**什么是PyTorch**

1.张量可以使用.to方法移动到任何设备(device）上

2.y = torch.ones\_like(x, device=device) # 直接在GPU上创建tensor

3.x = x.to(device) # 或者使用.to("cuda")方法,转移到GPU上去

**Autograd：自动求导**

1. torch.Tensor 是这个包的核心类。如果设置它的属性 .requires\_grad 为 True，那么autograd将会追踪对于该张量的所有操作。当完成计算后可以通过调用 .backward()，来自动计算所有的梯度。这个张量的所有梯度将会自动累加到.grad属性.
2. 要阻止一个张量被跟踪历史，可以调用 .detach() 方法将其与计算历史分离，并阻止它未来的计算记录被跟踪。
3. 为了防止跟踪历史记录(和使用内存），可以将代码块包装在 with torch.no\_grad(): 中。在评估模型时特别有用，因为模型可能具有 requires\_grad = True 的可训练的参数，但是我们不需要在此过程中对他们进行梯度计算。
4. 创建一个张量并设置requires\_grad=True用来追踪其计算历史， x = torch.ones(2, 2, requires\_grad=True)，但其不可被训练
5. 标量不是常数，而是shape为1的特殊张量，shape为n的是向量
6. 若结果不再是标量，则torch.autograd 不能直接计算完整的雅可比矩阵
7. backward函数在使用autograd时自动定义，backward函数用来计算导数

**神经网络**

1.使用torch.nn包来构建神经网络.

2. 一个模型的可学习参数可以通过net.parameters()返回

3. 梯度是累加的，反向传播误差时，需要清零现有(之前保留在grad中)的梯度，重新存储当前反向传播计算出的梯度

**训练分类器**

1. 特别对于视觉方面，我们创建了一个包，名字叫torchvision，其中包含了针对Imagenet、CIFAR10、MNIST等常用数据集的数据加载器(data loaders），还有对图像数据转换的操作，即torchvision.datasets和torch.utils.data.DataLoader
2. torchvision数据集加载完后的输出是范围在[0, 1]之间的PILImage。我们将其标准化为范围在[-1, 1]之间的张量。

**数据并行处理**

1. 调用my\_tensor.to(device)返回一个GPU上的my\_tensor副本，而不是重写my\_tensor。你需要把它赋值给一个新的张量并在GPU上使用这个张量
2. 使用DataParallel让模型并行运行来轻易的在多个GPU上运行你的操作model = nn.DataParallel(model)
3. DataParallel自动的划分数据，并将作业顺序发送到多个GPU上的多个模型。DataParallel会在每个模型完成作业后，收集与合并结果然后返回

**模型部署**

1. ONNX Runtime 是针对 ONNX 模型的以性能为中心的引擎，可在多个平台和硬件(Windows，Linux 和 Mac 以及 CPU 和 GPU 上）高效地进行推理。事实证明，ONNX Runtime 可大大提高多个模型的性能。
2. 在导出模型前，先需要torch\_model.eval()或torch\_model.train(False)，以将模型转换为推理模式，这一点很重要。 这是必需的，因为像 dropout 或 batchnorm 这样的运算符在推断和训练模式下的行为会有所不同。
3. 要导出模型，我们调用torch.onnx.export()函数

**通过示例学习 PyTorch**

1. tensor为n 维张量，类似于 numpy，但可以在 GPU 上运行，Numpy 是用于科学计算的通用框架。 它对计算图，深度学习或梯度一无所知