第八讲 深度学习软件

2019年5月9日　　9:18

1.CPU VS GPU

1.1 CPU VS GPU

CPU vs GPU

	# Cores	Clock Speed	Memory	Price
CPU (Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.4 GHz	Shared with system	$339
CPU (Intel Core i7-6950X)	10 (20 threads with hyperthreading)	3.5 GHz	Shared with system	$1723
GPU (NVIDIA Titan Xp)	3840	1.6 GHz	12 GB GDDR5X	$1200
GPU (NVIDIA GTX 1070)	1920	1.68 GHz	8 GB GDDR5	$399

CPU: Fewer cores, but each core is much faster and much more capable; great at sequential tasks

GPU: More cores, but each core is much slower and "dumber"; great for parallel tasks

图8.1.1 CPU与GPU

1.2 Programing GPUS

- CUDA (NVIDIA only)
  - Write C-like code that runs directly on the GPU
  - Higher-level APIs: cuBLAS, cuFFT, cuDNN, etc
- OpenCL
  - Similar to CUDA, but runs on anything
  - Usually slower :(
- Udacity: Intro to Parallel Programming https://www.udacity.com/course/cs344
  - For deep learning just use existing libraries

2.Deep Learning Framworks

2.1 Tensorflow

使用框架优点：1.轻松构建计算图
　　　　　　　2.轻松使用GPU
　　　　　　　3.底层运算很高效

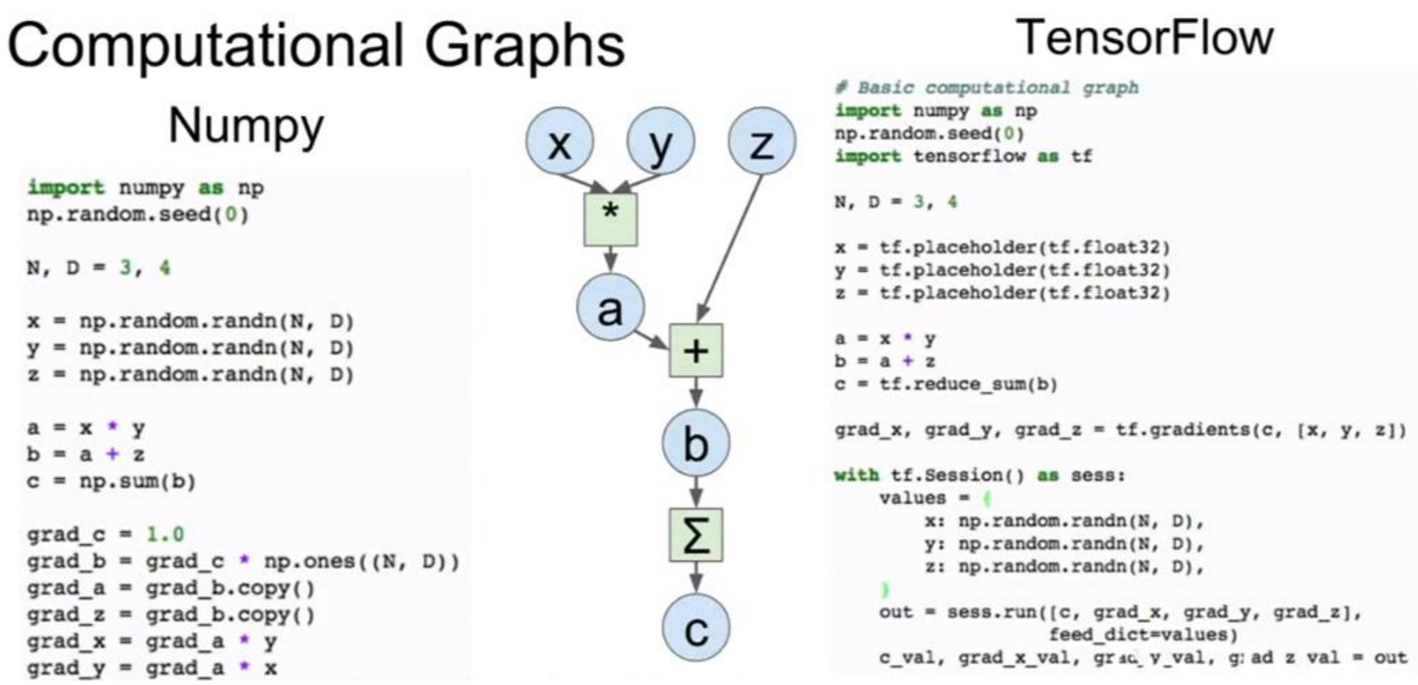


图 8.2.1 numpy 与tensorflow计算对比

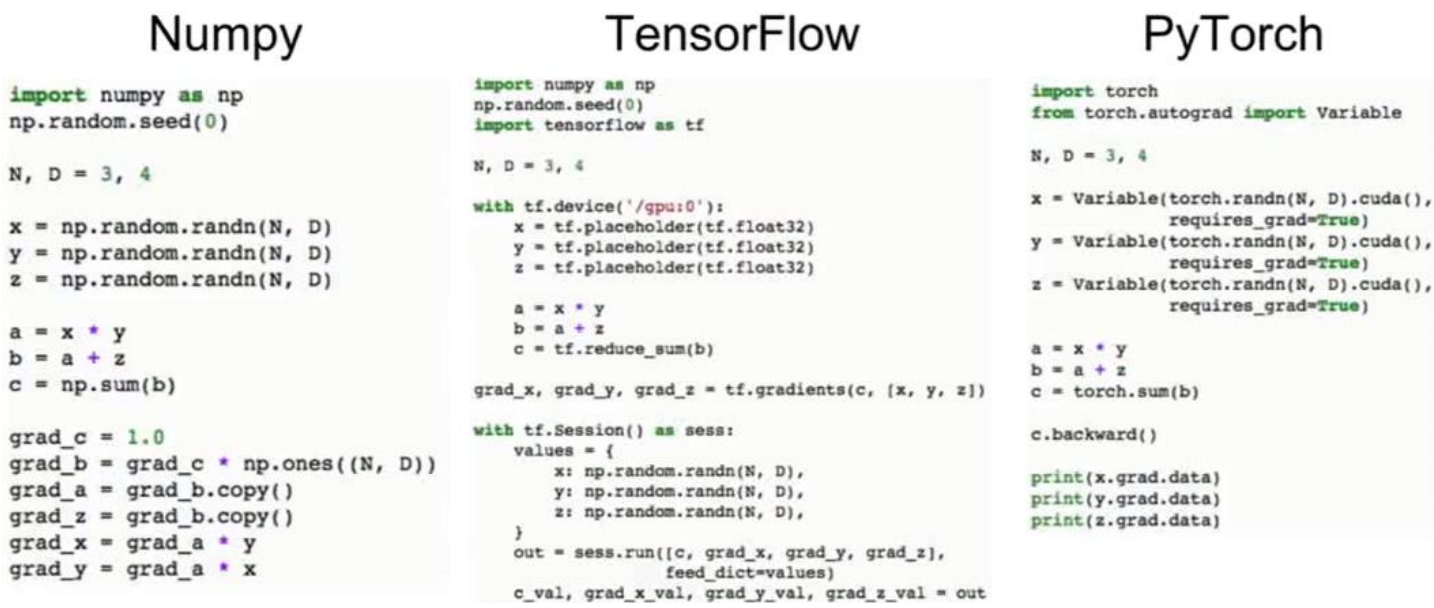


图 8.2.2 numpy tensorflow pytorch

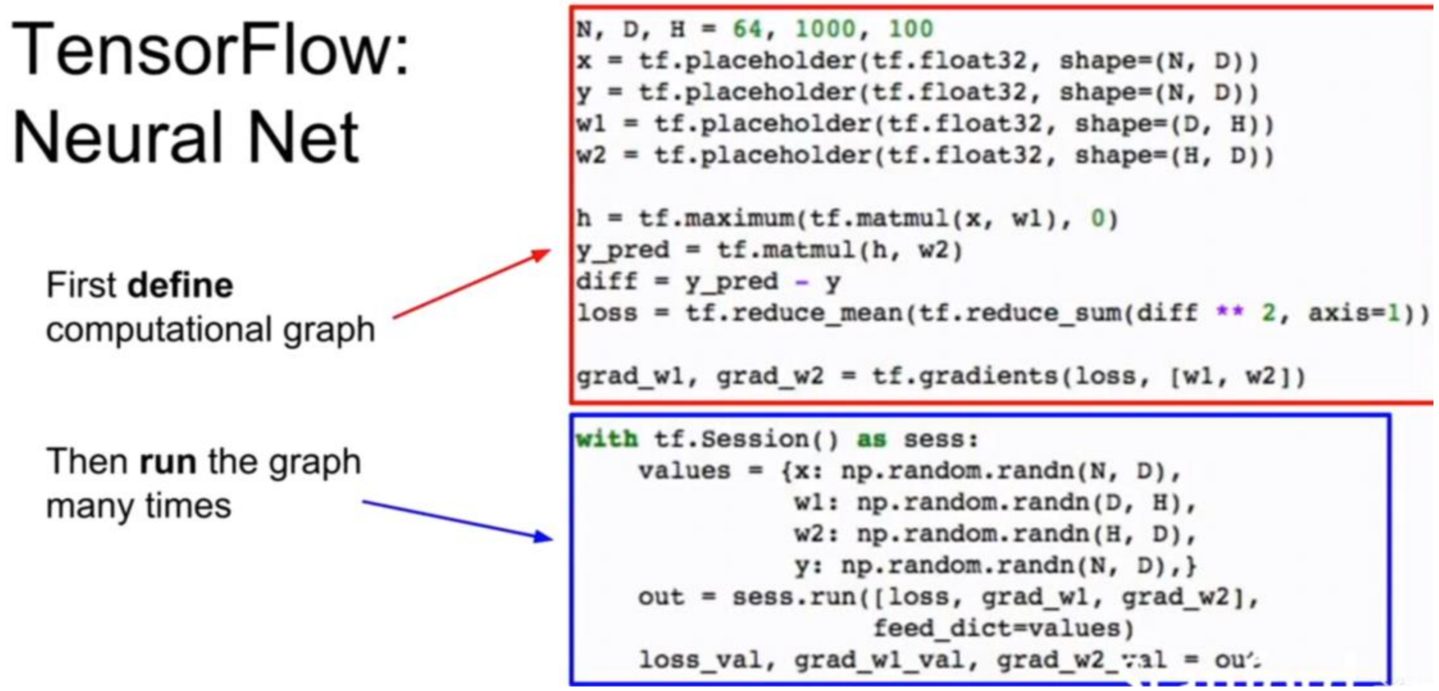


图 8.2.3 tensorflow 构建

两层RElu网络在tensorflow构建过程:

1. 红色部分构建模型图
2. 蓝色部分使用模型运算

Tf.placeholder(tf.float32,shape=(N,P)) 占位符

W1 = Tf.Variable(tf.random_normal((N,D))) 变量

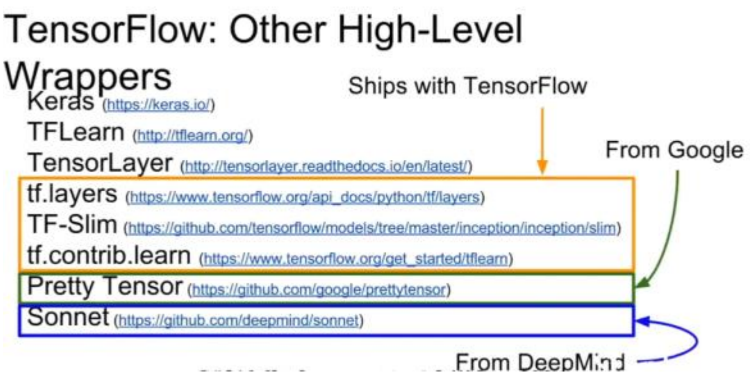
W1.assign(w1 - learnrate * grad) 更新变量

Tf.maxmum(tf.matmul(x,W1),0) 矩阵乘法

L2损失:tf.losses.mean_squared_error(y_pre, y_real)

占位符作为图的输入存放在内存中,而变量是位于图中,在计算时变量始终存放在GPU显存中。

官方demo:https://tensorflow.google.cn/tutorials/keras/basic_classification



对tensorflow更高级别的封装框架,首推:keras 最新:sonnet

2.2 pytorch

PyTorch: Tensors

PyTorch Tensors are just like numpy arrays, but they can run on GPU.

No built-in notion of computational graph, or gradients, or deep learning.

