PyTorch入门

黄海广 副教授

2021年03月

本章目录

1.Tensors张量

01 Tensors张量

02 Autograd自动求导

03 神经网络

04 训练一个分类器

1.Tensors张量的概念

Tensor实际上就是一个多维数组（multidimensional array）

标量（0阶张量）

向量（1阶张量）

矩阵（2阶张量）

张量（大于等于3阶张量）

1.Tensor张量乘法

1. 二维矩阵乘法 torch.mm()

```
torch.mm(mat1, mat2, out=None)
```

其中$mat1 \in \mathbb{R}^{n \times m}$,$mat2 \in \mathbb{R}^{m \times d}$, 输出的$out \in \mathbb{R}^{n \times d}$

该函数一般只用来计算两个二维矩阵的矩阵乘法，并且不支持broadcast操作。

1.Tensor张量乘法

2. 三维带batch的矩阵乘法 torch.bmm()

由于神经网络训练一般采用mini-batch，经常输入的时三维带batch的矩阵，所以提供`torch.bmm(bmat1, bmat2, out=None)`

其中 $bmat1 \in \mathbb{R}^{b\times n\times m}$,$bmat2 \in \mathbb{R}^{b\times m\times d}$, 输入出的 $out \in \mathbb{R}^{b\times n\times d}$

该函数的两个输入必须是三维矩阵并且第一维相同（表示Batch维度）， 不支持broadcast操作

1.Tensor张量乘法

3. 多维矩阵乘法 torch.matmul()

`torch.matmul(input, other, out=None)`

支持broadcast操作，使用起来比较复杂。针对多维数据`matmul()`乘法，可以认为该乘法使用使用两个参数的后两个维度来计算，其他的维度都可以认为是batch维度。

假设两个输入的维度分别是`input`$(1000\times500\times99\times11)$, `other`$(500\times11\times99)$那么我们可以认为`torch.matmul(input, other, out=None)`乘法首先是进行后两位矩阵乘法得到$(99\times11)\times(11\times99)\Rightarrow(99\times99)$，然后分析两个参数的batch size分别是$(1000\times500)$ 和 500, 可以广播成为 (1000×500)， 因此最终输出的维度是$(1000\times500\times99\times99)$。

1.Tensors张量乘法

4. 矩阵逐元素(Element-wise)乘法 torch.mul()

```
torch.mul(mat1, other, out=None)
```

其中 other 乘数可以是标量，也可以是任意维度的矩阵，只要满足最终相乘是可以broadcast的即可。

1.Tensors张量乘法

5. 两个运算符 @ 和 *

@：矩阵乘法，自动执行适合的矩阵乘法函数

***：element-wise乘法**

2. Autograd自动求导

01 Tensors张量

02 Autograd自动求导

03 神经网络

04 训练一个分类器

2. Autograd自动求导

PyTorch之自动梯度

在训练一个神经网络时，梯度的计算是一个关键的步骤，它为神经网络的优化提供了关键数据。

但是在面临复杂神经网络的时候导数的计算就成为一个难题，要求人们解出复杂、高维的方程是不现实的。

这就是自动求导出现的原因，当前最流行的深度学习框架如PyTorch、Tensorflow等都提供了自动微分的支持，让人们只需要很少的工作就能神奇般地自动计算出复杂函数的梯度。

2. Autograd自动求导

requires_grad属性

requires_grad属性默认为False，也就是Tensor变量默认是不需要求导的。

如果一个节点的requires_grad是True，那么所有依赖它的节点requires_grad也会是True。

换言之，如果一个节点依赖的所有节点都不需要求导，那么它的requires_grad也会是False。在反向传播的过程中，该节点所在的子图会被排除在外。

2. Autograd自动求导

Function类

我们已经知道PyTorch使用动态计算图(DAG)记录计算的全过程,DAG的节点是Function对象，边表示数据依赖，从输出指向输入。因此Function类在PyTorch自动求导中位居核心地位，但是用户通常不会直接去使用。

每当对Tensor施加一个运算的时候，就会产生一个Function对象，它产生运算的结果，记录运算的发生，并且记录运算的输入。Tensor使用.grad_fn属性记录这个计算图的入口。反向传播过程中，autograd引擎会按照逆序，通过Function的backward依次计算梯度。

2. Autograd自动求导

backward函数

backward函数是反向传播的入口点，在需要被求导的节点上调用backward函数会计算梯度值到相应的节点上。
backward函数是反向求导数，使用链式法则求导。

backward需要一个重要的参数grad_tensor，对非标量节点求导,需要指定grad_tensors，grad_tensors的shape必须和y的相同。

但如果节点只含有一个标量值，这个参数就可以省略（例如最普遍的loss.backward()与loss.backward(torch.tensor(1))等价）

2. Autograd自动求导

grad属性

backward函数本身没有返回值，它计算出来的梯度存放在叶子节点的grad属性中。

PyTorch文档中提到，如果grad属性不为空，新计算出来的梯度值会直接加到旧值上面。 为什么不直接覆盖旧的结果呢?

