

```
In [1]: %matplotlib inline
import torch
import numpy as np
import pandas as pd
from torch import nn
import matplotlib.pyplot as plt
```

```
In [2]: def squared_loss(y_hat, y):
        """均方损失。"""
        return ((y_hat - y.reshape(y_hat.shape))**2 / 2).mean()
```

```
In [3]: def set_learning_rate(optimizer, lr):
        for param_group in optimizer.param_groups:
            param_group['lr'] = lr
```

```
In [4]: def train(net, optimizer, loss, train_features, train_res, get_step=None, newton_Method=False):
        theta=torch.ones(net[0].weight.shape)
        loss_list=[]
        theta_bias=[]
        while torch.norm((net[0].weight-theta).squeeze(0),p=2,dim=0).item()**2 > 10**(-4):
            theta=torch.tensor(net[0].weight)
            l=loss(net(train_features), train_res)

            if net[0].weight.grad!=None:
                net[0].weight.grad.zero_()
            grad = torch.autograd.grad(l, net[0].weight, retain_graph=True, create_graph=True)
            grad=grad[0]

            if newton_Method==True:
                grad=grad.squeeze(0)
                Print = torch.tensor([])
                for anygrad in grad:
                    Print = torch.cat((Print, torch.autograd.grad(anygrad, net[0].weight,
                        grad=torch.mm(grad.reshape(1,2), torch.linalg.inv(Print))))
                net[0].weight.grad=grad

            if get_step!=None:
                set_learning_rate(optimizer, get_step(loss, net, train_features, train_res))

            optimizer.step()
            #准备可视化的数据
            #print("l:", l)
            loss_list.append(l.item())
            #print(torch.norm((net[0].weight-theta).squeeze(0),p=2,dim=0).item()**2 )
            theta_bias.append(torch.norm((net[0].weight-theta).squeeze(0),p=2,dim=0).item())
        return loss_list, theta_bias, theta
```

```
In [5]: #1-1-1
train_data_x=pd.read_csv("T:\project\programming\DeepLearning\experiment\dataset\data1.csv")
train_data_y=pd.read_csv("T:\project\programming\DeepLearning\experiment\dataset\data2.csv")

train_features=torch.tensor(train_data_x.values, dtype=torch.float32)
train_features=torch.cat([train_features, torch.ones(49,1)], dim=1)
train_res=torch.tensor(train_data_y.values, dtype=torch.float32)
```

```
In [6]:
```

```
#初始化超参数
net=nn.Sequential( nn.Linear(train_features.shape[1],1,bias=False) )
net[0].weight.data.fill_(0)
#net[0].bias.data.fill_(0)
optimizer=torch.optim.SGD(net.parameters(), 0.03)
loss = squared_loss
```

In [7]:

```
loss_list,theta_bias,theta=train(net,optimizer,loss,train_features,train_res)
epoches=np.arange(len(loss_list))
fig, ax = plt.subplots()
fmt='g-'
ax.set_title('Experiment1-1 loss descend')
ax.plot(epoches, loss_list,fmt)

fig_2, ax_2 = plt.subplots()
fmt='m-'
ax_2.set_title('Experiment1-1 theta subtract descend')
ax_2.plot(epoches, theta_bias,fmt)

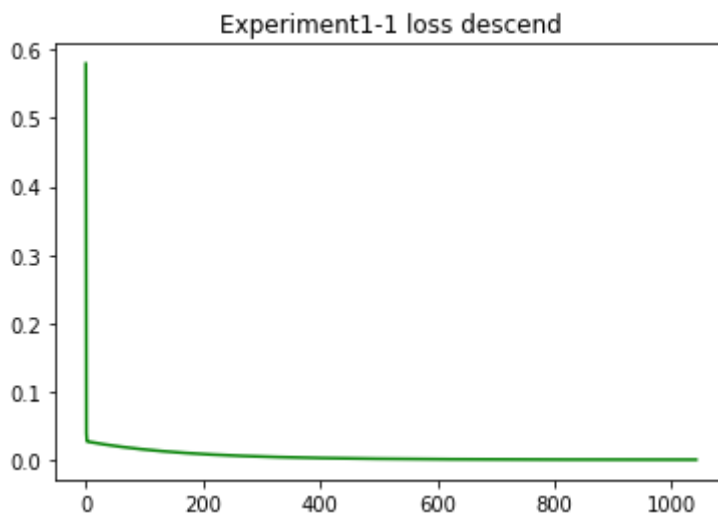
print("Experiment1-1 theta subtraction ",(torch.norm((net[0].weight-theta).squeeze(0)
print("Experiment1-1 loss ",squared_loss(net(train_features),train_res).data)
```

