算法示例

SGD\adagrad\adam\rmsprop

```
import torch
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 初始化变量
adaGrad_x = torch.tensor(-10., requires_grad=True)
sgd_x = torch.tensor(-10., requires_grad=True)
adaGrad_optimizer = torch.optim.Adagrad([adaGrad_x], lr=1)
sgd_optimizer = torch.optim.SGD([sgd_x], lr=1)

# 200次迭代优化
adaGrad_x_record, adaGrad_y_record = [], []
sgd_x_record, sgd_y_record = [], []
for i in range(200):
    # AdaGrad
    adaGrad_y = 1/(1 + torch.exp(adaGrad_x))
    adaGrad_x_record.append(adaGrad_x.detach().item())
    adaGrad_y_record.append(adaGrad_y.detach().item())
    adaGrad_optimizer.zero_grad()
    adaGrad_y.backward()
    adaGrad_optimizer.step()

    # SGD
    sgd_y = 1/(1 + torch.exp(sgd_x))
    sgd_x_record.append(sgd_x.detach().item())
    sgd_y_record.append(sgd_y.detach().item())
    sgd_optimizer.zero_grad()
    sgd_y.backward()
    sgd_optimizer.step()

# y = 1/(1+e^x)
a = torch.linspace(-10, 10, 1000)
b = 1/(1 + torch.exp(a))

# 创建画布
plt.figure(figsize=(12, 4))

# AdaGrad
plt.subplot(1, 2, 1)
plt.plot(a, b)
plt.scatter(adaGrad_x_record, adaGrad_y_record, c='r', alpha=0.5)
plt.title('AdaGrad')
plt.grid()
# SGD
plt.subplot(1, 2, 2)
plt.plot(a, b)
plt.scatter(sgd_x_record, sgd_y_record , c='r', alpha=0.5)
plt.title('SGD')
plt.grid()
# 显示图片
```

```
plt.suptitle('图1', y=0)
plt.show()
```

```
import torch
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus'] = False

adaGrad_x = torch.tensor(-10., requires_grad=True)
sgd_x = torch.tensor(-10., requires_grad=True)
adaGrad_optimizer = torch.optim.Adagrad([adaGrad_x], lr=1)
sgd_optimizer = torch.optim.SGD([sgd_x], lr=1)

# 100次迭代优化
adaGrad_x_record, adaGrad_y_record = [], []
sgd_x_record, sgd_y_record = [], []
for i in range(100):
    # AdaGrad
    adaGrad_y = adaGrad_x ** 2
    adaGrad_x_record.append(adaGrad_x.detach().item())
    adaGrad_y_record.append(adaGrad_y.detach().item())
    adaGrad_optimizer.zero_grad()
    adaGrad_y.backward()
    adaGrad_optimizer.step()

    # SGD
    sgd_y = sgd_x ** 2
    sgd_x_record.append(sgd_x.detach().item())
    sgd_y_record.append(sgd_y.detach().item())
    sgd_optimizer.zero_grad()
    sgd_y.backward()
    sgd_optimizer.step()

# y = x^2
a = torch.linspace(-10, 10, 1000)
b = a ** 2

# 创建画布
plt.figure(figsize=(12, 4))

# AdaGrad
plt.subplot(1, 2, 1)
plt.plot(a, b)
plt.scatter(adaGrad_x_record, adaGrad_y_record, c='r', alpha=0.5)
plt.title('AdaGrad')
plt.grid()
# SGD
plt.subplot(1, 2, 2)
plt.plot(a, b)
plt.scatter(sgd_x_record, sgd_y_record , c='r', alpha=0.5)
plt.title('SGD')
plt.grid()
# 显示图片
plt.suptitle('图2', y=0)
```

```python
plt.show()
```

```python
import torch
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 初始化变量
adaGrad_x = torch.tensor(-20., requires_grad=True)
rmsProp_x = torch.tensor(-20., requires_grad=True)
adaGrad_optimizer = torch.optim.Adagrad([adaGrad_x], lr=1)
rmsProp_optimizer = torch.optim.RMSprop([rmsProp_x], lr=1)

# 200次迭代优化
adaGrad_x_record, adaGrad_y_record = [], []
rmsProp_x_record, rmsProp_y_record = [], []
for i in range(200):
