optimizer = optimizers['Adam']

def cal_accuracy(preds, true_values):
    preds = torch.where(preds>0.5, 1, 0)
    acc = torch.sum(1-torch.abs(preds-true_values)) / preds.shape[0]
    return acc

def cal_f1(preds, true_values):
    preds = torch.where(preds>0.5, 1, 0).cpu().numpy()
    true_values = true_values.cpu().numpy()
    f1 = f1_score(true_values, preds)
    return f1

# training with early stopping
epochs = 500
loss_log = []
best_acc = 0
patience = 20
counter = 0

for epoch in range(epochs):
    model.train()
```

```
    optimizer.zero_grad()
    output, _ = model(train_set[:][:, :-1, np.newaxis])
    loss = criterion(output.view(-1), train_set[:][:, -1])
    acc = cal_accuracy(output.view(-1), train_set[:][:, -1])
    loss.backward()
    optimizer.step()
    if epoch % 10 == 0:
        print("Epoch {}: loss {} acc {}".format(epoch, loss.item(), acc))
    # Early stopping
    model.eval()
    output, _ = model(test_set[:][:, :-1, np.newaxis])
    test_acc = cal_accuracy(output.view(-1), test_set[:][:, -1])
    if test_acc > best_acc:
        best_acc = test_acc
        counter = 0
    else:
        counter += 1
    if counter >= patience:
        print("Early stopping at epoch {}".format(epoch))
        break

# performance on test set
output, _ = model(test_set[:][:, :-1, np.newaxis])
loss = criterion(output.view(-1), test_set[:][:, -1])
acc = cal_accuracy(output.view(-1), test_set[:][:, -1])
f1 = cal_f1(output.view(-1), test_set[:][:, -1])

print("Test set: loss {} acc {} f1 {}".format(loss.item(), acc, f1))
```

运行结果如下：



