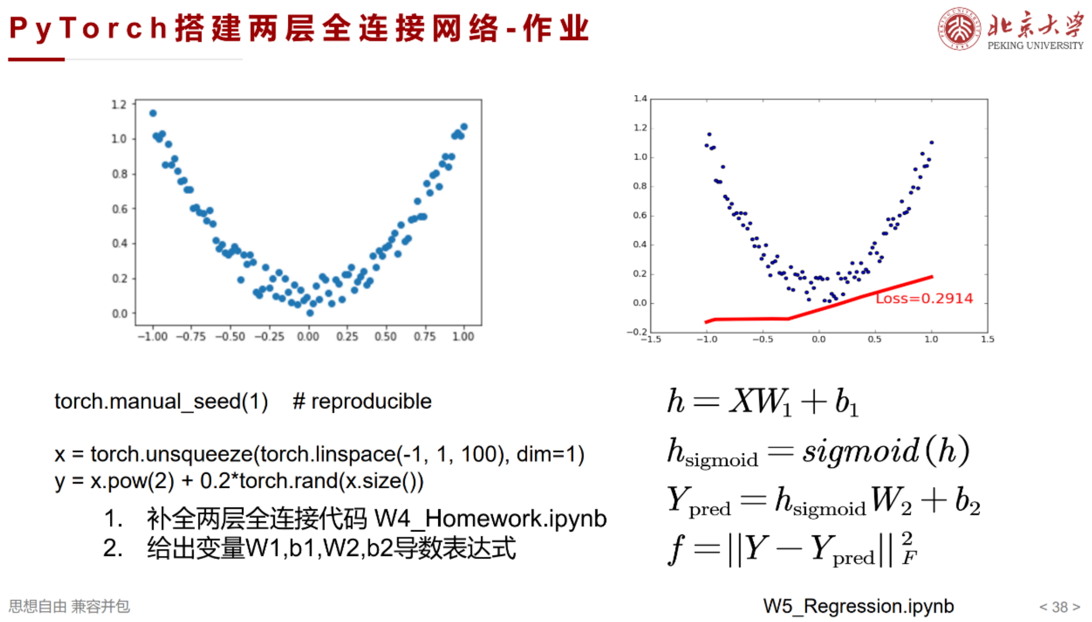
计算视觉第五次作业

姓名：成泽森

学号: 210111565



推导表达式

已知 F-范数的性质:

$$\|A\|_F = \sqrt{tr(A^TA)}$$

则有

$$\begin{aligned} f &= \|Y - Y_{pred}\|_F^2 \\ &= tr\left(\left(Y - Y_{pred}\right)^T\left(Y - Y_{pred}\right)\right) \end{aligned}$$

取微分:

$$\begin{aligned} df &= tr\left(-dY_{pred}^T\left(Y - Y_{pred}\right) + \left(Y - Y_{pred}\right)^T\left(-dY_{pred}\right)\right) \\ &= tr\left(\left[\left(Y - Y_{pred}\right)^T\left(-dY_{pre}\right)\right]^T + \left(Y - Y_{pred}\right)^T\left(-dY_{pred}\right)\right) \end{aligned}$$

由 $tr(A)=tr(A^T)$ 以及 $tr(A+B)=tr(A)+tr(B)$:

$$=tr\left(\left(Y-Y_{pred}\right)^T\left(-dY_{pred}\right)+\ \left(Y-Y_{pred}\right)^T\left(-dY_{pred}\right)\right)$$

$$=tr\left(2*\ \left(Y-Y_{pred}\right)^T\left(-dY_{pred}\right)\right)$$

设 $\sigma=sigmoid$, 且有 $\sigma'(x)=\frac{\exp(x)}{(1+\exp(x))^2}$, 除法为矩阵按位除法。

1. $\frac{\partial f}{\partial W_2}$

$$df=tr(-2*\ \left(Y-\ Y_{pred}\right)^T(\frac{\partial Y_{pred}}{\partial h_{sigmoid}W_2}\odot h_{sigmoid}dW_2)$$

$$=tr\left(-2*\left[\left(Y-Y_{pred}\right)\odot\left(\frac{\partial Y_{pred}}{\partial h_{sigmoid}W_2}\right)\right]^T h_{sigmoid}dW_2\right)$$

