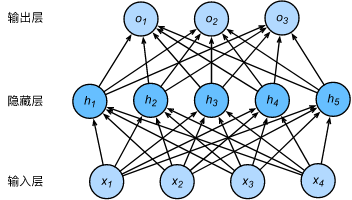
**多层感知器PyTorch实现一**

这次的文章我们来聊一聊多层感知器模型，下图展示了一个多层感知机的神经网络图，它含有一个隐藏层，该层中有5个隐藏单元，h1,h2,h3,h4,h5。



可以发现，多层感知器（假设只有一层隐藏层）比之前实现的softmax回归模型多了一层隐藏层（上图是多一层），那么在实现的时候我们就要多实现一层隐藏层即可。

Softmax回归模型其实是一种有偏置项的线性变换，而接下来我们实现的多层感知机（MLP,Multilayer Perceptron）是一种非线性变化。

我们来看这样的一个事实，现在假设我们有一n个样本，每一个样本有d个特征的小批量数据X：



对于具有h个隐藏单元（上述图片的中的隐藏单元数为5）单隐藏层多层感知机来说，假设：



表示隐藏层的输出。

将输入层到隐藏层的权重和偏置分别记录为：



将输入层到隐藏层的权重和偏置分别记录为：



单隐藏层的多层感知机的输出记录为：



则有：



将第一个公式带入第二个，替换则有：

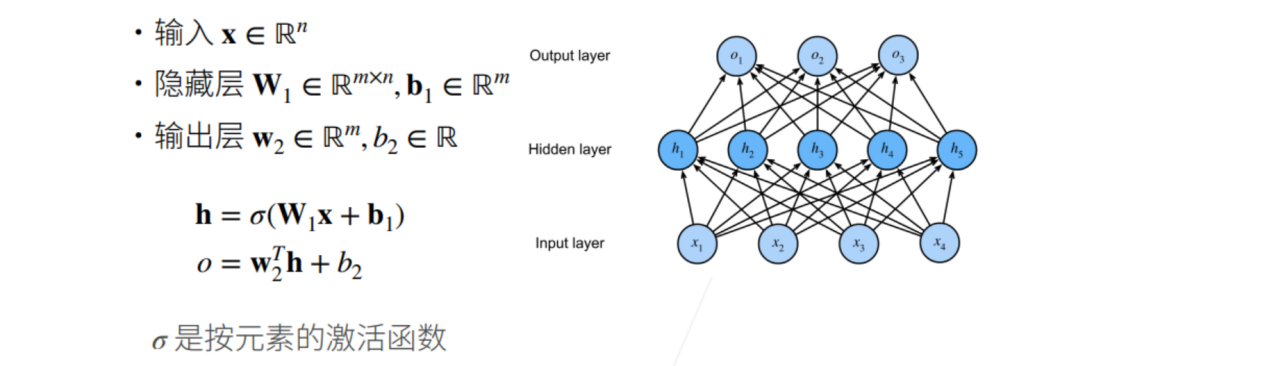


这样看来，一个多层的感知机模型在效果上是与单层的模型的效果一致。

当然了这不是们想要的，为了发挥多层的作用效果，我们引入了激活函（activation function），使用激活函数就不在会出现多层感知器模型退化为单层模型的情况：



在激活函数的加持下，隐藏层的输出将变得不再线性，表达能力将变强。上述阐述使用图进行解释如下：



**常见的激活函数**

激活函数（activation function）通过计算加权和并加上偏置来确定神经元是否应该被激活，⼤多数激活函数都是⾮线性的，非线性激活函数加强了网络的表示能力，解决线性模型无法解决的问题。激活函数在深度学习中是不可缺少的，常见的激活函数有ReLU()，Sigmoid()，Tanh()。

**ReLu激活函数：**



该函数图像可以使用以下代码来进行输出：

from IPython import display

from matplotlib import pyplot as plt

def set\_figsize(figsize=(3.5, 2.5)):

display.display\_svg()

plt.rcParams['figure.figsize'] = figsize

def set\_axes(axes, xlabel, ylabel, xlim, ylim, xscale, yscale, legend):

axes.set\_xlabel(xlabel)

axes.set\_ylabel(ylabel)

axes.set\_xscale(xscale)

axes.set\_yscale(yscale)

axes.set\_xlim(xlim)

axes.set\_ylim(ylim)

if legend:

axes.legend(legend)

axes.grid()

# 画图

def plot(X, Y=None, xlabel=None, ylabel=None, legend=None, xlim=None,

ylim=None, xscale='linear', yscale='linear',

fmts=('-', 'm--', 'g-.', 'r:'), figsize=(3.5, 2.5), axes=None):

"""Plot data points.

