Pytorch—Tutorial

<Excerpt in index | 首页摘要>
The blog mainly involve how to learning Pytorch and completing some project by Pytorch
KeyWords Plus: **Pytorch** **DL** **CNN** **DNN**

- **Relevant blog**：Pytorch-book Pytorch-tutorial Pytorch
- **official tutorial**：强烈推荐看官网

<The rest of contents | 余下全文>

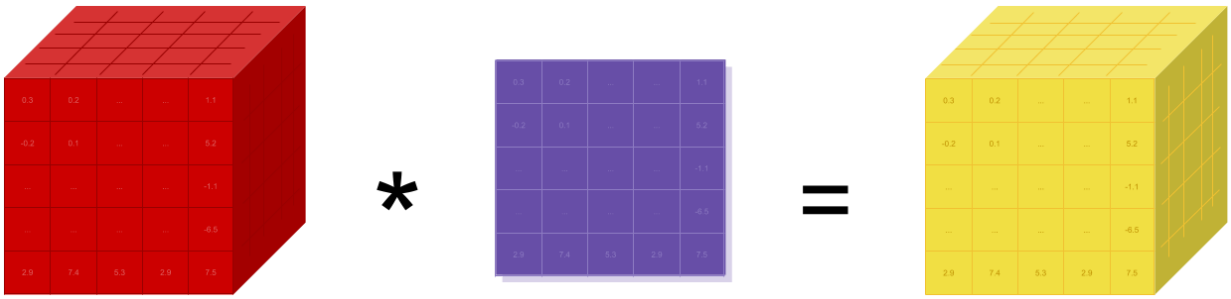
Basics

Introduction

PyTorch is a Python package that provides two high-level features:

- Tensor computation (like NumPy) with strong **GPU acceleration**
- **Deep neural networks** built on a tape-based autograd system

A GPU-Ready Tensor Library

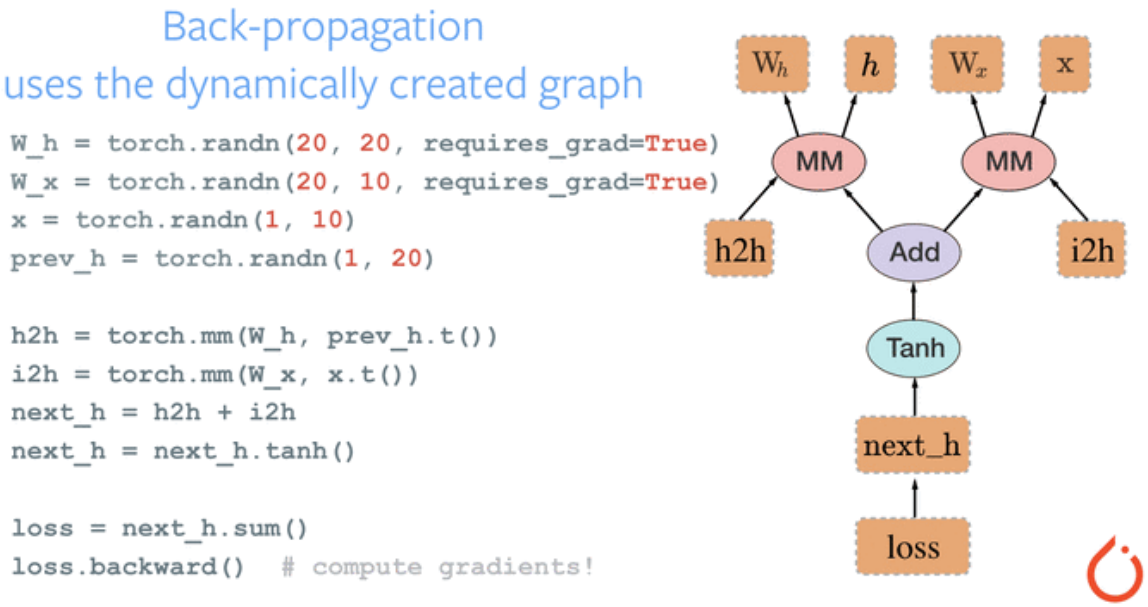


PyTorch provides Tensors that can live either on the **CPU** or the **GPU**, and **accelerates the computation by a huge amount.**

Pytorch provide a wide variety of tensor routines to accelerate and fit your **scientific computation** needs such as slicing, indexing, math operations, linear algebra, reductions. And they are fast!

