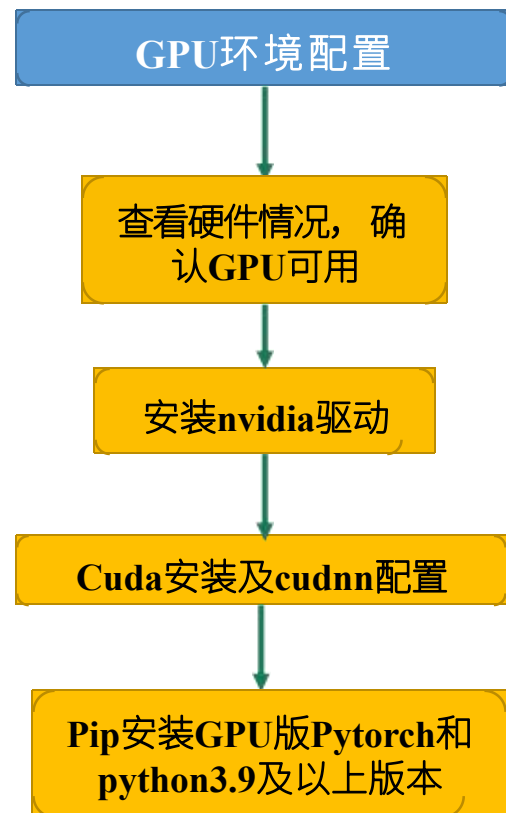
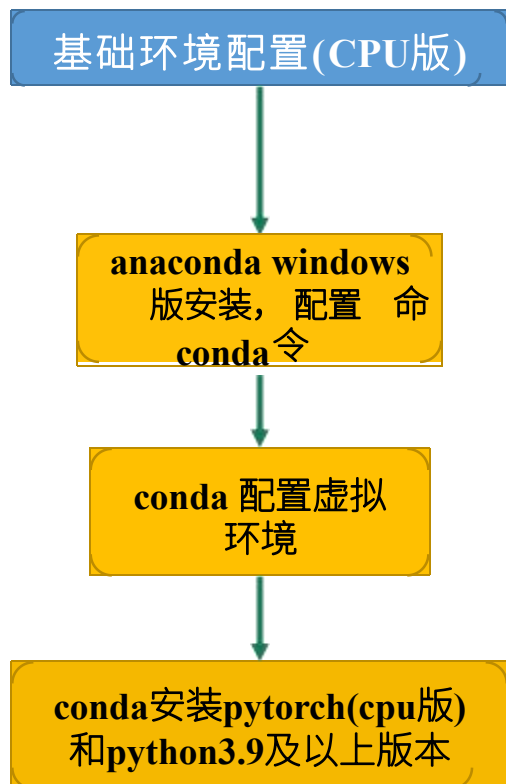




Pytorch环境搭建与基础知识

■ Python与Pytorch安装及代码运行环境配置



Pytorch环境搭建与基础知识

■ Python与Pytorch安装及代码运行环境配置

Anaconda官网: www.anaconda.com

Distribution

Free Download*

Register to get everything you need to get started on your workstation including Cloud Notebooks, Navigator, AI Assistant, Learning and more.

- ✓ Easily search and install thousands of data science, machine learning, and AI packages.
- ✓ Manage packages and environments from a desktop application or work from the command line.
- ✓ Deploy across hardware and software platforms.
- ✓ Distribution installation on Windows, MacOS, or Linux.

Provide email to download Distribution

Email Address:

☐ Agree to receive communication from Anaconda regarding relevant content, products, and services. I understand that I can revoke this consent [here](#) at any time.

By continuing, I agree to Anaconda's [Privacy Policy](#) and [Terms of Service](#).