这是因为有些Tensor可能有多个输出，那么就需要调用多个backward。叠加的处理方式使得backward不需要考虑之前有没有被计算过导数，只需要加上去就行了，这使得设计变得更简单。

因此我们用户在反向传播之前，常常需要用zero_grad函数对导数手动清零，确保计算出来的是正确的结果。

2. Autograd自动求导

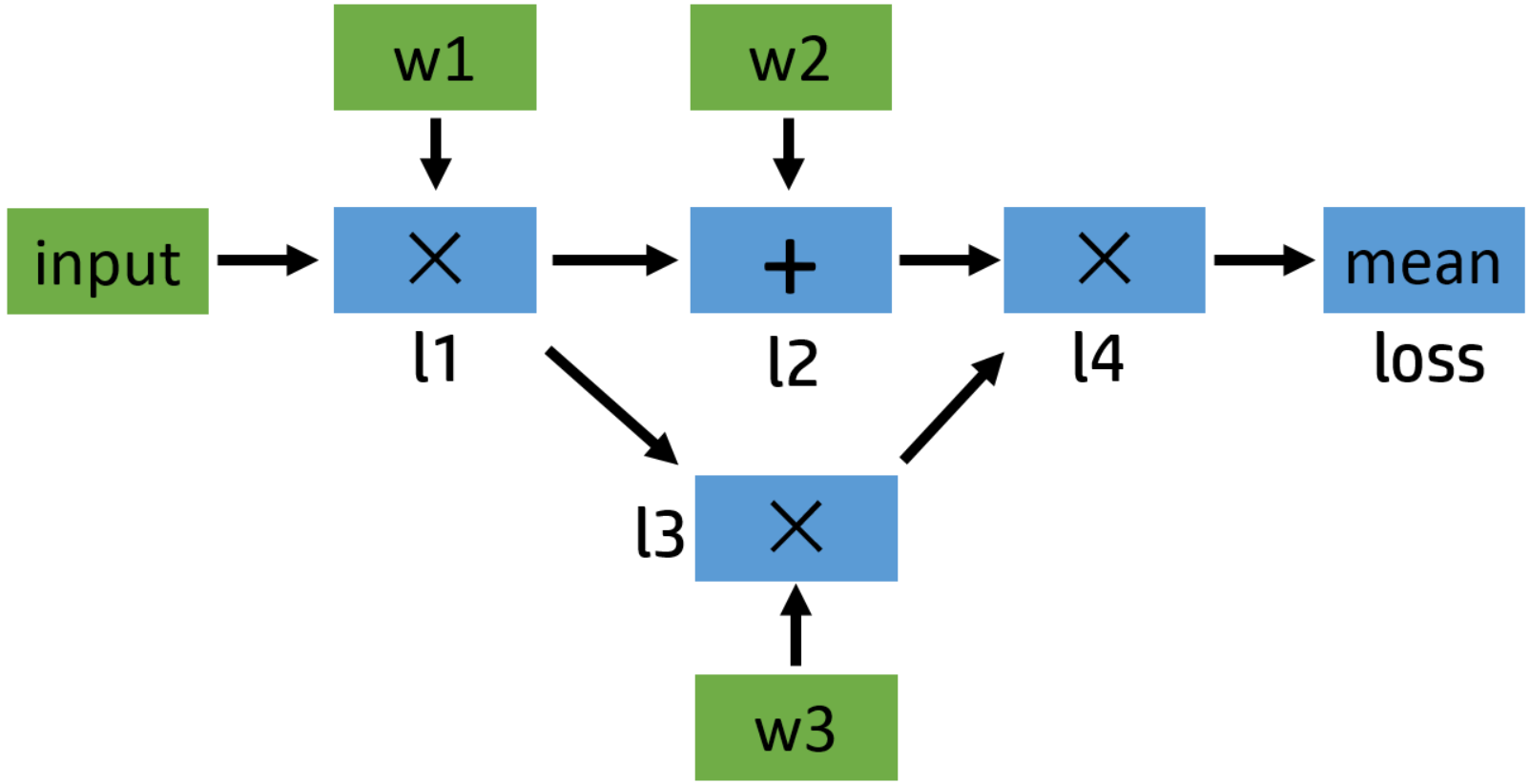
l1 = input x w1

l2 = l1 + w2

l3 = l1 x w3

l4 = l2 x l3

loss = mean(l4)