Dynamic Neural Networks: Tape-Based Autograd

PyTorch has a unique way of building neural networks: **using and replaying a tape recorder.**



Installation

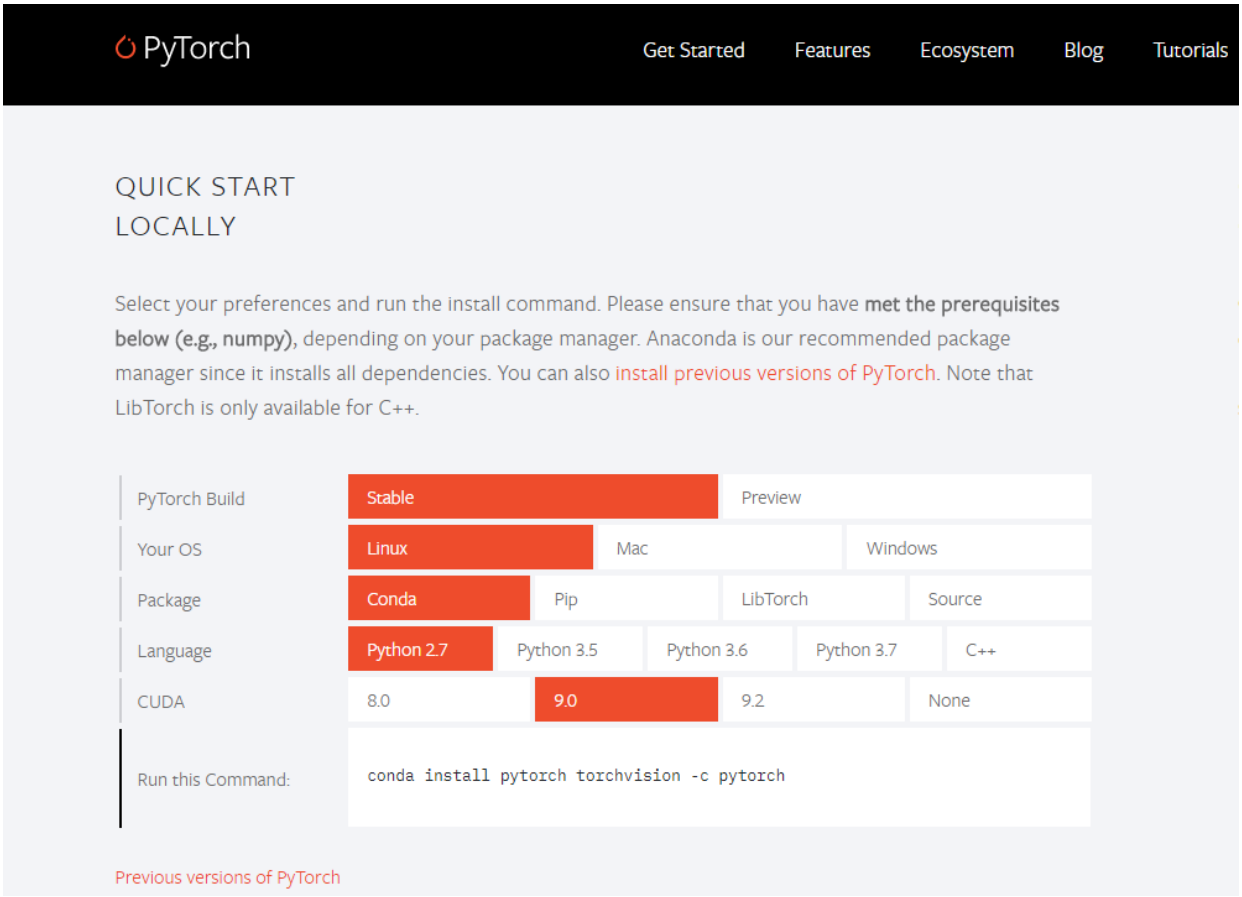
参考https://pytorch.org/

查看python版本

```
python –version
```

查看cuda 版本

```
cat /usr/local/cuda/version.txt
```



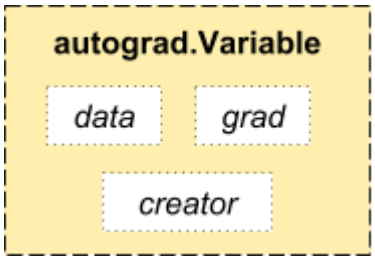
Autograd

`PyTorch`中所有神经网络的核心是`autograd`自动求导包. 我们先来简单介绍一下, 然后我们会去训练我们的第一个神经网络.