Submit >

Stop registration



Windows

Python 3.12

64-Bit Graphical Installer (912.3M)

```
(base) C:\Users\ZJ>conda -V  
conda 24.1.2
```

```
C:\ProgramData\anaconda3  
C:\ProgramData\anaconda3\Scripts  
C:\ProgramData\anaconda3\Library\mingw-w64\bin  
C:\ProgramData\anaconda3\Library\bin
```

环境变量配置



Pytorch环境搭建与基础知识

■ Python与Pytorch安装及代码运行环境配置

pytorch官网: <https://pytorch.org/>

PyTorch Build	Stable (2.7.0)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	CUDA 11.8	CUDA 12.6	CUDA 12.8	ROCm 6.3
Run this Command:	pip3 install torch torchvision torchaudio			

配置环境

```
(base) C:\Users\10622>conda create -n pytorch python=3.9  
Retrieving notices: ...working...
```

检查torch是否安装成功

```
>>> import torch  
>>> torch.__version__  
'2.4.0+cpu'
```

```
(pytorch) C:\Users\10622>python  
Python 3.9.19 (main, Mar 21 2024, 17:21:27) [MSC v.1916 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more  
>>> import torch  
>>> torch.cuda.is_available()
```



Pytorch环境搭建与基础知识

■ 张量创建和基本操作

□ 张量创建

从python列表或Numpy数组创建张量

```
import torch
import numpy as np

#从列表创建张量
tensor_from_list = torch.tensor([2, 3, 4])

#从numpy数组创建张量
numpy_array = np.array([4,5,6])
tensor_from_numpy = torch.tensor(numpy_array)

print(tensor_from_list)
print(tensor_from_numpy)
```

```
tensor([2, 3, 4])
tensor([4, 5, 6], dtype=torch.int32)
```

使用特定值创建张量

```
#创建全零张量
zeros_tensor = torch.zeros(3,4)
#创建全一张量
ones_tensor = torch.ones(2,2)
#创建指定范围的张量
range_tensor = torch.arange(0,10,2)
#创建均匀分布的张量
uniform_tensor = torch.rand(3,3)
#创建正态分布的张量
normal_tensor = torch.randn(2,2)
```

使用特定形状的张量

```
#创建未初始化的张量
uninitialized_tensor = torch.empty(2,2)
#创建与现有张量相同形状的张量
like_tensor = torch.ones_like(zeros_tensor)
```



Pytorch环境搭建与基础知识

■ 张量创建和基本操作

□ 张量的基本操作

索引和切片

```
#获取张量中的特定元素
element = tensor_from_list[1]
#切片操作
sliced_tensor = tensor_from_list[1:3]
```

张量形状

```
#获取张量的形状
shape = tensor_from_list.shape
#改变张量的形状
reshape_tensor = tensor_from_list.view(1,3)
#转置张量
transposed_tensor = reshape_tensor.t()
```

数值运算

```
#加法
sum_tensor = tensor_from_list + tensor_from_numpy

#乘法
product_tensor = torch.matmul(zeros_tensor, ones_tensor)

#广播操作
broadcasted_tensor = tensor_from_list * 2
```

总结： Pytorch提供了方便、灵活且高效的工具处理张量和构建模型，因此，成为深度学习领域的**主流选择**！！！！



Pytorch环境搭建与基础知识

■ 自动求导机制与反向传播原理

自动求导

1. 张量requires_grad属性
2. 张量操作和计算图
3. 计算梯度
4. 阻止梯度追踪
5. 使用with torch.autograd.set_grad_enabled(False):
控制梯度计算
6. 使用自动求导进行优化

