为学为师 求新求实

# 深度学习应用与工程实践
# 3. NumPy & PyTorch 教程
# 3. Numpy& PyTorch Tutorial

李冰
副研究员，硕士生导师
交叉科学研究院
首都师范大学

# 上节课内容

深度学习应用

计算机视觉 (CV)

MLP | CNN | UNet | YoLo | MobileNet

自然语言处理(NLP)

RNN | LSTM/GRU | **Transformer**

生成式模型

GAN | **Diffusion** | **GPT**

工程实践

模型设计

**PyTorch**

训练关键技术

部署

新增内容

# PyTorch

- **Py**: Python, 一门高级编程语言
- **Torch**: 前身，深度学习框架
  - 替代Python中的numpy, 能利用GPU
  - 更加灵活快速
- 第一步：安装python，我们选择Anaconda/Miniconda， 环境管理工具
- 第二步：安装PyTorch

# 主要内容

- 软件安装
  - Anaconda/Miniconda安装
  - PyTorch 安装
- NumPy 基础入门
- PyTorch 基础入门

# 软件安装

- Anaconda 安装
- PyTorch 安装

# Anaconda介绍

- Anaconda
  - 便捷获取python开发所需的依赖包，对这些安装包进行管理，
  - 支持创建不同软件版本的开发环境
- **特点：**
  - 开源
  - 安装过程简单
  - 支持多操作系统，Linux, Mac, Windows。
  - 1,000+开源库
  - 维护不同的项目环境
- **安装条件**
  - 系统要求：32位或64位系统均可
  - 下载文件大小：400-500MB左右
  - 所需空间大小：3GB空间大小（Miniconda仅需400MB空间即可）

# Anaconda 安装-官网下载界面

https://www.anaconda.com/products/individual

Anaconda Installers

| Windows | MacOS | Linux |
| --- | --- | --- |
| Python 3.8 | Python 3.8 | Python 3.8 |
| 64-Bit Graphical Installer (477 MB) | 64-Bit Graphical Installer (440 MB) | 64-Bit (x86) Installer (544 MB) |
| 32-Bit Graphical Installer (409 MB) | 64-Bit Command Line Installer (433 MB) | 64-Bit (Power8 and Power9) Installer (285 MB) |
| | | 64-Bit (AWS Graviton2 / ARM64) Installer |
| | | ...staller |

## 如何查看自己操作系统字长？

方法一在开始→运行中输入“winver”，如果您的系统是64位的，会明确标示出“x64 edition”。说明是32位，否则是64位【标明：“x64 Edition】。备注： 如果是正版的OS，可以点击“我的电脑”，右键“属性”，也可以看到版本。 2017年2月19日

```
(base) [bing@MacBook-Pro ~]$uname -a
Darwin MacBook-Pro.local 22.3.0 Darwin Kernel Version 22.3.0: Mon Jan 30 20:39:46 PST 2023; root:xnu-8792.81.3~2/RELEASE_ARM64_T6020 x86_64
```

Anaconda3 2019.07 (64-bit) Setup

ANACONDA

**Choose Install Location**

Choose the folder in which to install Anaconda3 2019.07 (64-bit).

Setup will install Anaconda3 2019.07 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder

C:\Users\builder\Anaconda3

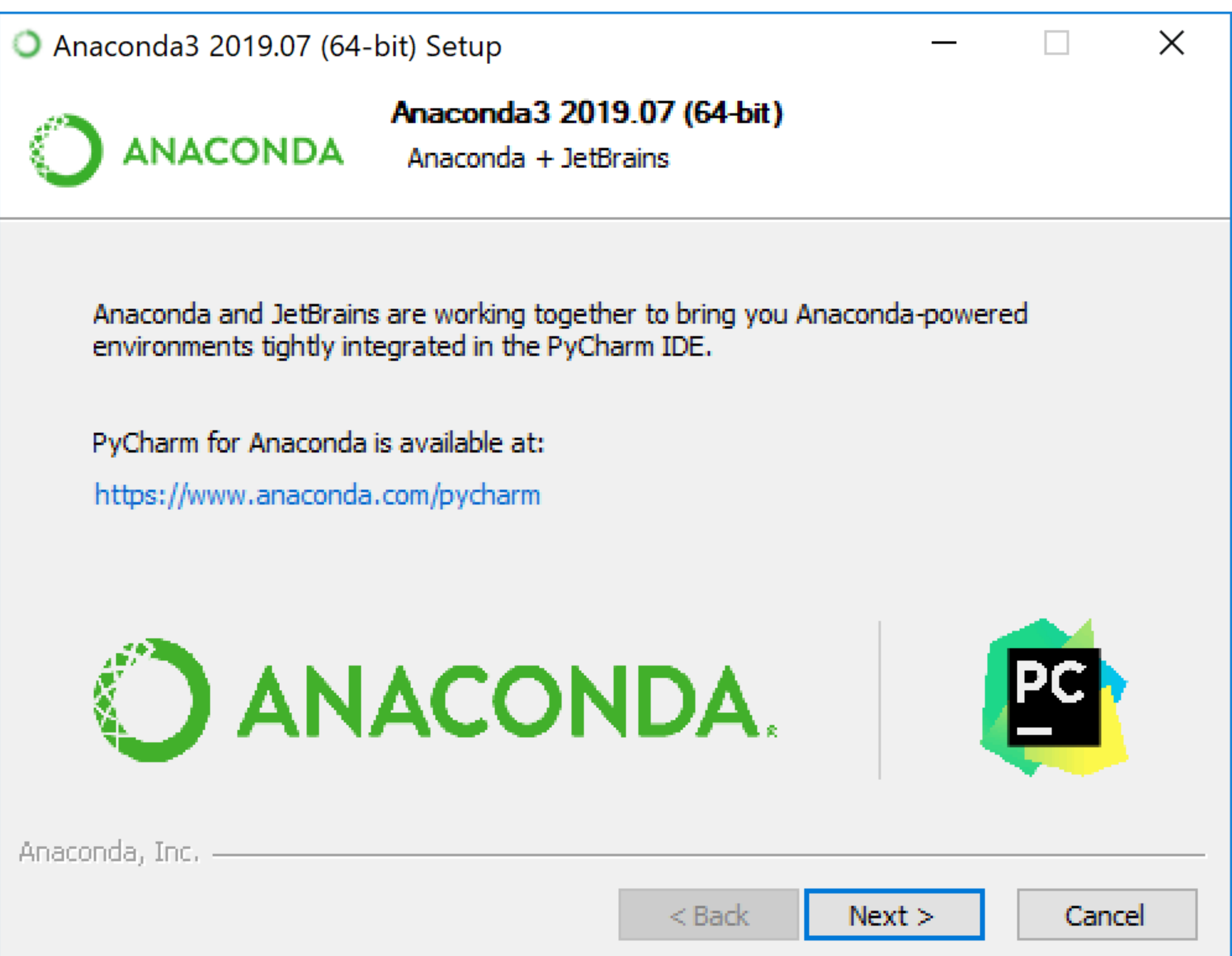
Browse...

Space required: 2.9GB

Space available: 38.1GB

Anaconda, Inc.

< Back | Next > | Cancel


Anaconda3 2019.07 (64-bit) Setup
ANACONDA
Anaconda3 2019.07 (64-bit)
Anaconda + JetBrains
Anaconda and JetBrains are working together to bring you Anaconda-powered environments tightly integrated in the PyCharm IDE.
PyCharm for Anaconda is available at:
https://www.anaconda.com/pycharm
ANACONDA
PC
Anaconda, Inc.
< Back
Next >
Cancel

Anaconda3 2019.07 (64-bit) Setup

ANACONDA.

## Thanks for installing Anaconda3!

Anaconda is the most popular Python data science platform.

Share your notebooks, packages, projects and environments on Anaconda Cloud!

- [x] Learn more about Anaconda Cloud
- [x] Learn how to get started with Anaconda

< Back | Finish | Cancel

# Anaconda安装-验证安装成功

Filters

Best match

Anaconda Navigator
Desktop app

Folders

anaconda_navigator - in site-packages
anaconda_navigator - in site-packages
anaconda_navigator-1.9.7-py3.7.egg-info - in site-packages
anaconda_navigator-1.9.7-py3.7.egg-info - in site-packages

Documents

anaconda-navigator-1.9.7-py37_0
anaconda-navigator - in Menu
anaconda-navigator - in Menu