Defined in :numref:`sec\_calculus`"""

if legend is None:

legend = []

set\_figsize(figsize)

axes = axes if axes else plt.gca()

# 如果X有一个轴（ndim=1），返回True

def has\_one\_axis(X):

return (hasattr(X, "ndim") and X.ndim == 1 or isinstance(X, list)

and not hasattr(X[0], "\_\_len\_\_"))

if has\_one\_axis(X):

X = [X]

if Y is None:

X, Y = [[]] \* len(X), X

elif has\_one\_axis(Y):

Y = [Y]

if len(X) != len(Y):

X = X \* len(Y)

axes.cla()

for x, y, fmt in zip(X, Y, fmts):

if len(x):

axes.plot(x, y, fmt)

else:

axes.plot(y, fmt)

set\_axes(axes, xlabel, ylabel, xlim, ylim, xscale, yscale, legend)

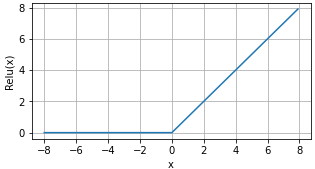
# 调用

x = torch.arange(-8.0, 8.0, 0.1, requires\_grad=True)

y = torch.relu(x)

plot(x.detach(), y.detach(), 'x', 'Relu(x)', figsize=(5, 2.5))

# 三行代码的就可以输出图像：



可以看出：

1. 当输入为负的时候，Relu的输出为0，其导数为0；

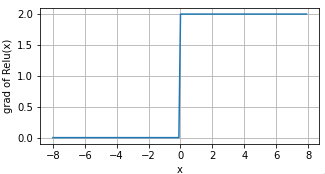
（2）当输入为正的时候，Relu的输出为正，其导数为1；

这里顺便画出其导数的图像：

y.backward(torch.ones\_like(x),retain\_graph = True)

plot(x.detach(), x.grad, 'x', 'grad of Relu(x)', figsize=(5, 2.5))

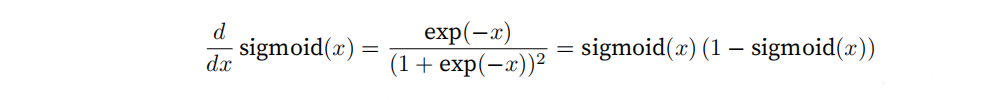
图像如下：



**Sigmoid激活函数：**



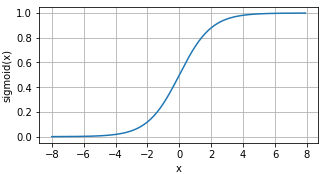
其导数为：



相应的画出sigmoid(x)的图像如下：

y = torch.sigmoid(x)

plot(x.detach(), y.detach(), 'x', 'sigmoid(x)', figsize=(5, 2.5))



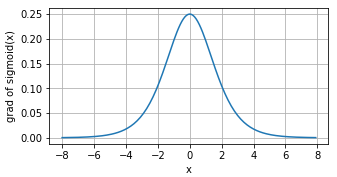
其导数图像如下：

# 清除之前的梯度

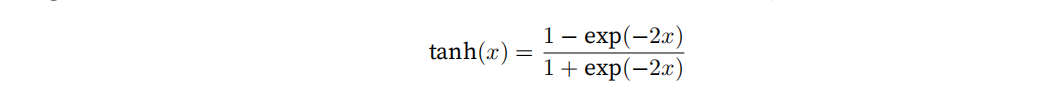
x.grad.data.zero\_()

y.backward(torch.ones\_like(x),retain\_graph=True)

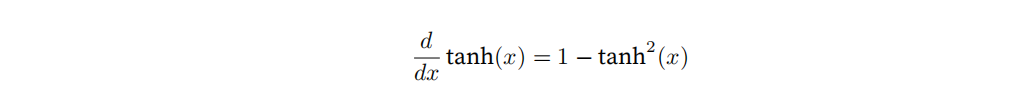
plot(x.detach(), x.grad, 'x', 'grad of sigmoid(x)', figsize=(5, 2.5))



**Tanh激活函数：**

****

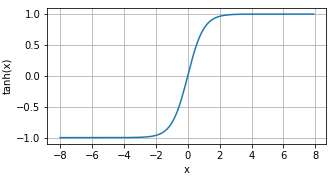
其导数形式为：



这里作出相应的图像：

y = torch.tanh(x)

plot(x.detach(), y.detach(), 'x', 'tanh(x)', figsize=(5, 2.5))