创建张量，可通过设置**requires_grad**属性为**True**指示跟踪张量的操作，从而构建**计算图(运算图)**，调用**backward()**计算梯度，通过**grad**属性获取梯度

```
import torch
#创建一个需要梯度的张量
x = torch.tensor([2.0, 3.0], requires_grad=True)

#张量操作和计算图
y = x + 3
z = y * y * 4
out = z.mean()

#计算梯度
out.backward()

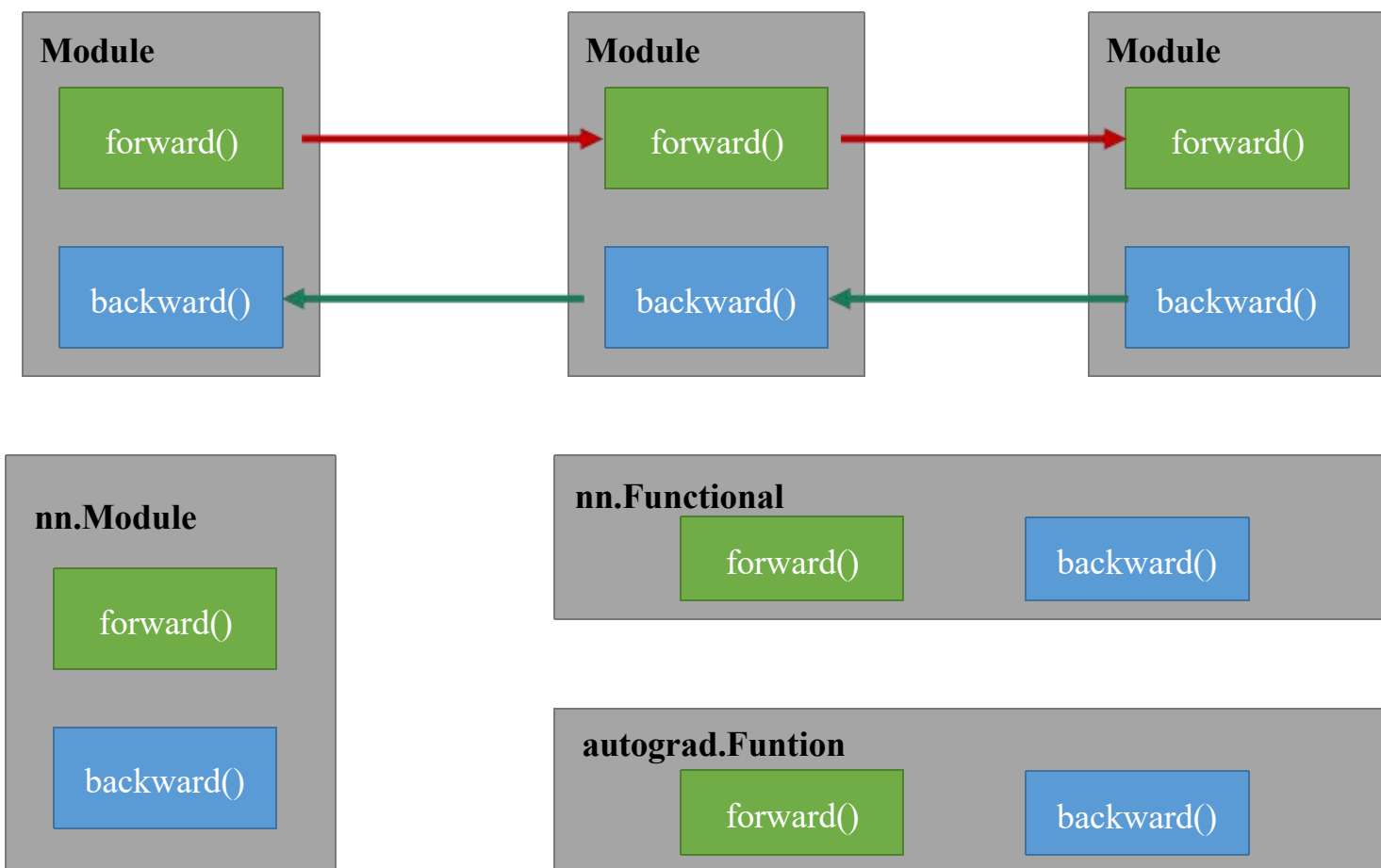
#获取梯度
print(x.grad)
```



Pytorch环境搭建与基础知识

■ 自动求导机制与反向传播原理

Pytorch计算图及自动求导架构





Pytorch环境搭建与基础知识

■ 自动求导机制与反向传播原理

阻止梯度追踪

某些情况，比如测试模式，需阻止Pytorch对梯度追踪，可使用`torch.no_grad()`方法或者在张量上使用`.detach()`方法

```
with torch.no_grad():  
    #不追踪梯度操作  
    y = x + 2  
  
#或者  
z = y.detach()
```

控制梯度计算

某些情况，需在关闭梯度计算，可使用`torch.autograd.set_grad_enabled(False)`方法

```
with torch.autograd.set_grad_enabled(False):  
    #在此处的操作不会被追踪，也不会计算梯度  
    y = x + 2
```