Variable（变量）

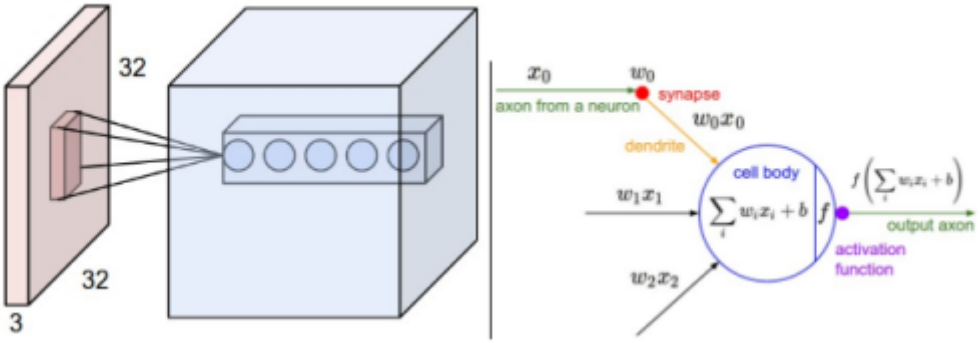
· `autograd.Variable` 是包的核心类. 它包装了张量, 并且支持几乎所有的操作. 一旦你完成了你的计算, 你就可以调用 `.backward()` 方法, 然后所有的梯度计算会自动进行.

你还可以通过`.data`属性来访问原始的张量, 而关于该`variable`（变量）的梯度会被累计到 .grad 上去.



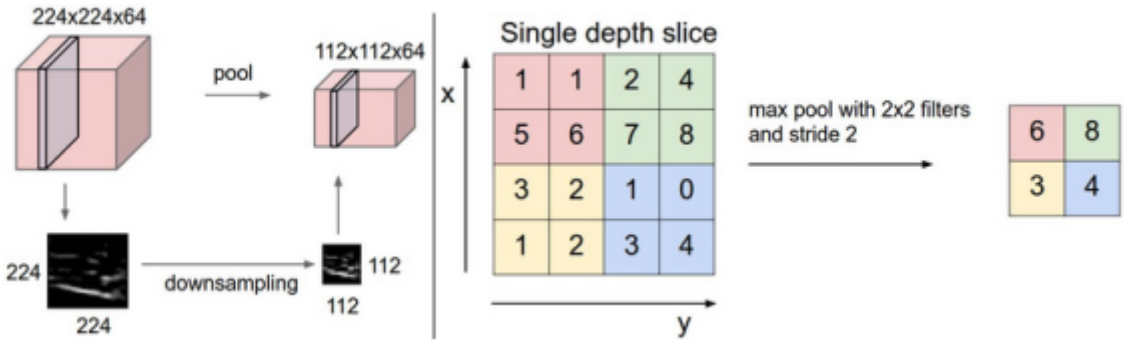
CNN

卷积层



所谓的卷积，就是这种小方块，我们设置一个小方块的大小，但是这个小方块的厚度必须和左边的这个大方块的厚度是一样的，大方块每一个像素点由一个0到255的数字表示，这样我们就可以赋予小方块权重，比如我们取小方块的大小是3×3，我们要求起厚度必须要和左边的大方块厚度一样，那么小方块的的大小就为3x3x3，我们就可以赋予其3x3x3个权重，然后我们就可以开始计算卷积的结果，将小方块从大方块的左上角开始，一个卷积小方块所覆盖的范围是3x3x3，然后我们将大方块中3x3x3的数字和小方块中的权重分别相乘相加，再加上一个偏差，就可以得到一个卷积的接过，可以抽象的写成Wx b这种形式，这就是图上所显示的接过，然后我们可以设置小方块的滑动距离，每次滑动就可以形成一个卷积的计算结果，**然后讲整张大图片滑动覆盖之后就可以形成一层卷积的结果，我们看到图中的卷积结果是很厚的，也就是设置了很多层卷积。**总结来说，就是每层卷积就是一个卷积核在图片上滑动求值，然后设置多个卷积核就可以形成多层的卷积层。

池化层



两种处理方式，一种是取这个小窗口里面所有元素的**最大值**来代表这个小窗口，一种是取**平均值**，然后将小窗口滑动，在第二的位置再做同样的处理，上层网络输出方块的每一层做完之后就进入这个大方块的下一层做同样的操作，这个处理办法就可以让整个大方块的大小变小

Code

定义网络

```
# Convolutional neural network (two convolutional layers)
class ConvNet(nn.Module):
    def __init__(self, num_classes=10):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
```

```
            nn.Conv2d(1, 16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Linear(7*7*32, num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        return out
```