```
PS E:\MARL_JUEWU\HW3> python rnn_task2.py
Epoch 0: loss 0.711223304271698 acc 0.38374999165534973
Epoch 10: loss 0.5707754492759705 acc 0.7749999761581421
Epoch 20: loss 0.4124735891819 acc 0.8274999856948853
Epoch 30: loss 0.35560908913612366 acc 0.8500000238418579
Epoch 40: loss 0.2586541771888733 acc 0.9300000071525574
Epoch 50: loss 0.14451591670513153 acc 0.9549999833106995
Epoch 60: loss 0.13181479275226593 acc 0.9712499976158142
Epoch 70: loss 0.2130882441997528 acc 0.9362499713897705
Early stopping at epoch 70
Test set: loss 0.15577811002731323 acc 0.9750000238418579 f1 0.9640287769784173
```

关闭

## 2.3 dropout+early-stopping

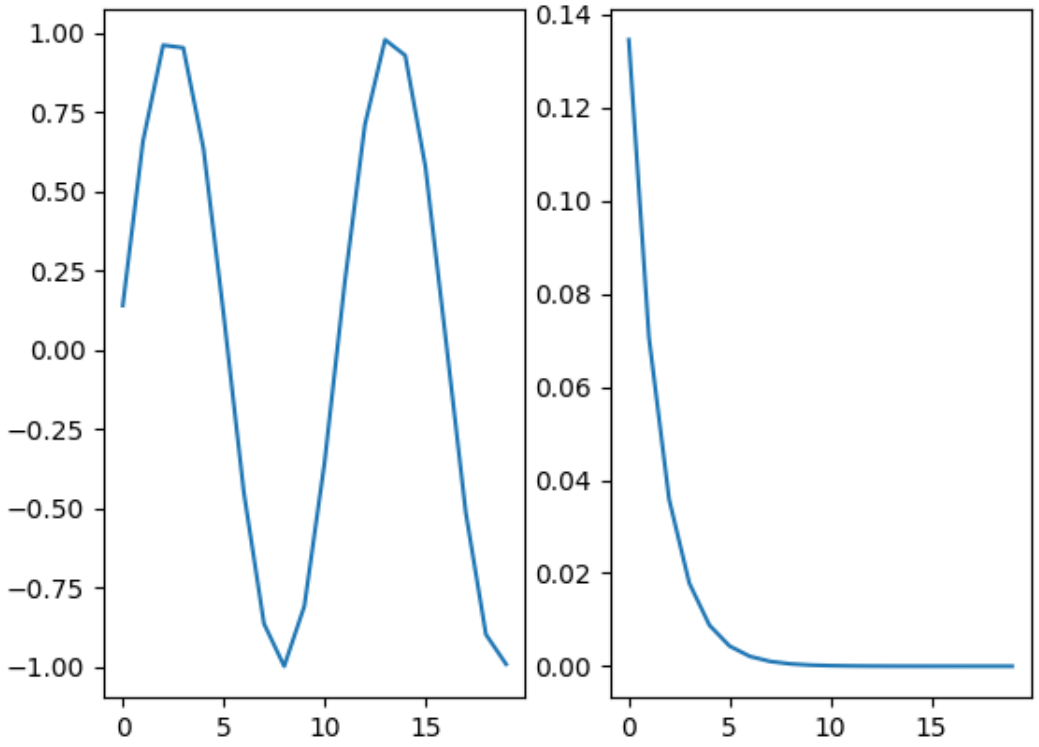
在2.2的基础上增加归一化，进一步防止过拟合：

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score

DATA_SIZE = 1000

sine_data_size = np.random.randint(int(0.3 * DATA_SIZE), int(0.7 * DATA_SIZE))
sigmoid_data_size = DATA_SIZE - sine_data_size

steps = np.arange(0, 10, 0.5)

# generate sine-like function samples
sine_init = np.random.uniform(-3, 3, (sine_data_size, 2))  # randomize a and b
for sin(ax+b)
sine_data = np.sin(sine_init[:, :1] * steps + sine_init[:, 1:])

# generate sigmoid-like function samples
sigmoid_init = np.random.uniform(-3, 3, (sigmoid_data_size, 2)) # randomize a and
b for 1/(1+e^(-ax+b))
sigmoid_data = 1 / (1 + np.exp(0 - sigmoid_init[:, :1] * steps + sigmoid_init[:,
1:]))
fig, axs = plt.subplots(1, 2)
axs[0].plot(sine_data[0])
axs[1].plot(sigmoid_data[1])
plt.show()

# mix data
sine_data = np.concatenate((sine_data, np.ones((sine_data_size, 1))), axis=1)
sigmoid_data = np.concatenate((sigmoid_data, np.zeros((sigmoid_data_size, 1))),
axis=1)
data = np.concatenate((sine_data, sigmoid_data), axis=0)
data = torch.Tensor(data)

# split two datasets
from torch.utils.data import random_split
train_set, test_set = random_split(data, [0.8, 0.2])
```

```
# define network
class SimpleClassificationRNN(nn.Module):
    def __init__(self, hidden_size):
        super(SimpleClassificationRNN, self).__init__()
        self.rnn = nn.RNN(input_size=1, hidden_size=hidden_size,
batch_first=True, num_layers=1)
        self.dropout = nn.Dropout(p=0.5)  # Adding dropout for regularization
        self.linear = nn.Linear(hidden_size, 1)

    def forward(self, seq, hc=None):
        tmp, hc = self.rnn(seq, hc)
        tmp = self.dropout(tmp)  # Apply dropout
        out = torch.sigmoid(self.linear(hc[-1, :, :]))
        return out, hc

hidden_size = 16
learning_rate = 0.01

model = SimpleClassificationRNN(hidden_size)
criterion = nn.BCELoss()

# Test different optimizers and learning rates
optimizers = {
    'Adam': optim.Adam(model.parameters(), lr=0.01),
    'SGD': optim.SGD(model.parameters(), lr=0.1),
    'RMSprop': optim.RMSprop(model.parameters(), lr=0.01)
}

# Use Adam optimizer in this example
optimizer = optimizers['Adam']

def cal_accuracy(preds, true_values):
    preds = torch.where(preds > 0.5, 1, 0)
    acc = torch.sum(1 - torch.abs(preds - true_values)) / preds.shape[0]
    return acc

def cal_f1(preds, true_values):
    preds = torch.where(preds > 0.5, 1, 0).cpu().numpy()
    true_values = true_values.cpu().numpy()
    f1 = f1_score(true_values, preds)
    return f1

# training with early stopping
epochs = 500
loss_log = []
best_acc = 0
patience = 20
counter = 0

for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    output, _ = model(train_set[:][:, :-1, np.newaxis])
    loss = criterion(output.view(-1), train_set[:][:, -1])
    acc = cal_accuracy(output.view(-1), train_set[:][:, -1])
    loss.backward()
```

```
    optimizer.step()

    if epoch % 10 == 0:
        print("Epoch {}: loss {} acc {}".format(epoch, loss.item(), acc))

    # Early stopping
    model.eval()
    output, _ = model(test_set[:][:, :-1, np.newaxis])
    test_acc = cal_accuracy(output.view(-1), test_set[:][:, -1])

    if test_acc > best_acc:
        best_acc = test_acc
        counter = 0
    else:
        counter += 1

    if counter >= patience:
        print("Early stopping at epoch {}".format(epoch))
        break

# performance on test set
output, _ = model(test_set[:][:, :-1, np.newaxis])
loss = criterion(output.view(-1), test_set[:][:, -1])
acc = cal_accuracy(output.view(-1), test_set[:][:, -1])
f1 = cal_f1(output.view(-1), test_set[:][:, -1])

print("Test set: loss {} acc {} f1 {}".format(loss.item(), acc, f1))
```