易得 $\frac{\partial Y_{pred}}{\partial h_{sigmoid}W_2}=1$:

$$=tr\left(-2*\left(Y-Y_{pred}\right)^T h_{sigmoid}dW_2\right)$$

$$=tr\left(\left[-2*h_{sigmoid}^T\left(Y-Y_{pred}\right)\right]^T dW_2\right)$$

由 $df=tr\left(\left(\frac{\partial f}{\partial W_2}\right)^T dW_2\right)$, 可得:

$$\frac{\partial f}{\partial W_2}=\ -2*h_{sigmoid}^T\left(Y-Y_{pred}\right)$$

2. $\frac{\partial f}{\partial B_2}$

$$df=tr(-2*\ \left(Y-\ Y_{pred}\right)^T(\frac{\partial Y_{pred}}{\partial B_2}\odot dB_2)$$

$$=tr(-2*\ \left(Y-\ Y_{pred}\right)^T dB_2)$$

由 $df=tr\left(\left(\frac{\partial f}{\partial B_2}\right)^T dB_2\right)$, 可得:

$$\frac{\partial f}{\partial B_2}=\ -2*(Y-Y_{pred})$$

3. $\frac{\partial f}{\partial W_1}$

$$df=tr(-2*\ \left(Y-\ Y_{pred}\right)^T(dY_{pred})$$

令 $\sigma = sigmoid$

$$dY_{pred} = d(\sigma(XW_1 + B_1)W_2)$$

$$= [d\sigma(XW_1 + B_1)]W_2$$

$$= \left[\frac{\partial\sigma(XW_1 + B_1)}{\partial(XW_1 + B_1)} \odot \mathrm{XdW_1}\right] W_2$$

$$= [\sigma'(XW_1 + B_1) \odot \mathrm{XdW_1}]W_2$$

则有:

$$df = tr\left(-2 * \left(Y - Y_{pred}\right)^T (\sigma'(XW_1 + B_1) \odot \mathrm{XdW_1})W_2\right)$$

$$= tr\left(-2 * W_2\left(Y - Y_{pred}\right)^T (\sigma'(XW_1 + B_1) \odot \mathrm{XdW_1})\right)$$

$$= tr(-2 * \left(\left(Y - Y_{pred}\right)W_2^T\right)^T (\sigma'(XW_1 + B_1) \odot \mathrm{XdW_1})$$

$$= tr(-2 * \left[\left(Y - Y_{pred}\right)W_2^T \odot \sigma'(XW_1 + B_1)\right]^T XdW_1$$

$$= tr\left(\left[-2 * X^T\left[\left(Y - Y_{pred}\right)W_2^T \odot \sigma'(XW_1 + B_1)\right]\right]^T dW_1\right)$$

由 $df = tr\left(\left(\frac{\partial f}{\partial W_1}\right)^T dW_1\right)$, 可得:

$$\frac{\partial f}{\partial W_1} = -2 * X^T\left[\left(Y - Y_{pred}\right)W_2^T \odot \sigma'(XW_1 + B_1)\right]$$