Search the web

anaconda navigator - See web results

anaconda navigator

Filters

Best match

Anaconda Prompt
Desktop app

Search the web

anaconda prompt - See web results
anaconda prompt **commands**
anaconda prompt **download**
anaconda prompt **change directory**
anaconda prompt **not working**
anaconda prompt **change environment**
anaconda prompt **install**
anaconda prompt **location**

anaconda prompt

# Anaconda 安装-官网下载界面

https://www.anaconda.com/products/individual

Anaconda Installers

Python 3.8

64-Bit Graphical Installer (477 MB)

32-Bit Graphical Installer (409 MB)

MacOS

Python 3.8

64-Bit Graphical Installer (440 MB)

64-Bit Command Line Installer (433 MB)

Linux

Python 3.8

64-Bit (x86) Installer (544 MB)

64-Bit (Power8 and Power9) Installer (285 MB)

64-Bit (AWS Graviton2 / ARM64) Installer (413 M)

64-bit (Linux on IBM Z & LinuxONE) Installer (292 M)

# Anaconda 安装-命令行

- **Linux**

```
1. wget https://repo.anaconda.com/archive/Anaconda3-2019.07-Linux-x86_64.sh
2. chmod +x Anaconda3-2019.07-Linux-x86_64.sh
3. ./Anaconda3-2019.07-Linux-x86_64.sh
```

- **macOSX**

```
1. wget https://repo.anaconda.com/archive/Anaconda3-2019.07-MacOSX-x86_64.sh
2. chmod +x Anaconda3-2019.07-MacOSX-x86_64.sh
3. ./Anaconda3-2019.07-MacOSX-x86_64.sh
```

- **安装结束后激活 conda 环境**

**macOSX**

```
source ~/.bash_profile.
```

**Linux**

```
source ~/.bashrc
```

# Anaconda安装-conda 的用法

- 1.获取版本号　　conda --version　　conda -V
- 2. 获取帮助　　conda --help　　conda -h

- 3.创建环境　　conda create –n 环境名字
- 4.激活环境　　source activate 环境名字
- 5. 删除环境　　conda remove –n 环境名字 --all

创建激活环境

```
conda create -n py36 python=3.6        #该环境是 python 3.6版本的开发环境
conda activate py36
```

# 软件安装

- Anaconda 安装
- PyTorch 安装

# 安装Pytorch

- 登录官网 [pytorch.org/get-started/locally](pytorch.org/get-started/locally)，

pytorch.org/get-started/locally/

PyTorch | Get Started | Ecosystem | Mobile | Blog | Tutorials | Docs | Resources | GitHub

Shortcuts
- Prerequisites
  - macOS Version
  - Python
  - Package Manager
- Installation
  - Anaconda
  - pip
- Verification
- Building from source
  - Prerequisites

| PyTorch Build | Stable (1.9.1) | Preview (Nightly) | LTS (1.8.2) | |
|---|---|---|---|---|
| Your OS | Linux | Mac | Windows | |
| Package | Conda | Pip | LibTorch | Source |
| Language | Python | | C++ / Java | |
| Compute Platform | ~~CUDA 10.2~~ | ~~CUDA 11.1~~ | ~~ROCm 4.2 (beta)~~ | CPU |
| Run this Command: | conda install pytorch torchvision torchaudio -c pytorch | | | |

输入命令 `conda install pytorch torchvision torchaudio -c pytorch`

**但是！！下载非常慢，甚至失败**

# 安装Pytorch的准备--更换conda安装源

- 更换conda安装源

1.我们会在本地目录下找到.condarc 文件

```
(base) [bing@MacBook-Pro ~]$cat ~/.condarc
channels:
  - defaults
```

提示：Windows 用户无法直接创建名为 .condarc 的文件，可先执行

```
conda config --set show_channel_urls yes
```

生成该文件之后再修改。

## 2. 更新安装源

```
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/
conda config --set show_channel_urls yes
```

```
(base) [bing@MacBook-Pro ~]$cat ~/.condarc
channels:
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/
  - defaults
show_channel_urls: true
```

# 安装Pytorch--更换conda安装源

- 添加第三方源
  - conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge/
  - conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/pytorch/
  - conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/msys2/
  - conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/bioconda/
  - conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/menpo/