2. Autograd自动求导

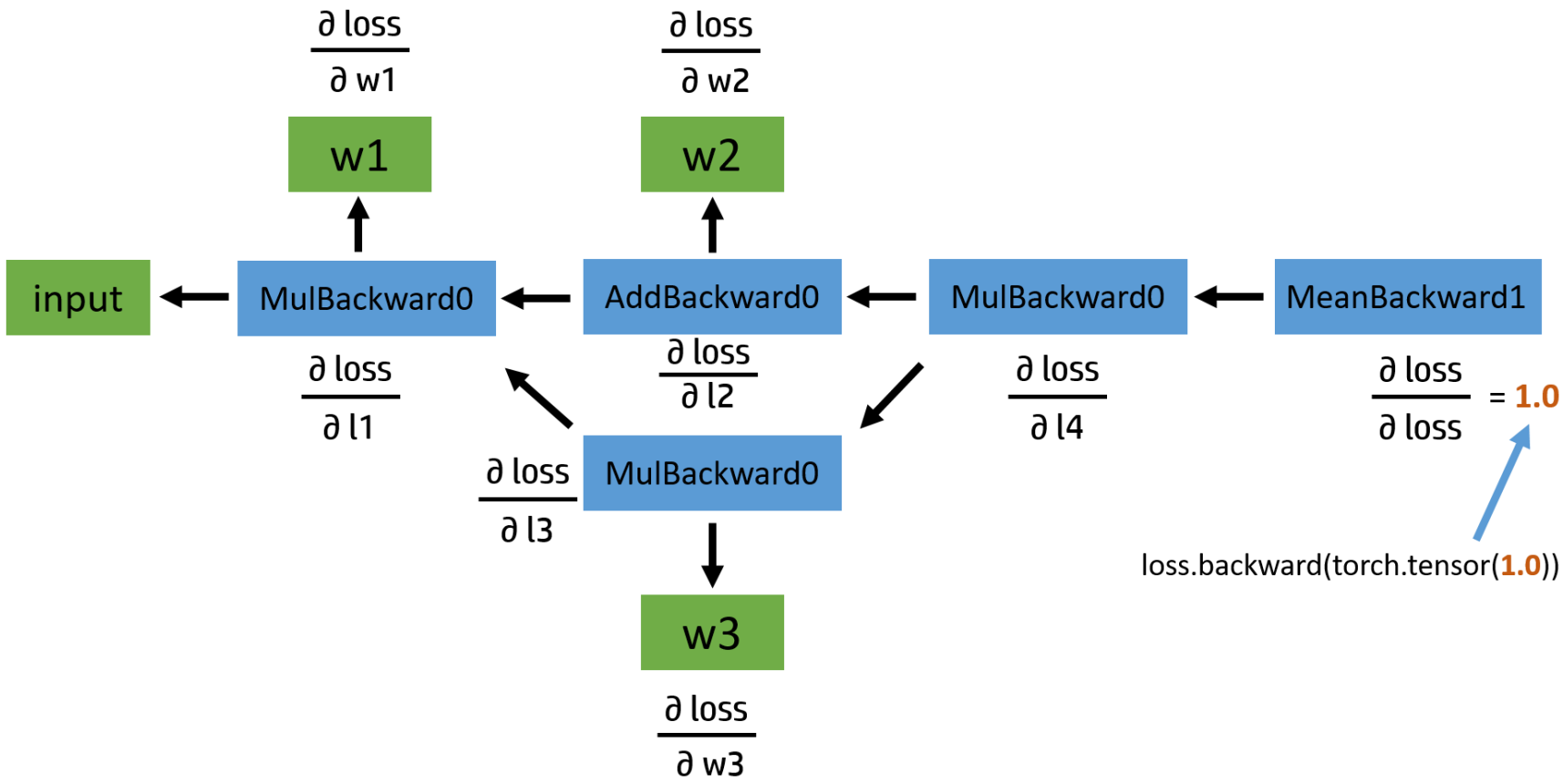
l1 = input x w1

l2 = l1 + w2

l3 = l1 x w3

l4 = l2 x l3

loss = mean(l4)