运行结果如下：



```
PS E:\MARL_JUEWU\HW3> python .\rnn_task_dropout_es.py
Epoch 0: loss 0.6890547275543213 acc 0.5375000238418579
Epoch 10: loss 0.49243542551994324 acc 0.7587500214576721
Epoch 20: loss 0.46549659967422485 acc 0.768750011920929
Epoch 30: loss 0.37810570001602173 acc 0.8237500190734863
Epoch 40: loss 0.24651989340782166 acc 0.9100000262260437
Epoch 50: loss 0.1378209888935089 acc 0.9662500023841858
Epoch 60: loss 0.10406380891799927 acc 0.9750000238418579
Epoch 70: loss 0.0934300646185875 acc 0.9762499928474426
Epoch 80: loss 0.08347737044095993 acc 0.9800000190734863
Epoch 90: loss 0.07873210310935974 acc 0.9800000190734863
Epoch 100: loss 0.2050207108259201 acc 0.9462500214576721
Early stopping at epoch 101
Test set: loss 0.25717365741729736 acc 0.925000011920929 f1 0.9206349206349206
```

### 2.4 scheduler+early-stopping

在2.2的基础上使用scheduler调度学习率：

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score
from torch.utils.data import random_split

# 数据生成
DATA_SIZE = 1000
sine_data_size = np.random.randint(int(0.3 * DATA_SIZE), int(0.7 * DATA_SIZE))
sigmoid_data_size = DATA_SIZE - sine_data_size
steps = np.arange(0, 10, 0.5)
sine_init = np.random.uniform(-3, 3, (sine_data_size, 2))
sine_data = np.sin(sine_init[:, :1] * steps + sine_init[:, 1:])
sigmoid_init = np.random.uniform(-3, 3, (sigmoid_data_size, 2))
sigmoid_data = 1 / (1 + np.exp(0 - sigmoid_init[:, :1] * steps + sigmoid_init[:,
1:]))
sine_data = np.concatenate((sine_data, np.ones((sine_data_size, 1))), axis=1)
sigmoid_data = np.concatenate((sigmoid_data, np.zeros((sigmoid_data_size, 1))),
axis=1)
data = np.concatenate((sine_data, sigmoid_data), axis=0)
data = torch.Tensor(data)
train_set, test_set = random_split(data, [0.8, 0.2])