# 安装PyTorch

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

提示将要下载安装的依赖包

```
The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2022.07.19 |       hecd8cb5_0         124 KB  https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
    certifi-2022.9.24          |   py39hecd8cb5_0         155 KB  https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
    conda-23.1.0               |   py39h6e9494a_0         908 KB  https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge
    ffmpeg-4.3                 |       h0a44026_0        10.1 MB  pytorch
    gnutls-3.6.13              |       h756fd2b_1         2.0 MB  https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge
    lame-3.100                 |    hb7f2c08_1003         530 KB  https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge
    nettle-3.6                 |       hedd7734_0         1.2 MB  https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge
    openh264-2.1.1             |       h8346a28_0         655 KB  https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
    openssl-1.1.1q             |       hca72f7f_0         2.2 MB  https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
    python_abi-3.9             |          2_cp39            4 KB  https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge
    pytorch-1.13.1             |          py3.9_0        71.9 MB  pytorch
    ruamel.yaml-0.17.21        |  py39ha30fb19_2          173 KB  https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge
    ruamel.yaml.clib-0.2.7     |  py39ha30fb19_1          118 KB  https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge
    torchaudio-0.13.1          |          py39_cpu         5.6 MB  pytorch
    torchvision-0.14.1         |          py39_cpu         6.1 MB  pytorch
    ------------------------------------------------------------
                                           Total:       101.7 MB

    Proceed ([y]/n)?
```

# 关键设置

- 验证PyTorch安装成功

```
(base) [bing@MacBook-Pro ~]$python
Python 3.9.13 (main, Aug 25 2022, 18:29:29)
[Clang 12.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> torch.__version__
'1.13.1'
```

**Note:**

PyTorch 版本 >= 1.1.0

**conda** update pytorch

如果升级到某一版本呢？

# 安装Pytorch—含GPU

- 安装 PyTorch
- For CUDA 9.2,

```
conda install pytorch torchvision cudatoolkit=9.2
```

- For CUDA 10.0,

```
conda install pytorch torchvision cudatoolkit=10.0
```

# 软件安装

- Anaconda的安装
  - 图形界面
  - 命令行安装
- PyTorch的安装
  - 安装PyTorch的准备
    - 更新安装源
    - 确认本地环境和安装版本
  - 创建环境 `conda create`
  - 激活环境 `source activate`
  - 安装命令 `conda install`

# 主要内容

- 软件安装
- NumPy 基础入门
- PyTorch 基础入门

# Numpy的使用

- NumPy是Python中科学计算的基础包。
  - 提供多维数组对象。
  - 用于数组快速操作的各种API，有包括数学、逻辑、形状操作、排序、选择、输入输出、离散傅立叶变换、基本线性代数，基本统计运算和随机模拟等等。
  - 核心是 *ndarray* 对象
- Numpy ndarray 对象和Pytorch Tensor对象是可以无缝转换的。

# NumPy基础入门

- 数组
- 索引
- 数学计算
- 广播
- 常用函数

# NumPy 基础入门

- 数组
  - 一系列同类型数据的集合，以 0 下标为开始进行集合中元素的索引。
  - 数组的维度也就是秩
  - 数组的形状是一个每个维度上数组的大小的元组。

```
import numpy as np                  # Import numpy 库
# a是列表.
a = [2,3,4,5]
# b 是numpy数组，值和形状与 a相同.
b = np.array([2,3,4,5])
# c 是numpy 数组，值和形状与 a相同.
c = np.array(a)
# d 是所有元素为零，形状是2×4的数组.
d = np.zeros((2,4))
# e是所有元素为零，形状与a相同.
e = np.zeros_like(a)
```

# NumPy 基础入门

- Array 形状

```
import numpy as np                  # Import numpy 库
# a 是 numpy 数组.
a = np.array([[2,3],[4,5]])
# 得到a形状.
print(a.shape)
Output: (2,2)
#  变为 1×4 数组.
a=np.reshape(a, (1,4))
print(a)
Output: array([[2, 3, 4, 5]])
```

# NumPy 基础入门

## •Array 索引

数组能够在多个维度进行切片操作.

```
import numpy as np                # Import NumPy 库
# a 是python 列表.
a = [[2,3],[4,5]]
# b 是NumPy 数组, 形状, 数值与a相同.
b = np.array([[2,3],[4,5]])
# 列表多个维度切分会报错
a[:1, :1]
TypeError: list indices must be integers or slices, not tuple

# NumPy 数值能够多个维度切分.
b[:1,:1]  # 第一个维度索引<1, 第二个维度索引<1
# Output: array([[2]])
b[0:,1:] # 第一个维度索引>=0, 第二个维度索引>=1
# Output:?
```