Here we fit a two-layer net using PyTorch Tensors:

```
import torch

dtype = torch.FloatTensor

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in).type(dtype)
y = torch.randn(N, D_out).type(dtype)
w1 = torch.randn(D_in, H).type(dtype)
w2 = torch.randn(H, D_out).type(dtype)

learning_rate = 1e-6
for t in range(500):
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)
    loss = (y_pred - y).pow(2).sum()

    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = x.t().mm(grad_h)
```

PyTorch: Tensors

To run on GPU, just cast tensors to a cuda datatype!

```
import torch

dtype = torch.cuda.FloatTensor

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in).type(dtype)
y = torch.randn(N, D_out).type(dtype)
w1 = torch.randn(D_in, H).type(dtype)
w2 = torch.randn(H, D_out).type(dtype)

learning_rate = 1e-6
for t in range(500):
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)
    loss = (y_pred - y).pow(2).sum()

    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = x.t().mm(grad_h)
```

Tip: 在pytorch中使用GPU运算的话需要把数据类型设置为 **torch.cuda.FloatTensor**

PyTorch: Autograd

A PyTorch **Variable** is a node in a computational graph

x.data is a Tensor

x.grad is a Variable of gradients (same shape as x.data)

x.grad.data is a Tensor of gradients

```
import torch
from torch.autograd import Variable

N, D_in, H, D_out = 64, 1000, 100, 10
x = Variable(torch.randn(N, D_in), requires_grad=False)
y = Variable(torch.randn(N, D_out), requires_grad=False)
w1 = Variable(torch.randn(D_in, H), requires_grad=True)
w2 = Variable(torch.randn(H, D_out), requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    if w1.grad: w1.grad.data.zero_()
    if w2.grad: w2.grad.data.zero_()
    loss.backward()

    w1.data -= learning_rate * w1.grad.data
    w2.data -= learning_rate * w2.grad.data
```

定义变量的时候将参数requires_grad 设置为 True pytorch会自动算其梯度

PyTorch: Pretrained Models

Super easy to use pretrained models with torchvision
https://github.com/pytorch/vision

```
import torch
import torchvision

alexnet = torchvision.models.alexnet(pretrained=True)
vgg16 = torchvision.models.vgg16(pretrained=True)
resnet101 = torchvision.models.resnet101(pretrained=True)
```

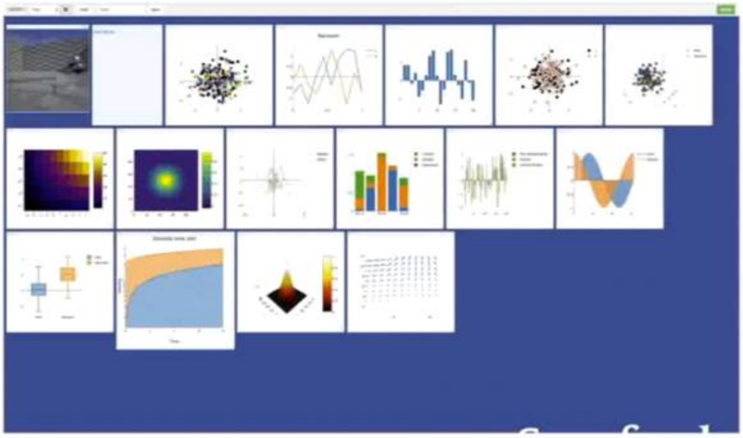
pytorch可视化

PyTorch: Visdom

Somewhat similar to TensorBoard: add logging to your code, then visualized in a browser

Can't visualize computational graph structure (yet?)

https://github.com/facebookresearch/visdom



一个pytorch的CNN网络demo：

```
#cnn
    Class Net(nn.Module):
        def__init__(self):
        super(Net,self).__init__()
        self.conv1=nn.Conv2d(3,6,5)
        self.pool=nn.MaxPool2d(2,2)
        self.conv2=nn.Conv2d(6,16,5)
        self.fc1=nn.Linear(16*5*5,120)
        self.fc2=nn.Linear(120,84)
        self.fc3=nn.Linear(84,10)

        defforward(self,x):
        x=self.pool(F.relu(self.conv1(x)))
        x=self.pool(F.relu(self.conv2(x)))
        x=x.view(-1,16*5*5)
        x=F.relu(self.fc1(x))
```

x=F.relu(self.fc2(x))
x=self.fc3(x)
Return x

#损失函数

net=Net()
criterion=nn.CrossEntropyLoss()
optimizer=optim.SGD(net.parameters(),lr=0.001,momentum=0.9)