Pytorch环境搭建与基础知识

■ 自动求导优化示例

演示如何使用自动求导来执行优化步骤，通过反向传播计算梯度并用优化器更新参数

```
import torch.optim as optim

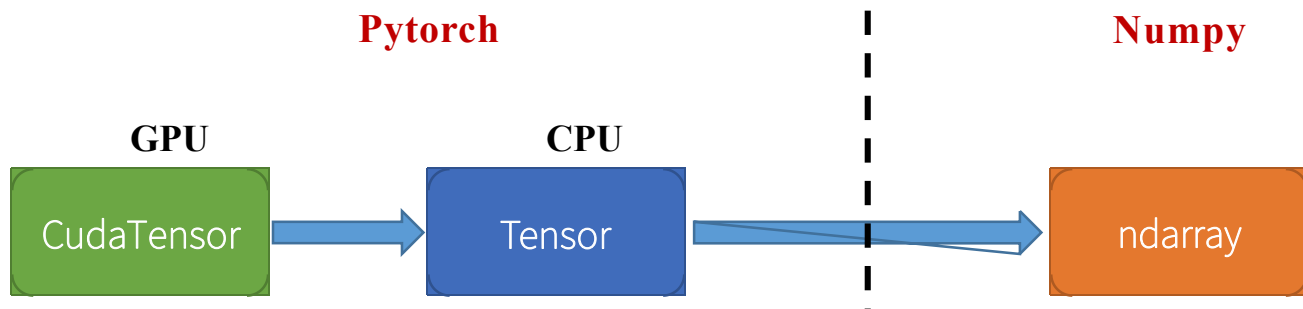
#定义一个变量并设置梯度
x = torch.tensor([2.0, 4.0], requires_grad= True)
#定义优化器
optimizer = optim.SGD([x], lr = 0.01)
#循环中执行优化步骤
for _ in range(100):
    y = x + 3
    loss = y[0] + y[1] #自定义一个损失函数
    optimizer.zero_grad() #清零梯度
    loss.backward() #计算梯度
    optimizer.step() #更新参数

#查看优化后结果
print(x)
```

Pytorch环境搭建与基础知识

■ GPU加速与Pytorch的CUDA支持

□ Pytorch CUDA Tensor



```
import torch
```

```
cuda_data = torch.randn(2, 3).cuda()
```

```
torch_data = cuda_data.cpu()
```

```
np_data = torch_data.numpy()
```

```
print(cuda_data)
```

```
print(torch_data)
```

```
print(np_data)
```

```
cuda_data.numpy()
```

```
tensor([[ 0.0943, -0.7022,  0.6402],
        [-0.2258, -0.5805, -0.4671]], device='cuda:0')
tensor([[ 0.0943, -0.7022,  0.6402],
        [-0.2258, -0.5805, -0.4671]])
[[ 0.09434418 -0.70220006  0.64023286]
 [-0.22579136 -0.58050865 -0.46714148]]
```

```
TypeError                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 cuda_data.numpy()

TypeError: can't convert cuda:0 device type tensor to numpy. Use Tensor.cpu() to copy the tensor to host memory first.
```

Pytorch环境搭建与基础知识

■ GPU加速与Pytorch的CUDA支持

□ Pytorch与多GPU

nvidia-smi



Windows 将根据你所输入的名称, 为你打开相应的程序、文件夹、文档或 Internet 资源。

打开(Q):

cmd

C:\Users\HP>nvidia-smi

单卡

NVIDIA-SMI 522.06				Driver Version: 522.06			CUDA Version: 11.8		
GPU	Name	TCC/WDDM	Bus-Id	Disp.A	Volatile	Uncorr.	ECC		
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute	M.		
0	NVIDIA RTX A5000	WDDM	00000000:01:00:0	Off	0%	Default	N/A		
30%	33C	P8	8W / 230W	237MiB / 24564MiB					
Processes:									
GPU	GI	CI	PID	Type	Process name	GPU Memory	Usage		
	ID	ID							
0	N/A	N/A	19508	C	...da\envs\LYJenv\python.exe	N/A			

