深度学习框架—Pytorch

官网：<https://pytorch.org/>

参考：https://morvanzhou.github.io/tutorials/machine-learning/torch/

**一、介绍**

Pytorch是Facebook 的 AI 研究团队发布了一个 Python 工具包,是Python优先的深度学习框架。作为 numpy 的替代品;使用强大的 GPU 能力,提供最大的灵活性和速度。

**二、安装**

pip install torch==你需要的版本

pip install torch torchvision==你需要的版本

**三、Tensor**

张量(Tensor)类似于NumPy的ndarray，另外还有Tensors也可用于GPU以加速计算。

Example:

from \_\_future\_\_ import print\_function

import torch

#1、构造一个未初始化的5x3矩阵

x = torch.empty(5,3)

print(x)

#2、构造一个随机初始化的矩阵：

x = torch.rand(5,3)

print(3)

#3、构造一个填充的零和dtype矩阵

x = torch.zeros(5,3,dtype=torch.long)

print (x)

# 4、torch 转为numpy

tensor2array = torch\_data.numpy()

print(tensor2array)

#5、运算符

data = [[5, 7], [1, 2]]

tensor = torch.FloatTensor(data) # 转为32位浮点数，torch接受的都是Tensor的形式，所以运算前先转化为Tensor

print(

'\n numpy', np.matmul(data, data),

'\n torch', torch.mm(tensor, tensor) # torch.dot()是点乘

)

**四、Variable**

在 Torch 中的 Variable 就是一个存放会变化的值的地理位置. 里面的值会不停的变化(变化的值是Torch的Tensor). 如果用一个 Variable 进行计算, 那返回的也是一个同类型的 Variable.举例：就像一个裝鸡蛋的篮子, 鸡蛋数会不停变动.里面的鸡蛋是 Torch 的 Tensor . （参考自https://morvanzhou.github.io/tutorials/machine-learning/torch/2-02-variable/）

Example:

#1、variable

import torch

from torch.autograd import Variable # torch 中 Variable 模块

# 先生鸡蛋

tensor = torch.FloatTensor([[1,2],[3,4]])

# 把鸡蛋放到篮子里, requires\_grad是参不参与误差反向传播, 要不要计算梯度

variable = Variable(tensor, requires\_grad=True)

print(tensor)

#2、对比tensor的计算和variable的计算.

t\_out = torch.mean(tensor\*tensor) # x^2

v\_out = torch.mean(variable\*variable) # x^2

print(t\_out)

print(v\_out) # 7.5

Variable 计算时, 它在背景幕布后面一步步默默地搭建着一个庞大的系统, 叫做计算图, computational graph.。图将所有的计算步骤 (节点) 都连接起来, 最后进行误差反向传递的时候, 一次性将所有 variable 里面的修改幅度 (梯度) 都计算出来。

v\_out = torch.mean(variable\*variable) 就是在计算图中添加的一个计算步骤, 图经常应用在计算误差反向传递。

Example

：

v\_out.backward() # 模拟 v\_out 的误差反向传递

# 下面两步看不懂没关系, 只要知道 Variable 是计算图的一部分, 可以用来传递误差就好.

# v\_out = 1/4 \* sum(variable\*variable) 这是计算图中的 v\_out 计算步骤

# 针对于 v\_out 的梯度就是, d(v\_out)/d(variable) = 1/4\*2\*variable = variable/2

print(variable.grad) # 初始 Variable 的梯度

'''

0.5000 1.0000

1.5000 2.0000

'''

**五、Torch 中的激励函数**

常用的激励函数：relu, sigmoid, tanh, softplus.