# NumPy 基础入门

- 布尔型索引

用最小的时间开销来索引任意的元素.

```
import numpy as np
a = np.array([[1,2], [3, 4], [5, 6]])
#寻找大于2的元素
bool_idx = (a > 2)
# 返回对应索引的布尔值.
print(bool_idx)
Output: array([[False False] [ True True] [ True True]])
print(a[bool_idx])
Output: array([3 4 5 6])
# 还可以用一个简洁的命令完成这个操作:
print(a[a > 2])
Output: [3 4 5 6]
```

# NumPy 基础入门

- 数学计算

大多数的计算是在数组元素上进行的，简单计算，重载python计算。

```
import numpy as np
# 初始化两个数组
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
# 加法. ‘+’ 重载.
print(x + y)
print(np.add(x, y))
Output: [[ 6.0 8.0]  [10.0 12.0]]
# 乘积; 结果是数组
print(x * y)
print(np.multiply(x, y))
Output: [[ 5.0 12.0] [21.0 32.0]]
# 平方根; 结果是数组
print(np.sqrt(x))
Output: [[ 1. 1.41421356] [ 1.73205081 2. ]]
```

# NumPy基础入门

- 广播

广播 允许不同形状的数组直接进行运算.

```
import numpy as np
# 向量v 与矩阵x的每一行相加
# 结果存在矩阵 y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v                          # v 拓展为 [[1,0,1],[1,0,1],[1,0,1],[1,0,1]]
# 利用广播，向量v 与x每行相加
print(y)
Output: [[ 2 2 4] [ 5 5 7] [ 8 8 10] [11 11 13]]
```

广播替代循环执行矩阵操作.

# NumPy基础入门

- 一些常用函数

```
import numpy as np
```

| 函数 | 功能 |
| --- | --- |
| np.concatenate | 连接两个数组 |
| np.random.random | 产生随机数组 |
| np.random.permutation | 产生随机序列 |
| np.sum/np.mean/np.std | 得到数组和/均值/方差 |
| np.random.choice | 随机从数组挑选元素 |
| np.min/np.max | 得到数组的最大和最小值 |

# 主要内容

- 软件安装
- NumPy 基础入门
- PyTorch 基础入门

# 为什么需要编程框架?

计算误差对权重 $w_{ij}$ 的偏导数是两次使用链式法则得到的：

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net_j}}\frac{\partial \mathrm{net_j}}{\partial w_{ij}}$$

在右边的最后一项中，只有加权和 $\mathrm{net_j}$ 取决于 $w_{ij}$，因此

$$\frac{\partial \mathrm{net_j}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=1}^{n} w_{kj}o_k\right) = o_i.$$

神经元 $j$ 的输出对其输入的导数就是激活函数的偏导数（这里假定使用逻辑函数）：

$$\frac{\partial o_j}{\partial \mathrm{net_j}} = \frac{\partial}{\partial \mathrm{net_j}}\varphi(\mathrm{net_j}) = \varphi(\mathrm{net_j})(1-\varphi(\mathrm{net_j}))$$

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from matplotlib import style

def SGD(samples, y, step_size=2, max_iter_count=1000):
  m, var = samples.shape
  theta = np.zeros(2)
  y = y.flatten()
  #进入循环内
  loss = 1
  iter_count = 0
  iter_list=[]
  loss_list=[]
  theta1=[]
```

**有必要将算法中的常用操作封装成组件提供给程序员，以提高深度学习算法开发效率**

但如果 $j$ 是网络中任一内层，求 $E$ 关于 $o_j$ 的导数就不太简单了。

考虑 $E$ 为接受来自神经元 $j$ 的输入的所有神经元 $L = u, v, \ldots, w$ 的输入的函数，

$$\frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(\mathrm{net}_u, \mathrm{net}_v, \ldots, \mathrm{net}_w)}{\partial o_j}$$

并关于 $o_j$ 取全微分，可以得到该导数的一个递归表达式：

$$\frac{\partial E}{\partial o_j} = \sum_{l\in L}\left(\frac{\partial E}{\partial \mathrm{net}_l}\frac{\partial \mathrm{net}_l}{\partial o_j}\right) = \sum_{l\in L}\left(\frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial \mathrm{net}_l}\frac{\partial \mathrm{net}_l}{\partial o_j}\right) = \sum_{l\in L}\left(\frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial \mathrm{net}_l}w_{jl}\right)$$