只有 **input** 一个变量是 **requires_grad=False** 的

https://zhuanlan.zhihu.com/p/69294347

3. 神经网络

01 Tensors张量

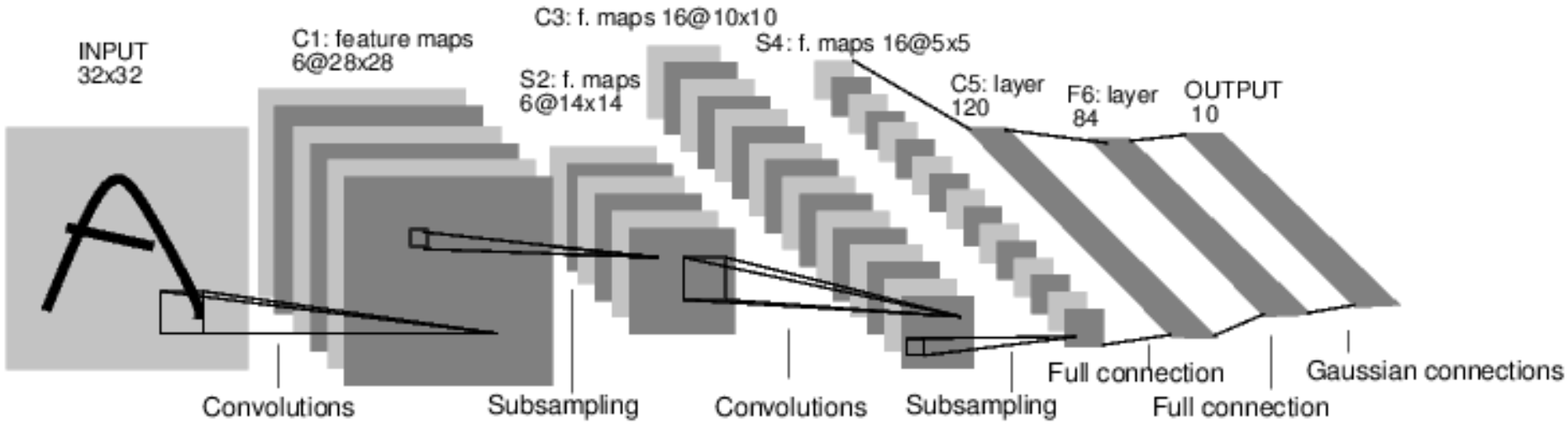
02 Autograd自动求导

03 神经网络

04 训练一个分类器

3. 神经网络

可以使用torch.nn包来构建神经网络. 你已知道autograd包,nn包依赖autograd包来定义模型并求导.一个nn.Module包含各个层和一个forward(input)方法,该方法返回output。



