Pytorch Notebook

> 安装Anaconda，里面涵盖numpy, pandas, matplotlib等库
>
> 在Prompt下执行conda list命令可列出所有已安装库

> 安装Pytorch，在Pytorch官网首页选择对应安装命令在Prompt下安装:

```
conda install pytorch torchvision torchaudio cpuonly -c pytorch
```

> Jupyter下运行如下命令确保安装成功：

```
import torch
import torch.nn as nn
from torchvision.transforms import Compose

nn.Linear(in_features=10, out_features=10)
```

张量

> 数组维度超过2称之为：张量（Tensor）

> 张量数据类型
>
> torch数据类型总共8种（float32, float64, float16等），默认是32位浮点型：torch.FloatTensor

```
import torch

# 查看张量数据类型

torch.tensor([1.2, 3.4]).dtype                    # torch.float32

# 更改默认数据类型

torch.set_default_tensor_type(torch.DoubleTensor)
torch.tensor([1.2,3.4]).dtype                     # torch.float64

# 类型转换
```

```
a = torch.tensor([1.2,3.4])
print(a.dtype)                                  # torch.float64
print(a.int().dtype)                            # torch.int32
print(a.long().dtype)                           # torch.int64
print(a.float().dtype)                          # torch.float32

# 获取默认的数据类型

torch.get_default_dtype()                       # torch.float64
```

张量的生成

torch.tensor()方法

```
# 列表生成张量

a = torch.tensor([[1.0, 1.0], [2, 2]])

# 获取张量的维度、大小、包含的元素数量

a.shape                                         # torch.Size([2, 2])
a.size()                                        # torch.Size([2, 2])
a.numel()                                       # 4

# 指定数据类型和是否需要梯度计算（只有浮点类型可以计算梯度）

b = torch.tensor((1, 2, 3), dtype=torch.float32, requires_grad=True)

# 针对b计算sum(b^2)在每个元素上的梯度大小

y = b.pow(2).sum()
y.backward()
b.grad                                          # tensor([2., 4., 6.])
```

torch.Tensor()方法

```
# 根据已有数据创建张量

a = torch.Tensor([1, 2, 3, 4])                  # tensor([1., 2., 3., 4.])

# 建立指定大小的张量

b = torch.Tensor(2, 3)                          # tensor([[0., 0., 0.],
                                                #         [0., 0., 0.]])

# 建立维度相同、性质相似的类似张量

torch.ones_like(b)                              # tensor([[1., 1., 1.],
                                                #         [1., 1., 1.]])
```

```
torch.zeros_like(b)                                 # tensor([[0., 0., 0.],
                                                                [0., 0., 0.]])
torch.rand_like(b)                                  #
tensor([[0.9601,0.0205,0.9121],
       [0.9093, 0.5951, 0.6884]])

# 根据已有张量建立同数据类型的新张量

c = [[1, 2], [3, 4]]
c = b.new_tensor(c)                                 # tensor([[1., 2.],[3., 4.]])

# 获取其他数据内容的同类型新张量（注意这里b本身不变）

b.new_full((3, 3), fill_value = 1)                  # 全1. 3x3
b.new_zeros((3, 3))                                 # 全0. 3x3
b.new_ones((3, 3))                                  # 全1. 3x3
b.new_empty((3, 3))                                 # 全0. 3x3
```

张量和numpy数组互相转换

```
import numpy as np

# np转换为tensor

f = np.ones((3, 3))                                 # np默认数据类型64位所以转换后也是

# 使用torch.as_tensor()

fTensor = torch.as_tensor(f)                        # tensor([[1., 1., 1.],
                                                                [1., 1., 1.],
                                                                  [1., 1., 1.]],


dtype=torch.float64)

# 使用torch.from_numpy()

fTensor = torch.from_numpy(f)                       # 同上

# tensor转化为np

fTensor.numpy()                                     # array([[1., 1., 1.],
                                                                [1., 1., 1.],
                                                                [1., 1., 1.]])
```

随机数生成张量

```
torch.manual_seed(123)

# 指定均值和标准差生成随机数

a = torch.normal(mean=0.0, std=torch.tensor(1.0))   # tensor(-0.1115)
```

```
# mean和std都只有一个则只生成一个随机数，std多选可生成多个随机数
# 分布均值全为0，分布标准差为1、2、3、4

a = torch.normal(mean=0.0, std=torch.arange(1, 5.0))
                                        # tensor([0.1204, -0.7393, -0.7213,
-4.7877])

# 分布均值为1-4，分布标准差为1-4

a = torch.normal(mean=torch.arange(1, 5.0), std=torch.arange(1, 5.0))
                                        # tensor([0.8915, 2.4207, 1.8275, 4.9399])
```

其他生成张量的函数

```
```

张量的生成

torch.tensor()生成张量

```
# 列表生成张量

a = torch.tensor([[1.0, 1.0], [2, 2]])

# 获取张量的维度、大小、包含的元素数量

a.shape                                             # torch.Size([2, 2])
a.size()                                            # torch.Size([2, 2])
a.numel()                                           # 4

# 指定数据类型和是否需要梯度计算（只有浮点类型可以计算梯度）

b = torch.tensor((1, 2, 3), dtype=torch.float32, requires_grad=True)

# 针对b计算sum(b^2)在每个元素上的梯度大小

y = b.pow(2).sum()
y.backward()
b.grad                                              # tensor([2., 4., 6.])

生成张量，获取张量的维度、大小、包含的元素数量
```

```
a = torch.tensor([[1.0, 1.0], [2, 2]])

a.shape                                                 # torch.Size([2, 2])
a.size()                                                # torch.Size([2, 2])
a.numel()                                               # 4
```

生成张量，获取张量的维度、大小、包含的元素数量

```
a = torch.tensor([[1.0, 1.0], [2, 2]])

a.shape                                                 # torch.Size([2, 2])
a.size()                                                # torch.Size([2, 2])
a.numel()                                               # 4
```

生成张量，获取张量的维度、大小、包含的元素数量

```
a = torch.tensor([[1.0, 1.0], [2, 2]])

a.shape                                                 # torch.Size([2, 2])
a.size()                                                # torch.Size([2, 2])
a.numel()                                               # 4
```

生成张量，获取张量的维度、大小、包含的元素数量

```
a = torch.tensor([[1.0, 1.0], [2, 2]])

a.shape                                                 # torch.Size([2, 2])
a.size()                                                # torch.Size([2, 2])
a.numel()                                               # 4
```