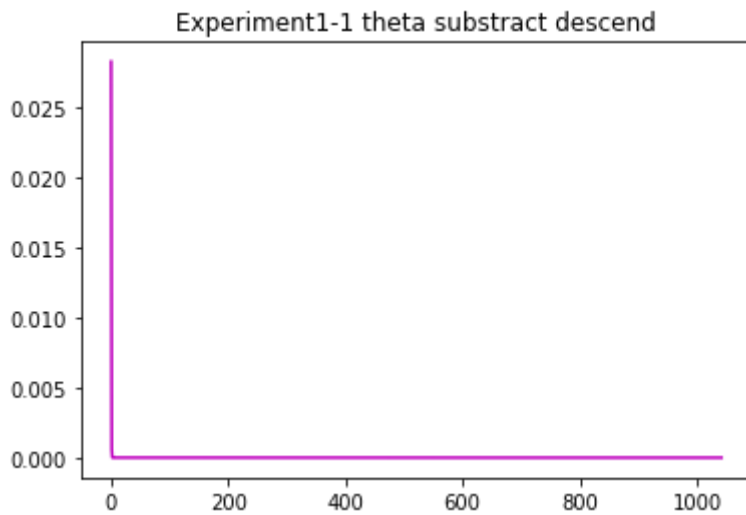
C:\Users\Young\AppData\Local\Temp\ipykernel\_7520\1599417071.py:6: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().requires\_grad\_(True), rather than torch.tensor(sourceTensor).

```
theta=torch.tensor(net[0].weight)
```

Experiment1-1 theta subtraction 9.976567192460472e-09

Experiment1-1 loss tensor(0.0009)





```
In [8]: #l1-2
train_data2_x=pd.read_csv("T:\project\programming\DeepLearning\experiment\dataset\data2_x.csv")
train_data2_y=pd.read_csv("T:\project\programming\DeepLearning\experiment\dataset\data2_y.csv")
print(train_data2_x.shape)
#标准化
train_data2_x[:, :] = train_data2_x[:, :].apply(
    lambda x: (x - x.mean()) / (x.std()))

train2_features=torch.tensor(train_data2_x.values, dtype=torch.float32)
train2_features=torch.cat([torch.ones(46, 1), train2_features], dim=1)
train2_res=torch.tensor(train_data2_y.values, dtype=torch.float32)

(46, 2)
```

```
In [9]: #初始化超参数
net=nn.Sequential( nn.Linear(train2_features.shape[1], 1, bias=False) )
net[0].weight.data.fill_(0)
#net[0].bias.data.fill_(0)
optimizer=torch.optim.SGD(net.parameters(), 0.03)
loss = squared_loss
```

```
In [10]: loss_list_train2, theta_bias_train2, theta_train2=train(net, optimizer, loss, train2_features)

epoches_train2=np.arange(len(loss_list_train2))
fig_train2, ax_train2 = plt.subplots()
fmt='g-'
ax_train2.set_title('Experiment1-2 loss descend')
ax_train2.plot(epoches_train2, loss_list_train2, fmt)

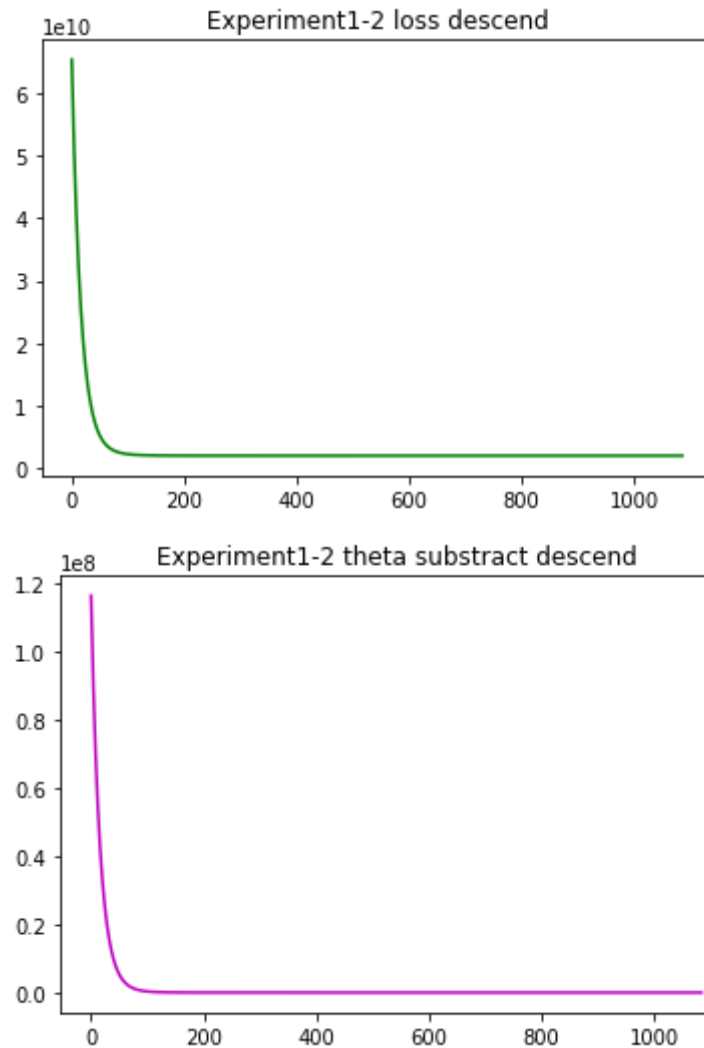
fig_2_train2, ax_2_train2 = plt.subplots()
fmt='m-'
ax_2_train2.set_title('Experiment1-2 theta subtract descend')
ax_2_train2.plot(epoches_train2, theta_bias_train2, fmt)

print("Experiment1-1-2 theta subtraction ", (torch.norm((net[0].weight-theta_train2)).data))
print("Experiment1-1-2 loss ", squared_loss(net(train2_features), train2_res).data)
```