典型的神经网络

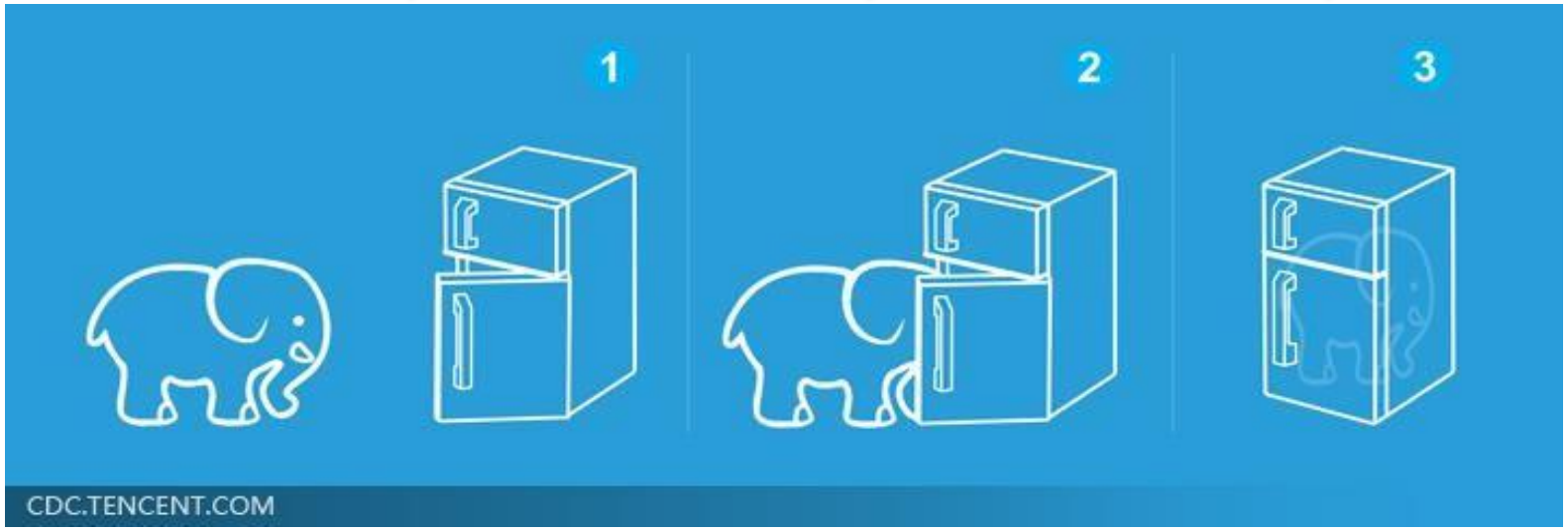
3. 神经网络

神经网络的典型训练过程如下:

- 定义神经网络模型,它有一些可学习的参数(或者权重);
- 在数据集上迭代;
- 通过神经网络处理输入;
- 计算损失(输出结果和正确值的差距大小)
- 将梯度反向传播回网络的参数;
- 更新网络的参数,主要使用如下简单的更新原则:

```
weight = weight - learning_rate * gradient
```

深度学习的三个步骤



深度学习很简单……

来源：李宏毅《1天搞懂深度学习》

3. 神经网络

torch.Tensor-支持自动编程操作（如backward()）的多维数组。同时保持梯度的张量。

nn.Module-神经网络模块.封装参数,移动到GPU上运行,导出,加载等

nn.Parameter-一种张量,当把它赋值给一个Module时,被自动的注册为参数。

autograd.Function-实现一个自动求导操作的前向和反向定义, 每个张量操作都会创建至少一个Function节点，该节点连接到创建张量并对其历史进行编码的函数。

4. 训练一个分类器

01 Tensors张量

02 Autograd自动求导

03 神经网络

04 训练一个分类器

4. 训练一个分类器

训练一个分类器流程

4. 训练一个分类器

torch.nn.Linear

PyTorch的nn.Linear（）是用于设置网络中的全连接层的，需要注意的是全连接层的输入与输出都是二维张量，一般形状为[batch_size, size]，不同于卷积层要求输入输出是四维张量。

in_features指的是输入的二维张量的大小，即输入的[batch_size, size]中的size。

out_features指的是输出的二维张量的大小，即输出的二维张量的形状为[batch_size，output_size]，当然，它也代表了该全连接层的神经元个数。

从输入输出的张量的shape角度来理解，相当于一个输入为[batch_size, in_features]的张量变换成了[batch_size, out_features]的输出张量。

4. 训练一个分类器

torch.nn

计算图和autograd是十分强大的工具，可以定义复杂的操作并自动求导；然而对于大规模的网络，autograd太过于底层。 在构建神经网络时，我们经常考虑将计算安排成层，其中一些具有可学习的参数，它们将在学习过程中进行优化。

TensorFlow里，有类似Keras，TensorFlow-Slim和TFLearn这种封装了底层计算图的高度抽象的接口，这使得构建网络十分方便。

在PyTorch中，包nn 完成了同样的功能。nn包中定义一组大致等价于层的模块。一个模块接受输入的tesnor，计算输出的tensor，而且 还保存了一些内部状态比如需要学习的tensor的参数等。nn包中也定义了一组损失函数（loss functions），用来训练神经网络。

4. 训练一个分类器

torch.optim

使用optim包定义优化器（Optimizer）。Optimizer将会为我们更新模型的权重。
这里我们使用Adam优化方法；optim包还包含了许多别的优化算法。
Adam构造函数的第一个参数告诉优化器应该更新哪些张量。

```
learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

参考文献

1. IAN GOODFELLOW等，《深度学习》，人民邮电出版社，2017
2. Andrew Ng，http://www.deeplearning.ai
3. Christopher M. Bishop, Pattern Recognition and Machine Learning, Springer-Verlag, 2006
4. 李宏毅，《一天搞懂深度学习》


谢 谢!