    # AdaGrad
    adaGrad_y = 1/(1 + torch.exp(adaGrad_x))
    adaGrad_x_record.append(adaGrad_x.detach().item())
    adaGrad_y_record.append(adaGrad_y.detach().item())
    adaGrad_optimizer.zero_grad()
    adaGrad_y.backward()
    adaGrad_optimizer.step()

    # RMSProp
    rmsProp_y = 1/(1 + torch.exp(rmsProp_x))
    rmsProp_x_record.append(rmsProp_x.detach().item())
    rmsProp_y_record.append(rmsProp_y.detach().item())
    rmsProp_optimizer.zero_grad()
    rmsProp_y.backward()
    rmsProp_optimizer.step()

# y = 1/(1+e^x)
a = torch.linspace(-20, 20, 1000)
b = 1/(1 + torch.exp(a))

# 创建画布
plt.figure(figsize=(12, 4))

# AdaGrad
plt.subplot(1, 2, 1)
plt.plot(a, b)
plt.scatter(adaGrad_x_record, adaGrad_y_record, c='r', alpha=0.5)
plt.title('AdaGrad')
plt.grid()
# RMSProp
plt.subplot(1, 2, 2)
plt.plot(a, b)
plt.scatter(rmsProp_x_record, rmsProp_y_record , c='r', alpha=0.5)
plt.title('RMSProp')
plt.grid()
# 显示图片
plt.suptitle('图3', y=0)
plt.show()
```

```
import torch
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 初始化变量
rmsProp_x = torch.tensor(-10., requires_grad=True)
adam_x = torch.tensor(-10., requires_grad=True)
rmsProp_optimizer = torch.optim.RMSprop([rmsProp_x], lr=1)
adam_optimizer = torch.optim.Adam([adam_x], lr=1, betas=(0.9, 0.61))

# 40次迭代优化
rmsProp_x_record, rmsProp_y_record = [], []
adam_x_record, adam_y_record = [], []
for i in range(40):
    # RMSProp
    rmsProp_y =  16 * rmsProp_x ** 2 - rmsProp_x ** 3
    rmsProp_x_record.append(rmsProp_x.detach().item())
    rmsProp_y_record.append(rmsProp_y.detach().item())
    rmsProp_optimizer.zero_grad()
    rmsProp_y.backward()
    rmsProp_optimizer.step()

    # Adam
    adam_y =  16 * adam_x ** 2 - adam_x ** 3
    adam_x_record.append(adam_x.detach().item())
    adam_y_record.append(adam_y.detach().item())
    adam_optimizer.zero_grad()
    adam_y.backward()
    adam_optimizer.step()

# y = 16*x^2-x^3
a = torch.linspace(-10., 20., 1000)
b = 16 * a ** 2 - a ** 3

# 创建画布
plt.figure(figsize=(12, 4))

# RMSProp
plt.subplot(1, 2, 1)
plt.plot(a, b)
plt.scatter(rmsProp_x_record, rmsProp_y_record, c='r', alpha=0.5)
plt.title('RMSProp')
plt.grid()
# Adam
plt.subplot(1, 2, 2)
plt.plot(a, b)
plt.scatter(adam_x_record, adam_y_record , c='r', alpha=0.5)
plt.title('Adam')
plt.grid()
# 显示图片
plt.suptitle('图4', y=0)
plt.show()
```

```
 #全部常用算法集合比较

import torch
import torch.utils.data as Data
import torch.nn.functional as F
from torch.autograd import Variable
import matplotlib.pyplot as plt

# 超参数
LR = 0.01
BATCH_SIZE = 32
EPOCH = 12

# 生成假数据
# torch.unsqueeze() 的作用是将一维变二维，torch只能处理二维的数据
x = torch.unsqueeze(torch.linspace(-1, 1, 1000), dim=1)  # x data (tensor),
shape(100, 1)
# 0.2 * torch.rand(x.size())增加噪点
y = x.pow(2) + 0.1 * torch.normal(torch.zeros(*x.size()))

# 定义数据库
dataset = Data.TensorDataset(x, y)

# 定义数据加载器
loader = Data.DataLoader(dataset=dataset, batch_size=BATCH_SIZE, shuffle=True,
num_workers=0)


# 定义pytorch网络
class Net(torch.nn.Module):
    def __init__(self, n_features, n_hidden, n_output):
        super(Net, self).__init__()