# 定义网络
class SimpleClassificationRNN(nn.Module):
    def __init__(self, hidden_size):
        super(SimpleClassificationRNN, self).__init__()
        self.rnn = nn.RNN(input_size=1, hidden_size=hidden_size,
batch_first=True, num_layers=1)
        self.linear = nn.Linear(hidden_size, 1)

    def forward(self, seq, hc=None):
        tmp, hc = self.rnn(seq, hc)
        out = torch.sigmoid(self.linear(hc[-1, :, :]))
        return out, hc
```

```
hidden_size = 16
learning_rate = 0.01
model = SimpleClassificationRNN(hidden_size)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=10,
factor=0.5)

def cal_accuracy(preds, true_values):
    preds = torch.where(preds > 0.5, 1, 0)
    acc = torch.sum(1 - torch.abs(preds - true_values)) / preds.shape[0]
    return acc

def cal_f1(preds, true_values):
    preds = torch.where(preds > 0.5, 1, 0).cpu().numpy()
    true_values = true_values.cpu().numpy()
    f1 = f1_score(true_values, preds)
    return f1

# 训练与早停策略
epochs = 500
loss_log = []
best_acc = 0
patience = 20
counter = 0

for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    output, _ = model(train_set[:][:, :-1, np.newaxis])
    loss = criterion(output.view(-1), train_set[:][:, -1])
    acc = cal_accuracy(output.view(-1), train_set[:][:, -1])
    loss.backward()
    optimizer.step()

    # 调度器
    scheduler.step(loss)

    if epoch % 10 == 0:
        print("Epoch {}: loss {} acc {}".format(epoch, loss.item(), acc))

    # 早停策略
    model.eval()
    output, _ = model(test_set[:][:, :-1, np.newaxis])
    test_loss = criterion(output.view(-1), test_set[:][:, -1])
    test_acc = cal_accuracy(output.view(-1), test_set[:][:, -1])

    if test_acc > best_acc:
        best_acc = test_acc
        counter = 0
    else:
        counter += 1

    if counter >= patience:
        print("Early stopping at epoch {}".format(epoch))
```

```
        break

# 测试集上的表现
output, _ = model(test_set[:][:, :-1, np.newaxis])
loss = criterion(output.view(-1), test_set[:][:, -1])
acc = cal_accuracy(output.view(-1), test_set[:][:, -1])
f1 = cal_f1(output.view(-1), test_set[:][:, -1])

print("Test set: loss {} acc {} f1 {}".format(loss.item(), acc, f1))
```

运行结果如下：

```
PS E:\MARL_JUEWU\HW3> python .\rnn_task_sc_es.py
Epoch 0: loss 0.670623779296875 acc 0.6862499713897705
Epoch 10: loss 0.5564970970153809 acc 0.7649999856948853
Epoch 20: loss 0.398080974817276 acc 0.8387500047683716
Epoch 30: loss 0.3917495608329773 acc 0.8487499952316284
Epoch 40: loss 0.2994972765445709 acc 0.8924999833106995
Epoch 50: loss 0.13371632993221283 acc 0.9649999737739563
Epoch 60: loss 0.16902607679367065 acc 0.9612500071525574
Epoch 70: loss 0.07310260832309723 acc 0.9862499833106995
Epoch 80: loss 0.06441682577133179 acc 0.9862499833106995
Early stopping at epoch 85
Test set: loss 0.05949251726269722 acc 0.9800000190734863 f1 0.9743589743589743
```