C:\Users\Young\AppData\Local\Temp\ipykernel\_7520\1599417071.py:6: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().requires\_grad\_(True), rather than torch.tensor(sourceTensor).

```
theta=torch.tensor(net[0].weight)
Experiment1-1-2 theta subtraction 0.0
```

Experiment1-1-2 loss tensor(2.0665e+09)



```
In [11]: #1-2
train_data3x=pd.read_csv("T:\project\programming\DeepLearning\experiment\dataset\data3x.csv")
train_data3y=pd.read_csv("T:\project\programming\DeepLearning\experiment\dataset\data3y.csv")

train3_features=torch.tensor(train_data3x.values, dtype=torch.float32)
train3_res=torch.tensor(train_data3y.values, dtype=torch.float32)
```

```
In [12]: #我自己写的优化方法，假设方向为一阶导数，改变步长；
# 会出现z字形曲线，效果一般
def get_step(loss, net, train_features, train_res):
    a=torch.tensor(0.0, requires_grad=True)
    #weight=net[0].weight.clone().detach().requires_grad_(True)
    #test=torch.mm(train_features, torch.transpose(net[0].weight+a*net[0].weight.grad.data, 0, 1))
    test=torch.sigmoid(torch.mm(train_features, torch.transpose(net[0].weight+a*net[0].weight.grad.data, 0, 1)))
    l=loss(test, train_res)
    gd_1 = torch.autograd.grad(l, a, create_graph=True)
    gd_2 = torch.autograd.grad(gd_1, a)
    #print(float(gd_1[0].data)/float(gd_2[0].data))
    #net[0].weight.data=weight
    return float(gd_1[0].data)/float(gd_2[0].data)
```

```
In [29]: #初始化超参数
net=nn.Sequential( nn.Linear(train3_features.shape[1],1,bias=False), nn.Sigmoid())
net[0].weight.data.fill_(0)
#net[0].bias.data.fill_(0) 不设置bias
```

```
optimizer=torch.optim.SGD(net.parameters(), 1)
loss = nn.BCELoss()
```