4. $\frac{\partial f}{\partial B_1}$

$$df = tr(-2 * \left(Y - Y_{pred}\right)^T (dY_{pred})$$

令 $\sigma = sigmoid$

$$dY_{pred} = d(\sigma(XW_1 + B_1)W_2)$$

$$= [d\sigma(XW_1 + B_1)]W_2$$

$$= \left[\frac{\partial\sigma(XW_1 + B_1)}{\partial(XW_1 + B_1)} \odot dB_1\right] W_2$$

$$= [\sigma'(XW_1 + B_1) \odot dB_1]W_2$$

则有:

$$= tr\left(-2 * \left(Y - Y_{pred}\right)^T [\sigma'(XW_1 + B_1) \odot dB_1]W_2\right)$$

$$= tr\left(-2 * W_2\left(Y - Y_{pred}\right)^T[\sigma'(XW_1 + B_1)\odot dB_1]\right)$$

$$= tr\left(-2 * \left(\left(Y - Y_{pred}\right)W_2^T\right)^T \ [\sigma'(XW_1 + B_1)\odot dB_1]\right)$$

$$= tr\left(-2 * \left(\left(Y - Y_{pred}\right)W_2^T \odot \sigma'(XW_1 + B_1)\right)^T dB_1\right)$$

由 $df = tr\left(\left(\frac{\partial f}{\partial \mathrm{B}_1}\right)^T d\mathrm{B}_1\right)$, 可得:

$$\frac{\partial f}{\partial \mathrm{B}_1} = -2 * \left(\left(Y - Y_{pred}\right)W_2^T \odot \sigma'(XW_1 + B_1)\right)^T$$

代码补全

搭建两层含有bias的全连接网络，隐藏层输出个数为20，激活函数都用sigmoid()

```python
class Net(nn.Module):
    def __init__(self, n_feature, n_hidden, n_output):
        super(Net, self).__init__()
        self.linear1 = nn.Linear(n_feature, n_hidden, bias=True)

        self.activation = nn.Sigmoid()

        self.linear2 = nn.Linear(n_hidden, n_output, bias=True)

    def forward(self, x):
        x = self.linear1(x)
        x = self.activation(x)
        x = self.linear2(x)
        return x
```

✓ 0.2s Python

代码验证

```python
# XXX: init a new network to avoid gradient from steps before.
net = Net(n_feature=1, n_hidden=20, n_output=1)     # define the network

# XXX: net forward (The pytorch linear implementation is: X @ W.t() + B):
# H  = X @ W1.t() + B1
# Hs = sigmoid(H)
# Yp = Hs @ W2.t() + B2
# don't match with ppt expression format.
W1 = net.linear1.weight.t()
B1 = net.linear1.bias
W2 = net.linear2.weight.t()
B2 = net.linear2.bias

# Data & Label defenition
x = torch.randn((1, 1)).float()
y = x.pow(2) + 0.2*torch.rand(x.size())

prediction = net(x)

# 1. MSE loss (sum wise)
loss_func = torch.nn.MSELoss(reduction='sum')  # this is for regression mean squared loss
loss = loss_func(prediction, y)
# 2. squared F-norm of (y - prediction)
f = torch.trace((y - prediction).t() @ (y - prediction))
# These two implementation is equal.
print('===================')
print(f'Sum wise MSE loss: \n{loss} \nsquared F-norm of (y - prediction): \n{f}')
print(f'match results: {torch.allclose(loss, f)}')

loss.backward()

# W2 grad
W2_grad = -2 * F.sigmoid(x @ W1 + B1).t() @ (y - prediction)
W2_autograd = net.linear2.weight.grad.t()
print('===================')
print(f'W2 torch gradient: \n{W2_autograd} \nW2 matrix partial derivative results: \n{W2_grad}')
print(f'match results: {torch.allclose(W2_autograd, W2_grad)}')
```

```python
# B2 grad
B2_grad = -2 * (y - prediction).t()
B2_autograd = net.linear2.bias.grad
print('==================')
print(f'B2 torch gradient: \n{B2_autograd} \nB2 matrix partial derivative results: \n{B2_grad}')
print(f'match results: {torch.allclose(B2_autograd, B2_grad)}')

# W1 grad
W1_grad = -2 * x.t() @ ((y - prediction) @ W2.t()) * (torch.exp(x @ W1 + B1) / torch.square(1 + torch.exp(x @ W1 + B1)))
W1_autograd = net.linear1.weight.grad.t()
print('==================')
print(f'W1 torch gradient: \n{W1_autograd} \W1 matrix partial derivative results: \n{W1_grad}')
print(f'match results: {torch.allclose(W1_autograd, W1_grad)}')

# B1 grad
B1_grad = -2 * ((y - prediction) @ W2.t()) * (torch.exp(x @ W1 + B1) / torch.square(1 + torch.exp(x @ W1 + B1)))
B1_autograd = net.linear1.bias.grad
print('==================')
print(f'B1 torch gradient: \n{B1_autograd} \W1 matrix partial derivative results: \n{B1_grad}')
print(f'match results: {torch.allclose(B1_autograd, B1_grad)}')
```