Example：

import torch

import torch.nn.functional as F # 激励函数都在这

from torch.autograd import Variable

# 做一些假数据来观看图像

x = torch.linspace(-5, 5, 200) # x data (tensor), shape=(100, 1)

x = Variable(x)

import torch

import torch.nn.functional as F # 激励函数都在这

from torch.autograd import Variable

# 几种常用的 激励函数

y\_relu = F.relu(x).data.numpy()

y\_sigmoid = F.sigmoid(x).data.numpy()

y\_tanh = F.tanh(x).data.numpy()

y\_softplus = F.softplus(x).data.numpy()

# y\_softmax = F.softmax(x) softmax 比较特殊, 不能直接显示, 不过他是关于概率的, 用于分类

import matplotlib.pyplot as plt # python 的可视化模块

plt.figure(1, figsize=(8, 6))

plt.subplot(221)

plt.plot(x\_np, y\_relu, c='red', label='relu')

plt.ylim((-1, 5))

plt.legend(loc='best')

plt.subplot(222)

plt.plot(x\_np, y\_sigmoid, c='red', label='sigmoid')

plt.ylim((-0.2, 1.2))

plt.legend(loc='best')

plt.subplot(223)

plt.plot(x\_np, y\_tanh, c='red', label='tanh')

plt.ylim((-1.2, 1.2))

plt.legend(loc='best')

plt.subplot(224)

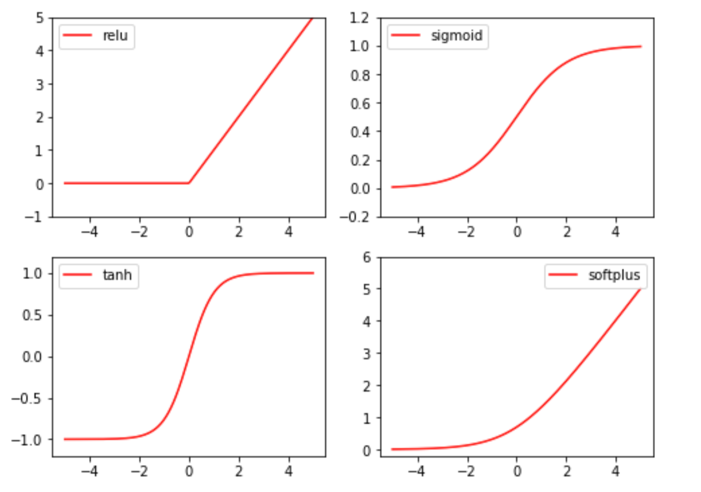
plt.plot(x\_np, y\_softplus, c='red', label='softplus')

plt.ylim((-0.2, 6))

plt.legend(loc='best')

plt.show()

结果：



**六、神经网络搭建—以回归为例**

1、建立数据集

#设置一个一元二次函数: y = a \* x^2 + b,给 y 数据加上一点噪声来更加真实的展示它.

Example：

x = torch.unsqueeze(torch.linspace(-1,1,100),dim=1)#

y = x.pow(2) + 0.2\*torch.rand(x.size())

# plt.scatter(x.data.numpy(),y.data.numpy())

# plt.show()

2、搭建神经网络

流程：

(1)定义所以层属性

(2)逐层搭建层与层之间的关系链接即（def forward(x): pass）,中间会用到激活函数(relu).

Example:

import torch

import torch.nn.functional as F #激活函数在function库里面

class Net(torch.nn.Module):#继承torch的Module

def \_\_init\_\_(self,n\_feature,n\_hidden,n\_output):

super(Net,self).\_\_init\_\_()#继承\_\_init\_\_()功能

self.hidden = torch.nn.Linear(n\_feature,n\_hidden)#定义隐藏层线性输出

self.predict = torch.nn.Linear(n\_hidden,n\_output)#定义输出层线性输出

def forward(self,x):#输入信息—module中的forward功能

x = F.relu(self.hidden(x))#激活函数(隐藏层的线性值)

x = self.predict(x) #得到输出值

return x

3、训练网络

(1)训练工具：optimizer(优化器)：常用的有SGD, Momentum, RMSprop, Adam.

optimizer = torch.optim.SGD(net.parameters(),lr=0.5)#input：传入net的所有参数，学习率

loss\_func = torch.nn.MSELoss()#预测值和真实值的误差计算公式 (均方差)

for t in range(100):

prediction = net(x)#输入训练数据x,输出预测结果

loss = loss\_func(prediction,y)#计算loss(两者误差)

optimizer.zero\_grad()#梯度设为0：清空上一步残余更新参数值

loss.backward()#误差反向传播 计算 参数更新值

optimizer.step()#优化梯度：将参数更新值施加到net的parameters上

#可视化训练过程

if t%5 == 0:

#plt and show learning process

plt.cla()

plt.scatter(x.data.numpy(),y.data.numpy())

plt.plot(x.data.numpy(),prediction.data.numpy(),'r-',lw=5)

#plt.text(0.5,0,'Loss=%.4f',loss.data[0],fontdict={'size':20,'color':'red'})

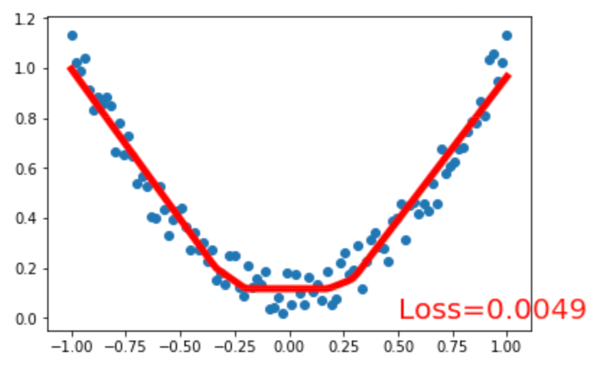
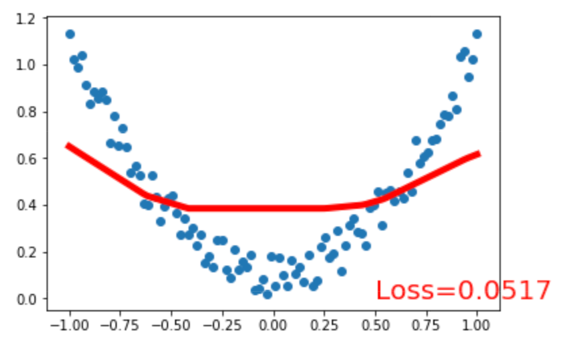
plt.text(0.5, 0, 'Loss=%.4f' % loss.data.numpy(), fontdict={'size': 20, 'color': 'red'})

plt.show()

plt.pause(0.1)

plt.ioff()

结果：



七、快速搭建法

net2 = torch.nn.Sequential(

torch.nn.Linear(1, 10),

torch.nn.ReLU(),

torch.nn.Linear(10, 1)

)

八、保存和提取

训练好了一个模型之后通过save方法进行保存模型,留到下次要用的时候直接提取(load)直接用,

1、保存

torch.save(net1, 'net.pkl') # 保存整个网络

torch.save(net1.state\_dict(), 'net\_params.pkl') # 只保存网络中的参数 (速度快, 占内存少)

2、提取

net2 = torch.load('net.pkl')

# 新建 net3 --搭建跟保存的文件(net\_params.pkl)网络要一样

net3 = torch.nn.Sequential(

torch.nn.Linear(1, 10),

torch.nn.ReLU(),

torch.nn.Linear(10, 1)

)

net3.load\_state\_dict(torch.load('net\_params.pkl'))

九、批训练-data loader

DataLoader是torch 给你用来包装你的数据的工具.

操作步骤：1、将自己的 (numpy array 或其他) 数据形式装换成 Tensor

2、放进这个包装器中.这样可以帮你有效地迭代数据。

Example：

import torch

import torch.utils.data as Data

BATCH\_SIZE = 5 # 批训练的数据个数

x = torch.linspace(1, 10, 10) # x data (torch tensor)

y = torch.linspace(10, 1, 10) # y data (torch tensor)

# 先转换成 torch 能识别的 Dataset

torch\_dataset = Data.TensorDataset(data\_tensor=x, target\_tensor=y)

# 把 dataset 放入 DataLoader

loader = Data.DataLoader(

dataset=torch\_dataset, # torch TensorDataset format

batch\_size=BATCH\_SIZE, # mini batch size

shuffle=True, # 要不要打乱数据 (打乱比较好)

num\_workers=2, # 多线程来读数据

)

for epoch in range(3): # 训练所有数据次数=3

for step, (batch\_x, batch\_y) in enumerate(loader): # 每一步 loader 释放一小批数据用来学习

# 打出来一些数据

print('Epoch: ', epoch, '| Step: ', step, '| batch x: ',

batch\_x.numpy(), '| batch y: ', batch\_y.numpy())

后续有CNN网络、GAN网络、强化学习、RNN以及上述提到的代码见GitHub：