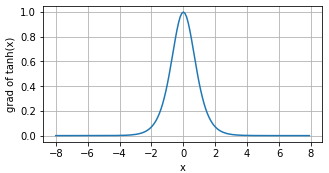
其导数图像：

# 清除之前的梯度

x.grad.data.zero\_()

y.backward(torch.ones\_like(x),retain\_graph=True)

plot(x.detach(), x.grad, 'x', 'grad of tanh(x)', figsize=(5, 2.5))



大家可以研究一下以下几个问题：

（1）不同的激活函数使用的场景。

1. 激活函数的优点和缺点。

**接下来来实现多层感知机算法（假设隐藏层只有一层）**。

1. **加载数据**

这里我们依然选择FashionMNIST数据集进行试验：

import os

import sys

print(sys.path)

import torch

import torchvision

from matplotlib import pyplot as plt

from torch.utils import data

from torchvision import transforms

root\_path = '/home/aipf/work/鲁班7号/'

def load\_data\_fashion\_mnist(batch\_size, resize=None):

root = "/home/aipf/work/鲁班7号/FashionMNIST/"

trans = [transforms.ToTensor()]

if resize:

trans.insert(0, transforms.Resize(resize))

trans = transforms.Compose(trans)

fmnist\_train = torchvision.datasets.FashionMNIST(root=root, train=True, transform=trans, download=True)

fmnist\_test = torchvision.datasets.FashionMNIST(root=root, train=False, transform=trans, download=True)

return (data.DataLoader(fmnist\_train, batch\_size, shuffle=True),

data.DataLoader(fmnist\_test, batch\_size, shuffle=False))

train\_data,test\_data = load\_data\_fashion\_mnist(256)

这里的代码与之前的一样，这里就不再进行解释了。

1. **初始化模型参数**

回想一下，Fashion-MNIST中的每个图像由28×28 = 784个灰度像素值组成，即我们使用784的打平的像素值来表示一个图像，所有图像共分为10个类别。忽略像素之间的空间结构，我们可以将每个图像视为具有784个输⼊特征和10个类的简单类数据集。我们这里设置⼀个具有单隐藏层的多层感知机，它包含256个隐藏单元。

import torch.nn as nn

num\_inputs, num\_outputs, num\_hiddens = 784, 10, 256

W1 = nn.Parameter(torch.randn(num\_inputs, num\_hiddens, requires\_grad=True) \* 0.01)

b1 = nn.Parameter(torch.zeros(num\_hiddens, requires\_grad=True))

W2 = nn.Parameter(torch.randn(num\_hiddens, num\_outputs, requires\_grad=True) \* 0.01)

b2 = nn.Parameter(torch.zeros(num\_outputs, requires\_grad=True))

params = [W1, b1, W2, b2]

这里对权重参数使用torch.randn()进行初始化，对偏置参数使用torch.zeros()进行初始化。

1. **激活函数**

在torch中有对ReLu函数的实现，这里我们先手动实现：

def my\_relu(X):

# 初始化与X形状一致的张量tensor a

a = torch.zeros\_like(X)

return torch.max(X, a)

1. **模型实现**

这里定义一下模型，即多层感知器算法的数学公式部分：

def net(X):

# 更改形状，将X转换为自适应的批量大小，长度为num\_imputs的向量

X = X.reshape((-1, num\_inputs))

# 这⾥“@”代表矩阵乘法，注意满足矩阵乘法的规则，W1维度(784,256),b1为256

H = my\_relu(X@W1 + b1)

# print("H的维度：",H.shape)

return (H@W2 + b2)

1. **损失函数定义**

这里依然使用交叉熵损失函数，这里之间使用高级API来实现了：

loss = nn.CrossEntropyLoss(reduction='none')

num\_epochs, lr = 10, 0.1

optimizer = torch.optim.SGD(params, lr=lr)

print(type(optimizer),'\n')

print(optimizer,'\n')

if isinstance(optimizer, torch.optim.Optimizer):

print('Yes')

# 输出：

<class 'torch.optim.sgd.SGD'>

SGD (

Parameter Group 0

dampening: 0

lr: 0.1

momentum: 0   
nesterov: False

weight\_decay: 0

)

Yes

可以看出其中的相关参数情况。

1. **模型训练和优化器**

class Accumulator:

def \_\_init\_\_(self, n):

self.data = [0.0] \* n

# 进行累加

def add(self, \*args):

self.data = [a + float(b) for a, b in zip(self.data, args)]

def reset(self):

self.data = [0.0] \* len(self.data)

def \_\_getitem\_\_(self, idx):

return self.data[idx]

def accuracy(y\_hat,y):

if len(y\_hat.shape)>1 and y\_hat.shape[1]>1:

# 寻找哪一项是最大的

y\_hat = y\_hat.argmax(axis =1)

cmp = y\_hat.type(y.dtype) == y

# print('CMP:',cmp)

return float(cmp.type(y.dtype).sum())

def evaluate\_accuracy(net, data\_iter):

# 将模型设置为评估模式

if isinstance(net, torch.nn.Module):

net.eval()

# 预测正确的数量和预测数量

metric = Accumulator(2) # No. of correct predictions, no. of predictions

with torch.no\_grad():

for X, y in data\_iter:

metric.add(accuracy(net(X), y), y.numel())

# 精度

return metric[0] / metric[1]

def train(net, train\_data, loss, optimizer):

# 将模型设置为训练模式

if isinstance(net, torch.nn.Module):

net.train()

# 累加训练损失值、累加训练精度，统计样本数

# 初始化

metric = Accumulator(3)

for X, y in train\_data:

# 计算梯度和更新参数

y\_hat = net(X)

# 样本损失

l = loss(y\_hat, y)

if isinstance(optimizer, torch.optim.Optimizer):

# 梯度清零，即将优化器的梯度清零，避免梯度累加的影响

optimizer.zero\_grad()

# 利用反向传播算法计算损失函数关于模型参数的梯度。

l.mean().backward()

# l.sum().backward()

# 利用优化器对模型参数进行更新，以最小化损失函数。

optimizer.step()

else:

l.sum().backward()

optimizer(X.shape[0])

metric.add(float(l.sum()), accuracy(y\_hat, y), y.numel())

# 返回训练损失和训练精度

return metric[0] / metric[2], metric[1] / metric[2]

for epoch in range(num\_epochs):

train\_loss, train\_acc = train(net, train\_data, loss, optimizer)

test\_acc = evaluate\_accuracy(net, test\_data)

print("epoch", epoch + 1, ', train\_loss', "{:.3f}".format(train\_loss), ', train\_acc', "{:.3f}".format(train\_acc), ', test\_acc', "{:.3f}".format(test\_acc))

# 输出如下

epoch 1 , train\_loss 1.051 , train\_acc 0.637 , test\_acc 0.743

epoch 2 , train\_loss 0.603 , train\_acc 0.789 , test\_acc 0.809

epoch 3 , train\_loss 0.521 , train\_acc 0.818 , test\_acc 0.815

epoch 4 , train\_loss 0.482 , train\_acc 0.830 , test\_acc 0.825

epoch 5 , train\_loss 0.450 , train\_acc 0.842 , test\_acc 0.791

epoch 6 , train\_loss 0.435 , train\_acc 0.847 , test\_acc 0.820

epoch 7 , train\_loss 0.416 , train\_acc 0.853 , test\_acc 0.844

epoch 8 , train\_loss 0.402 , train\_acc 0.859 , test\_acc 0.842

epoch 9 , train\_loss 0.389 , train\_acc 0.862 , test\_acc 0.847

epoch 10 , train\_loss 0.379 , train\_acc 0.866 , test\_acc 0.852

在10个epochs之后，训练的精度可以达到85%，还是非常不错的。

1. **预测和绘图**

def get\_fmnist\_labels(labels):

# 获取对应的标签

text\_labels = ['t-shirt', 'trouser', 'pullover', 'dress', 'coat', 'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']

return [text\_labels[int(i)] for i in labels]

from matplotlib import pyplot as plt

def display\_images(imgs, num\_rows, num\_cols, scale=3,titles=None,):

# 设置图像画图大小

figsize = (num\_cols \* scale, num\_rows \* scale)

\_, axes = plt.subplots(num\_rows, num\_cols, figsize=figsize)

axes = axes.flatten()

for i, (ax, img) in enumerate(zip(axes, imgs)):

if torch.is\_tensor(img):

# 图⽚张量

ax.imshow(img.numpy())

else:

# PIL图⽚

ax.imshow(img)

ax.axes.get\_xaxis().set\_visible(True)

ax.axes.get\_yaxis().set\_visible(True)

if titles:

ax.set\_title(titles[i])

# 显示图片

plt.show()

return axes

# 预测

def predict(net, test\_data, n=6):

for X, y in test\_data:

#在测试的数据集中拿出一个样本

break

# 取出真实标签

trues = get\_fmnist\_labels(y)

# 取出预测标签

preds = get\_fmnist\_labels(net(X).argmax(axis=1))