结果:

```
===================
Sum wise MSE loss:
1.8570191860198975
squared F-norm of (y - prediction):
1.8570191860198975
match results: True
===================
W2 torch gradient:
tensor([[-1.3207],
        [-1.7060],
        [-1.5753],
        [-1.8741],
        [-1.7241],
        [-1.5976],
        [-1.7935],
        [-1.1395],
        [-1.8699],
        [-1.6506],
        [-0.9117],
        [-1.9007],
        [-1.6432],
        [-1.3424],
        [-1.9508],
        [-0.7225],
        [-1.7838],
        [-2.0284],
        [-0.5232],
        [-0.7805]])
W2 matrix partial derivative results:
tensor([[-1.3207],
        [-1.7060],
        [-1.5753],
        [-1.8741],
        [-1.7241],
        [-1.5976],
        [-1.7935],
        [-1.1395],
        [-1.8699],
        [-1.6506],
        [-0.9117],
        [-1.9007],
        [-1.6432],
        [-1.3424],
        [-1.9508],
        [-0.7225],
        [-1.7838],
```

```
        [-1.8741],
        [-1.7241],
        [-1.5976],
        [-1.7935],
        [-1.1395],
        [-1.8699],
        [-1.6506],
        [-0.9117],
        [-1.9007],
        [-1.6432],
        [-1.3424],
        [-1.9508],
        [-0.7225],
        [-1.7838],
        [-2.0284],
        [-0.5232],
        [-0.7805]], grad_fn=<MmBackward>)
match results: True
===================
B2 torch gradient:
tensor([-2.7254])
B2 matrix partial derivative results:
tensor([[-2.7254]], grad_fn=<MulBackward0>)
match results: True
===================
W1 torch gradient:
tensor([[-0.0098,  0.0124, -0.1540,  0.0334, -0.1175,  0.1351,  0.1413,  0.0148,
          0.0901,  0.0460, -0.0166,  0.1209, -0.1087,  0.1002, -0.1031, -0.0421,
          0.0280,  0.1100, -0.0170,  0.0754]])
W1 matrix partial derivative results:
tensor([[-0.0098,  0.0124, -0.1540,  0.0334, -0.1175,  0.1351,  0.1413,  0.0148,
          0.0901,  0.0460, -0.0166,  0.1209, -0.1087,  0.1002, -0.1031, -0.0421,
          0.0280,  0.1100, -0.0170,  0.0754]], grad_fn=<MulBackward0>)
match results: True
===================
B1 torch gradient:
tensor([-0.0090,  0.0115, -0.1421,  0.0308, -0.1084,  0.1247,  0.1303,  0.0136,
         0.0831,  0.0424, -0.0153,  0.1116, -0.1003,  0.0924, -0.0951, -0.0388,
         0.0258,  0.1015, -0.0156,  0.0696])
W1 matrix partial derivative results:
tensor([[-0.0090,  0.0115, -0.1421,  0.0308, -0.1084,  0.1247,  0.1303,  0.0136,
          0.0831,  0.0424, -0.0153,  0.1116, -0.1003,  0.0924, -0.0951, -0.0388,
          0.0258,  0.1015, -0.0156,  0.0696]], grad_fn=<MulBackward0>)
match results: True
```