        self.hidden = torch.nn.Linear(n_features, n_hidden)
        self.predict = torch.nn.Linear(n_hidden, n_output)

    def forward(self, x):
        x = F.relu(self.hidden(x))
        y = self.predict(x)
        return y

# 定义不同的优化器网络
net_SGD = Net(1, 10, 1)
net_Momentum = Net(1, 10, 1)
net_Adagrad = Net(1, 10, 1)
net_Adadelta = Net(1, 10, 1)
net_RMSprop = Net(1, 10, 1)
net_Adam = Net(1, 10, 1)
net_Adamax = Net(1, 10, 1)
net_AdamW = Net(1, 10, 1)
net_LBFGS = Net(1, 10, 1)

# 选择不同的优化方法
```

```python
opt_SGD = torch.optim.SGD(net_SGD.parameters(), lr=LR)
opt_Momentum = torch.optim.SGD(net_Momentum.parameters(), lr=LR, momentum=0.9)
opt_Adagrad = torch.optim.Adagrad(net_Adagrad.parameters(), lr=LR)
opt_Adadelta = torch.optim.Adadelta(net_Adadelta.parameters(), lr=LR)
opt_RMSprop = torch.optim.RMSprop(net_RMSprop.parameters(), lr=LR, alpha=0.9)
opt_Adam = torch.optim.Adam(net_Adam.parameters(), lr=LR, betas=(0.9, 0.99))
opt_Adamax = torch.optim.Adamax(net_Adamax.parameters(), lr=LR, betas=(0.9,
0.99))
opt_AdamW = torch.optim.AdamW(net_AdamW.parameters(), lr=LR, betas=(0.9, 0.99))
opt_LBFGS = torch.optim.LBFGS(net_LBFGS.parameters(), lr=LR, max_iter=10,
max_eval=10)



nets = [net_SGD, net_Momentum, net_Adagrad, net_Adadelta, net_RMSprop, net_Adam,
net_Adamax, net_AdamW, net_LBFGS]
optimizers = [opt_SGD, opt_Momentum, opt_Adagrad, opt_Adadelta, opt_RMSprop,
opt_Adam, opt_Adamax, opt_AdamW, opt_LBFGS]

# 选择损失函数
loss_func = torch.nn.MSELoss()

# 不同方法的loss
loss_SGD = []
loss_Momentum = []
loss_Adagrad = []
loss_Adadelta = []
loss_RMSprop = []
loss_Adam = []
loss_Adamax = []
loss_AdamW = []
loss_LBFGS = []

# 保存所有loss
losses = [loss_SGD, loss_Momentum, loss_Adagrad, loss_Adadelta, loss_RMSprop,
loss_Adam, loss_Adamax, loss_AdamW, loss_LBFGS]

# 执行训练
for epoch in range(EPOCH):
    for step, (batch_x, batch_y) in enumerate(loader):
        var_x = Variable(batch_x)
        var_y = Variable(batch_y)
        for net, optimizer, loss_history in zip(nets, optimizers, losses):
            if isinstance(optimizer, torch.optim.LBFGS):
                def closure():
                    y_pred = net(var_x)
                    loss = loss_func(y_pred, var_y)
                    optimizer.zero_grad()
                    loss.backward()
                    return loss
                loss = optimizer.step(closure)
            else:
                # 对x进行预测
                prediction = net(var_x)
                # 计算损失
                loss = loss_func(prediction, var_y)
                # 每次迭代清空上一次的梯度
                optimizer.zero_grad()
                # 反向传播
                loss.backward()
                # 更新梯度
                optimizer.step()
            # 保存loss记录
            loss_history.append(loss.data)

# 画图
labels = ['SGD', 'Momentum', 'Adagrad', 'Adadelta', 'RMSprop', 'Adam', 'Adamax',
'AdamW', 'LBFGS']
for i, loss_history in enumerate(losses):
    plt.plot(loss_history, label=labels[i])
plt.legend(loc='best')
plt.xlabel('Steps')
plt.ylabel('Loss')
plt.ylim((0, 0.2))
plt.show()
```