Readme

Day01

Date:2021-07-22
Made By: 纸 石头 紫阳花

深度学习框架 Pytorch

理解基本的神经网络以及深度学习系统

1. 准备数据集
   数据集分割为训练集和测试集
2. 选择训练模型
3. 对模型进行训练
   在训练过程中能知道输入和对应输出的标签的学习模式称为监督学习。

在训练结束后通过测试集（测试集通常只知输入而不知输出）来确定模型的精确度。
泛化能力差说明模型过拟合，故通常在训练集中再切分出一个开发集（验证集）进行模型的评估。（由于大多时候测试集只知输入而不知输出），如果对于开发集的泛化能力强，则将所有训练集数据（包括开发集）用来对模型进行训练，将训练出来的模型用于对测试集进行预测。

4. 确定模型权重
5. 对新数据进行预测

Tensor是Pytorch最基本的数据类型，可以动态的构建计算图。Tensor中可以保存data和grad（grad也是tensor在与data共同使用时要取grad.data，否则会二次建立计算图）分别存储结点的值和其关于损失函数的梯度。

Tensor的基本方法

tensor.data.view()相当于reshape，重组张量的维度。

pytorch的广播机制

假如在运算中两个张量逻辑上无法进行运算，则可以通过广播机制扩充其中维度不够的张量。

Pytorch构建计算图

```
import torch
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = torch.Tensor([1.0])
w.requires_grad = True        # 是否存储梯度

for epoch in range(100)
    for x, y in zip(x_data, y_data):
        l = loss(x, y)
        l.backward()          # 自动进行关于loss的反向传播，通过w和x的grad属性获取w和x关于loss的导数
    print("grad = ", w.grad.item())      # .item()可以直接取出tensor中的数值最为标量存在
    w.data -= 0.01 * w.grad.data

    w.grad.data.zero_()       # 将梯度重新置零
```

`l = loss`代表正向传播的过程

`l.backward()`代表反向传播的过程

所有在反向传播中计算出的梯度都存储在结点张量的grad中。

线性模型

在机器学习中一般先使用线性模型进行测试，根据结果对模型进行更换选择更好的模型。

线性模型公式：

$$\hat{y} = x * \omega$$

w表示权重，在机器学习中对权重的处理方法是首先随机取一个随机数作为权重，随后对当前权重所表示的模型进行评估，确定偏移程度。

即 $\hat{y}(x_i) - y(x_i)$

故需要一个专用于测试偏移程度的评估模型，即Train Loss损失函数(针对单个样本)。由于上述差值可能为负，故对结果求平方。(n为训练集长度)

$$loss = (\hat{y} - y)^2$$

损失函数结果越接近0说明该样本关于模型拟合程度越高。

对于整个训练模型，损失函数我们采用Mean Squard Error（MSE）求每个样本数据的偏移程度的平均值。

$$cost = \frac{1}{N}\sum_{n=1}^{N}(\hat{y_n} - y_n)^2$$

Python实现线性模型

```
import numpy as np
import matpltlib.pyplot as plt

# 构建数据集
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

# 正向传播(线性模型)
def forward(x):
    return x*w

# 损失函数
def loss(x, y):
    y_predict = forward(x)
    return np.sqrt((y_predict - y), 2)

w_list = [] # 权重列表
mse_list = [] # 损失值列表
for i in np.arange(0.0, 4.1, 0.1)
    print("w = ", w)
    l_sum = 0
    for x_val, y_val in zip(x_data, y_data):
```

```
        y_predict = forward(y)
        loss_val = loss(x_val, y_val)
        l_sum += loss_val
     print(f"MSE = {l_sum/len(x_data)}")
    w_list.append(w)
    mse_list.append(l_sum/len(x_data))


    # 绘图
    plt.plot(w_list, mse_list)
    plt.ylabel("Loss")
    plt.xlabel("w")
    plt.show()
```

梯度下降算法

梯度下降算法用来取得一个相对最佳的权重，首先为模型随机取一个权重。

对MSE关于权重求导（利用反向传播进行）

$$\frac{\partial cost}{\partial w}$$

更新权重的方法为：

$$\omega = \omega - \alpha \frac{\partial cost}{\partial w} = \omega - \alpha \frac{1}{N} \sum_{n=1}^{N} \cdot 2 \cdot x_n \cdot (x_n - y_n)$$

α作为超参数学习率，决定每次梯度下降对权重更新的程度。

梯度下降法只能找到局部最优点（局部最优非常少见，通常鞍点的存在会带来更大问题）而无法找到全局最优点。

鞍点（导数为0但并非最优点）

损失函数python实现

```
def cost(xs, ys):
    cost = 0
    for x, y in zip(xs, ys):
        y_predict =  forward(x)
        cost += (y_predict - y)**2
    return cost/len(xs)
```

梯度下降python实现

```
def gradient(xs, ys):
    grad = 0
    for x, y in zip(xs, yx):
        grad += 2 * x * (x - y)
    return grad/len(xs)
```