多卡

NVIDIA-SMI 418.39				Driver Version: 418.39			CUDA Version: 10.1		
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC		
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute	M.		
0	Tesla P40	On	00000000:00:0A:0	Off	100%	Default	0		
N/A	83C	P0	140W / 250W	10565MiB / 22919MiB					
1	Tesla P40	On	00000000:00:0B:0	Off	99%	Default	0		
N/A	69C	P0	142W / 250W	10441MiB / 22919MiB					
2	Tesla P40	On	00000000:00:0C:0	Off	99%	Default	0		
N/A	67C	P0	178W / 250W	11739MiB / 22919MiB					
3	Tesla P40	On	00000000:00:0D:0	Off	99%	Default	0		
N/A	65C	P0	159W / 250W	12701MiB / 22919MiB					
4	Tesla P40	On	00000000:00:0E:0	Off	100%	Default	0		
N/A	59C	P0	151W / 250W	13765MiB / 22919MiB					
5	Tesla P40	On	00000000:00:0F:0	Off	100%	Default	0		
N/A	59C	P0	142W / 250W	13673MiB / 22919MiB					
6	Tesla P40	On	00000000:00:10:0	Off	100%	Default	0		
N/A	61C	P0	147W / 250W	13673MiB / 22919MiB					
7	Tesla P40	On	00000000:00:11:0	Off	100%	Default	0		
N/A	58C	P0	137W / 250W	13673MiB / 22919MiB					

nn.Module

```
net = torch.nn.DataParallel(model, device_ids=[0,1,2])
output = net(input_data)
```

```
a = torch.randn(2, device=cuda0)
b = torch.randn(2).to(cuda0)
f = torch.randn(2).cuda(cuda0)
```



Pytorch环境搭建与基础知识

■ Pytorch与Numpy的数据转换及常用API对应关系

NumPy API	PyTorch API	描述
<code>numpy.array()</code>	<code>torch.tensor()</code>	创建数组或张量
<code>numpy.zeros()</code>	<code>torch.zeros()</code>	创建一个全0的数组/张量
<code>numpy.ones()</code>	<code>torch.ones()</code>	创建一个全1的数组/张量
<code>numpy.full()</code>	<code>torch.full()</code>	用给定值填充数组/张量
<code>numpy.arange()</code>	<code>torch.arange()</code>	生成一个范围的1维数组/张量
<code>numpy.linspace()</code>	<code>torch.linspace()</code>	在指定区间内生成等间隔的值
<code>numpy.eye()</code>	<code>torch.eye()</code>	创建一个对角线为1的2维数组/张量
<code>numpy.random.randn()</code>	<code>torch.randn()</code>	从标准正态分布生成随机数
<code>numpy.random.rand()</code>	<code>torch.rand()</code>	从均匀分布[0, 1)生成随机数
<code>numpy.sum()</code>	<code>torch.sum()</code>	计算数组/张量的总和
<code>numpy.mean()</code>	<code>torch.mean()</code>	计算数组/张量的平均值
<code>numpy.max()</code>	<code>torch.max()</code>	计算数组/张量的最大值
<code>numpy.min()</code>	<code>torch.min()</code>	计算数组/张量的最小值
<code>numpy.reshape()</code>	<code>torch.reshape()</code>	重新塑形数组/张量

<code>numpy.transpose()</code>	<code>torch.t()</code> 或 <code>torch.transpose()</code>	转置矩阵
<code>numpy.expand_dims()</code>	<code>torch.unsqueeze()</code>	增加数组/张量的维度
<code>numpy.squeeze()</code>	<code>torch.squeeze()</code>	删除单维度的数组/张量
<code>numpy.vstack()</code>	<code>torch.stack()</code>	垂直堆叠数组/张量
<code>numpy.hstack()</code>	不直接对应, 但可使用 <code>torch.cat()</code>	水平堆叠数组/张量
<code>numpy.where()</code>	<code>torch.where()</code>	根据条件索引数组/张量
<code>numpy.argmax()</code>	<code>torch.argmax()</code>	找到最大值的索引
<code>numpy.argmin()</code>	<code>torch.argmin()</code>	找到最小值的索引

注意： 函数功能具有相似性，但在Pytorch和Numpy中实现方式和行为略有不同。例如 **`torch.sum()`** 默认返回一个**标量**，而 **`numpy.sum()`** 返回一个**0维数组**。Pytorch具有Numpy不具备的**自动微分功能**，因此更适用于深度学习和需要梯度下降的优化任务