In [30]:

```
loss_list_train3, theta_bias_train3, theta_train3=train(net, optimizer, loss, train3_features)

#loss_list_train3, theta_bias_train3, theta_train3=train(net, optimizer, loss, train3_features)

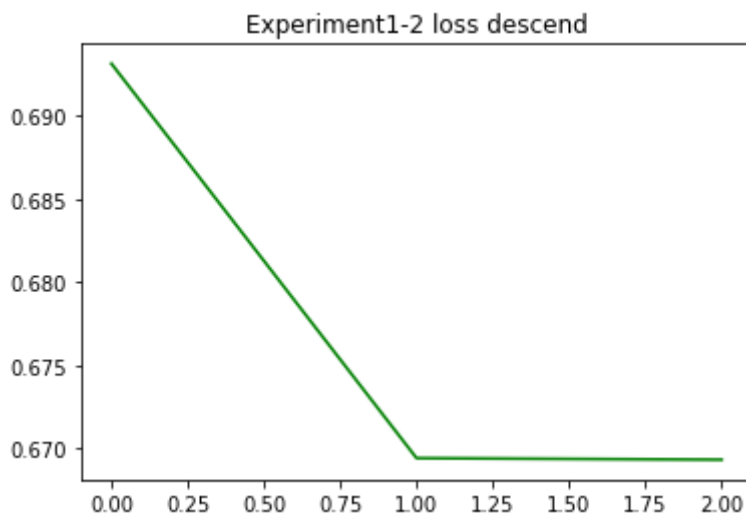
epoches_train3=np.arange(len(loss_list_train3))
fig_train3, ax_train3 = plt.subplots()
fmt='g-'
ax_train3.set_title('Experiment1-2 loss descend')
ax_train3.plot(epoches_train3, loss_list_train3, fmt)

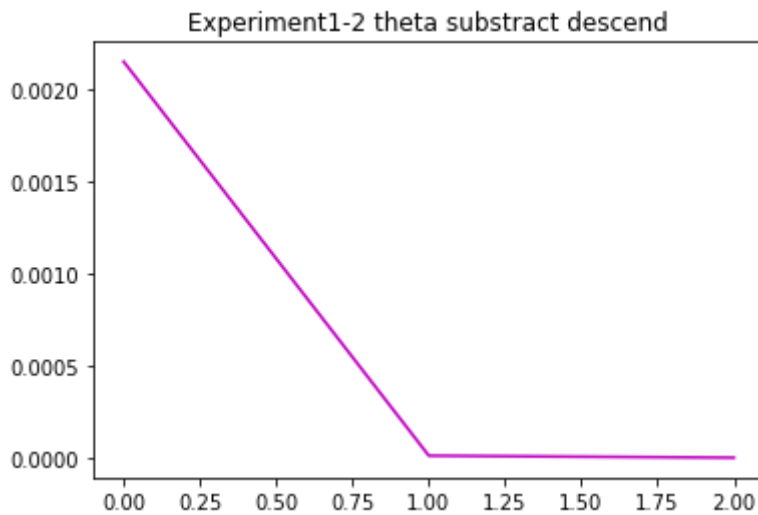
fig_3_train3, ax_3_train3 = plt.subplots()
fmt='m-'
ax_3_train3.set_title('Experiment1-2 theta subtract descend')
ax_3_train3.plot(epoches_train3, theta_bias_train3, fmt)

print("Experiment1-2 theta subtraction ", (torch.norm((net[0].weight-theta_train3).data).sqrt()))
print("Experiment1-2 loss ", squared_loss(net(train3_features), train3_res).data)
print("Experiment1-2 weight ", net[0].weight)
```

C:\Users\Young\AppData\Local\Temp\ipykernel\_7520\1599417071.py:6: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().requires\_grad\_(True), rather than torch.tensor(sourceTensor).

```
theta=torch.tensor(net[0].weight)
Experiment1-2 theta subtraction 1.3289582460288901e-09
Experiment1-2 loss tensor(0.1193)
Experiment1-2 weight Parameter containing:
tensor([[ 0.0443, -0.0224]], requires_grad=True)
```





```
In [27]: net(torch.tensor([20.0, 80.0]))
#录取率只有28%，因此大概率不会录取
```

```
Out[27]: tensor([0.2871], grad_fn=<SigmoidBackward0>)
```

```
In [ ]: #求二阶导数
x = torch.tensor([0., 0, 0], requires_grad=True)
b = torch.tensor([1., 3, 5], requires_grad=True)
A = torch.tensor([[-5, -3, -0.5], [-3, -2, 0], [-0.5, 0, -0.5]])
y = b@x + 0.5*x@A@x

# 计算一阶导数, 因为我们需要继续计算二阶导数, 所以创建并保留计算图
grad = torch.autograd.grad(y, x, retain_graph=True, create_graph=True)
print(grad[0])#grad返回一个元组, 依次包含了[x,b]的梯度, 但是并不会存储在x.grad中
#值得注意的是, grad[0].shape=[3], 这样显然不能参与数组的运算, 必须变为[1,3], 此时可以用
Print = torch.tensor([])
grad=grad[0]
for anygrad in grad:
    print(anygrad)
    Print = torch.cat((Print, torch.autograd.grad(anygrad, x, retain_graph=True)[0].data.view(-1)))

print("Hessian ", Print)
print("inverse of Hessian ", torch.linalg.inv(Print))

print(torch.mm(grad.reshape(1,3), torch.linalg.inv(Print)))
x.grad=torch.mm(grad.reshape(1,3), torch.linalg.inv(Print)).squeeze(0)
```