定义损失函数和优化器

神经网络强大之处就在于 `反向传播`，通过比较 `预测结果` 与 `真实结果`， 修整 `网络参数`。这里的 `比较` 就是 `损失函数`，而 `修整网络参数` 就是 `优化器`。

```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

训练网络

所有网络的训练的流程都是类似的，不断执行（轮）：

- 给**网络输入数据**
- **前向传播+反向传播**
- **更新网络参数**

```
for epoch in range(epochs):

    running_loss = 0.0

    for i, data in enumerate(trainloader):
        inputs, labels = data
        #inputs, labels = Variable(inputs), Variable(labels)

        #梯度清零
        optimizer.zero_grad()

        #forward+backward
        outputs = net(inputs)
```

```
            #对比预测结果和labels，计算loss
            loss = criterion(outputs, labels)

            #反向传播
            loss.backward()

            #更新参数
            optimizer.step()

            #打印log
            running_loss += loss.item()
            if i % 2000 == 1999: #每2000个batch打印一次训练状态
                average_loss = running_loss/2000
                print("[{0},{1}] loss:  {2}".format(epoch+1, i+1, average_los
s))
                average_loss_series.append(average_loss)
                running_loss = 0.0
```

Grad-CAM

torchvision.models

PyTorch框架中有一个非常重要且好用的包：torchvision，该包主要由3个子包组成，分别是：

1、**torchvision.datasets**
2、**torchvision.models**
3、**torchvision.transforms**

对于models

使用例子：

```
    import torchvision
    model = torchvision.models.resnet50(pretrained=True)
```

这样就导入了resnet50的预训练模型了。如果只需要网络结构，不需要用预训练模型的参数来初始化，那么就是：

```
    model = torchvision.models.resnet50(pretrained=False)
```

Pytorch

pytorch-tutorial

Github

This repository provides tutorial code for deep learning researchers to learn **PyTorch**. In the tutorial, most of the models were implemented with **less than 30 lines of code**. Before starting this tutorial, it is recommended to finish Official Pytorch Tutorial.

Table of Contents

pytorch-book

Github

chapter10-图像描述(Image Caption)	update	4 months ago
chapter11-语音识别(LSTM-CTC)	refactor code	8 months ago
chapter2-快速入门	update chapter2 to v0.4	8 months ago
chapter3-Tensor和autograd	backup	4 months ago
chapter4-神经网络工具箱nn	backup	4 months ago
chapter5-常用工具	backup	4 months ago
chapter6-实战指南	update	4 months ago
chapter7-GAN生成动漫头像	update	4 months ago
chapter8-风格迁移(Neural Style)	update	4 months ago
chapter9-神经网络写诗(CharRNN)	update	4 months ago

这是书籍《**深度学习框架PyTorch：入门与实践**》的对应代码，但是也可以作为一个独立的PyTorch入门指南和教程。

基础部分 （前五章）讲解PyTorch内容，这部份介绍了PyTorch中主要的的模块，和深度学习中常用的一些工具。

第二章: 介绍如何安装PyTorch和**配置学习环境。**
第三章： 介绍了PyTorch中多维数组**Tensor和动态图**autograd/Variable的使用
第四章: 介绍了PyTorch中神经网络模块**nn的基础用法**，同时讲解了神经网络中**“层”，“损失函数”，“优化器”**等。
第五章: 介绍了PyTorch中数据加载，**GPU加速，持久化和可视化等相关工具。**

实战部分（第六到十章）利用PyTorch实现了几个酷炫有趣的应用

第六章: 结合Kaggle中一个经典的比赛，实现一个深度学习中比较**简单的图像二分类问题。**在实现过程中，提出代码规范以合理的组织程序，代码，使得程序更加可读，可维护。第六章还介绍了在PyTorch中如何进行debug。
第七章: 为读者讲解了当前最火爆的**生成对抗网络（GAN）**，带领读者从头实现一个**动漫头像生成器**，能够利用GAN生成风格多变的动漫头像。
第八章： 为读者讲解了**风格迁移的相关知识**，并带领读者实现风格迁移网络，将自己的**照片变成高大上的名画。**
第九章: 为读者讲解了一些**自然语言处理**的基础知识，并讲解了CharRNN的原理。而后利用收**集了几万首唐诗**，训练出了一个可以自动写诗歌的小程序。
第十章: 为读者介绍了**图像描述任务**，并以最新的**AI Challenger**比赛的数据为例，带领读者实现了一个可以进行简单图像描述的的小程序。
第十一章: （**新增，实验性**） 由Diamondfan 编写的语音识别。完善了本项目（本项目已囊括图像，文本，语音三大领域的例子）。

反馈与建议

- 微博：@柏林designer
- 邮箱：weijia.wu@foxmail.com