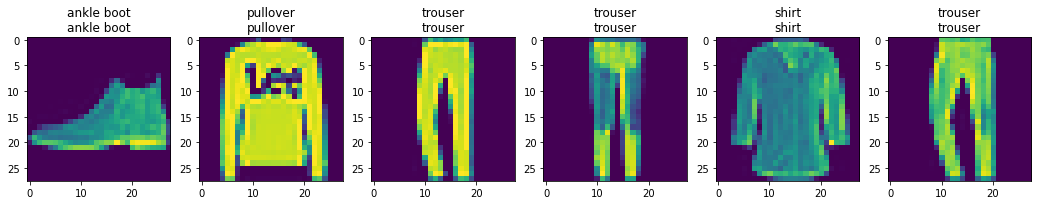
titles = [true +'\n' + pred for true, pred in zip(trues, preds)]

display\_images(

X[0:n].reshape((n, 28, 28)), 1, n, titles=titles[0:n])

predict(net, test\_data)

# 输出图像如下所示



这样就完成了基于PyTorch一步一步实现了多层感知器算法模型，我们下次的文章来使用PyTorch中的APi来MLP模型。

1. **参考文献**

《动手学深度学习》- 李沐

# 问题解答汇总

http://www.dtmao.cc/java/74093.html

https://blog.csdn.net/weixin\_44179269/article/details/124573992

<https://blog.csdn.net/qq_44665283/article/details/128336465>

<https://zhuanlan.zhihu.com/p/479135095>

[https://blog.csdn.net/h\_qiu/article/details/129203115?spm=1001.2101.3001.6650.14&utm\_medium=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7ERate-14-129203115-blog-124174526.235%5Ev27%5Epc\_relevant\_3mothn\_strategy\_and\_data\_recovery&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7ERate-14-129203115-blog-124174526.235%5Ev27%5Epc\_relevant\_3mothn\_strategy\_and\_data\_recovery&utm\_relevant\_index=18](https://blog.csdn.net/h_qiu/article/details/129203115?spm=1001.2101.3001.6650.14&utm_medium=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromBaidu~Rate-14-129203115-blog-124174526.235^v27^pc_relevant_3mothn_strategy_and_data_recovery&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromBaidu~Rate-14-129203115-blog-124174526.235^v27^pc_relevant_3mothn_strategy_and_data_recovery&utm_relevant_index=18)

[https://blog.csdn.net/qq\_42012782/article/details/121625446?spm=1001.2101.3001.6650.5&utm\_medium=distribute.pc\_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-5-121625446-blog-123478906.235%5Ev27%5Epc\_relevant\_3mothn\_strategy\_and\_data\_recovery&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-5-121625446-blog-123478906.235%5Ev27%5Epc\_relevant\_3mothn\_strategy\_and\_data\_recovery&utm\_relevant\_index=10](https://blog.csdn.net/qq_42012782/article/details/121625446?spm=1001.2101.3001.6650.5&utm_medium=distribute.pc_relevant.none-task-blog-2~default~CTRLIST~Rate-5-121625446-blog-123478906.235^v27^pc_relevant_3mothn_strategy_and_data_recovery&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~CTRLIST~Rate-5-121625446-blog-123478906.235^v27^pc_relevant_3mothn_strategy_and_data_recovery&utm_relevant_index=10)

<https://blog.csdn.net/weixin_42067873/article/details/123478906?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_baidulandingword~default-4-123478906-blog-127463720.235>^v27^pc\_relevant\_3mothn\_strategy\_and\_data\_recovery&spm=1001.2101.3001.4242.3&utm\_relevant\_index=7

[https://blog.csdn.net/qq\_45956730/article/details/127463720?spm=1001.2101.3001.6650.18&utm\_medium=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7ERate-18-127463720-blog-124174526.235%5Ev27%5Epc\_relevant\_3mothn\_strategy\_and\_data\_recovery&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7ERate-18-127463720-blog-124174526.235%5Ev27%5Epc\_relevant\_3mothn\_strategy\_and\_data\_recovery&utm\_relevant\_index=22](https://blog.csdn.net/qq_45956730/article/details/127463720?spm=1001.2101.3001.6650.18&utm_medium=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromBaidu~Rate-18-127463720-blog-124174526.235^v27^pc_relevant_3mothn_strategy_and_data_recovery&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromBaidu~Rate-18-127463720-blog-124174526.235^v27^pc_relevant_3mothn_strategy_and_data_recovery&utm_relevant_index=22)

<https://blog.csdn.net/qq_39906884/article/details/124174526>