训练过程

```
a  = 0.01
for epoch in range(100):|
    cost_val = cost(x_data, y_data)
    grad_val = gradient(x_data, y_data)
    w -= a * grad_val
    w_list.append(w)
```

通常假如最后图像收敛于0说明训练收敛，否则训练发散。

通常发散的原因都是学习率过大，应适当的降低学习率

随机梯度下降

在实际的学习过程中梯度下降的应用较小，通常我们使用的是随机梯度下降（Stochastic Gradient Descent） $\omega = \omega - \alpha \frac{\partial loss}{\partial \omega}$

每一次更新从n个数据中随机选取的损失作为更新依据

随机梯度下降可避免鞍点。

随机梯度下降python实现

```
def loss(x, y):
    return (forward(x) - y)**2
def gradient(x, y):
    return 2 * x * (x * w - y)
```

训练过程

```
for epoch in range(100)
    for x, y in zip(x_data, y_data):
        grad = gradient(x)       # 无需将gradient相加，  直接进行更新
        w = w - 0.01 * grad
        l = loss(x, y)
```

随机梯度下降由于权重是每个数据进行求gradient时所共同依赖的，所以只能串行运行。故牺牲了并行性。所以通常使用折衷的算法：mini-batch。

反向传播

随着神经网络层数的增多，逐层求导变得困难，所以需要反向传播来根据链式法则的原理进行求导。

计算图

反向传播的核心在于计算图，计算图的每一步都只能进行原子计算。

计算图先经过正向计算（前馈过程），在正向计算过程中，当对一个结点计算时，同时计算该结点输出结果（如c = a + b）关于其前一层结点输出内容（a和b）的偏导值（即$\frac{\partial c}{\partial a}$和$\frac{\partial c}{\partial b}$），并存储在前一层结点（a和b）中。

当除输出层外每条路径的每个结点上都存储有对下一层结点的偏导值时，根据链式法则，将这条路径上所有存储的偏导内容相乘，可得到输出层结点对这条路径输入层结点的偏导值。

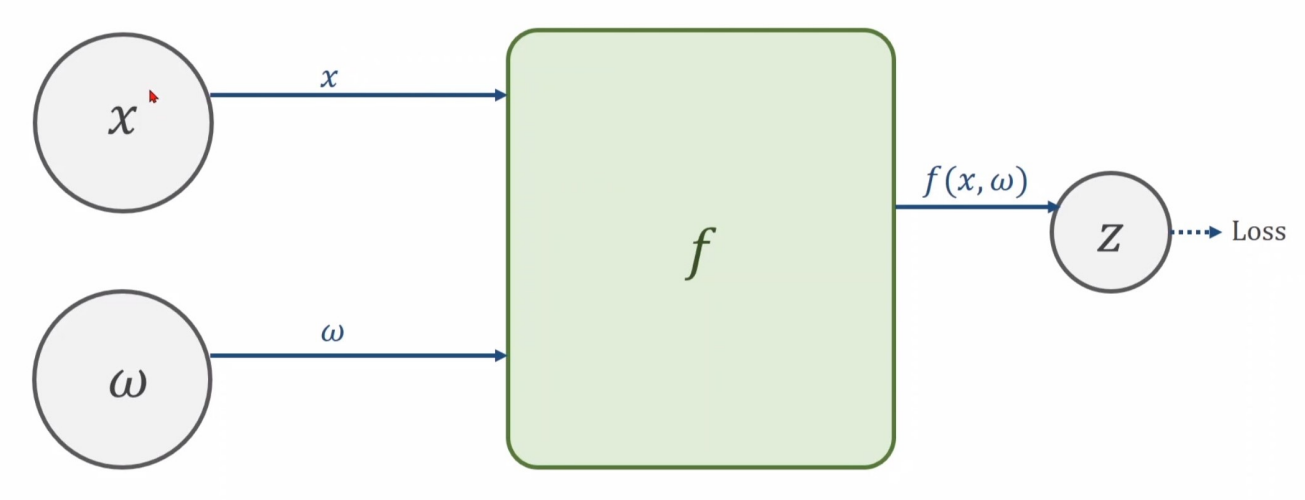
Pytorch的计算图采用动态图的方式，无需手动释放，在计算完成后框架自动释放计算图。

神经网络

模型的权重数可以确定一个神经网络模型的层数，由于模型的权重和偏重经过合并最终都可以化为单层神经网络，为了保持神经网络的复杂度，为每一层神经网络的输出增加一个非线性的变化函数σ（如sigmoid函数）成为激活函数。

sigmoid函数：$y = \frac{1}{1+e^{-x}}$

链式求导运算过程



1. 创建计算图
2. 前馈运算：
   正向由输入先计算最后的Loss，并在计算的过程中求出输出值z关于x和ω的导数并进行存储。
3. 反向传播：
   输出接口z拿到损失Loss关于z的偏导，在经过f时，分别和之前计算的关于x和w的偏导数进行相乘，根据链式法则原理，最终求得损失函数关于权重的导数即gradient。