```
    h = np.dot(theta,samples[rand1].T)
    #关键点，只需要一个样本点来更新权值
    for i in range(len(theta)):
      theta[i] =theta[i] - step_size*(1/m)*(h - y[rand1])*samples[rand1,i]
    #计算总体的损失精度，等于各个样本损失精度之和
    for i in range(m):
      h = np.dot(theta.T, samples[i])
      #每组样本点损失的精度
      every_loss = (1/(var*m))*np.power((h - y[i]), 2)
      loss = loss + every_loss

    print("iter_count: ", iter_count, "the loss:", loss)
```

**算法理论复杂**

**代码实现工作量大**

# 为什么需要编程框架?

- 深度学习算法具有多层结构，每层的运算由一些基本操作构成
- 这些基本操作中存在大量共性运算，如卷积、池化、激活等。将这些共性运算操作封装起来，可以提高编程实现效率
- 面向这些封装起来的操作，硬件程序员可以基于硬件特征，有针对性的进行充分优化，使其能充分发挥硬件的效率

深度学习编程框架:将深度学习算法中的基本操作封装成一系列组件，这一系列深度学习组件，即构成一套深度学习框架

中科院计算所

# 什么是PyTorch

- Numpy的GPU 版本.
- 灵活快速的深度学习开发框架.

# PyTorch 教程

- PyTorch 基础
- 构建 DNN 模块
- 训练设置
- 例子: 动态网络
- 常用函数

# 一些重要的库

```
import torch.nn as nn                         # pytorch神经网络模型
import torch.nn.functional as F  # 函数 比如激活函数.
import torchvision.models as models   # pytorch 计算机视觉模型zoo
```

- 提供了神经网络层的实现
  - nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)
  - nn.MaxPool1d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)
- nn.Module
  - 定义的网络模型需要继承的基类/父类。

# Imports

```
import torch.nn as nn                          # pytorch神经网络模型
import torch.nn.functional as F  # 函数 比如激活函数.
import torchvision.models as models   # pytorch 计算机视觉模型zoo
```

- **import** torch.nn.functional **as** F # 函数
- F.conv2d(input, weight, bias=None, stride=1, padding=0, dilation=1, groups=1)
- F.relu(input, inplace=False)
- F.softmax(input, dim=None, _stacklevel=3, dtype=None)
- F.layer_norm(input, normalized_shape, weight=None, bias=None, eps=1e-05)
- 与nn.Conv2d的差异体现在用法上：
  - 这里显式地给出输入参数。
  - 而nn中的函数，在调用的时候，传递参数。

# Imports

```
import torch.nn.functional as F  # 函数 比如激活函数.
import torch.nn as nn                      # pytorch神经网络模型
import torchvision.models as models    # pytorch 计算机视觉模型zoo
```

- torchvision.models
- 包含alexnet、densenet、inception、resnet、squeezenet、vgg等常用的网络结构
- 提供了预训练模型
- 可以通过简单调用来读取网络结构和预训练模型。

```
import torchvision.models as models

resnet18 = models.resnet18(weights=True)
alexnet = models.alexnet(weights=True)
#如果不需要用与训练模型的参数来初始化,只需要网络结构
vgg16=models.vgg16(weights = False)
vgg16=models.vgg16()
```

```
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /Users/bing/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
```

100% 44.7M/44.7M [00:20<00:00, 2.03MB/s]　网络模型下载到本地缓存

# 数据类型：张量Tensors

- PyTorch 用 **Tensors** 存放神经网络计算过程中的 权重与激活值.
- **Tensors** 与NumPy的数组类似，另外能够用GPU实现计算加速.

# Tensors: 例子

```
import torch
# 创建一个 5x3 矩阵, 未初始化:
x = torch.empty(5, 3)
# 创建一个随机初始化的5x3 矩阵:
x = torch.rand(5, 3)
# 创建一个5x3的矩阵, 初始化为long型的零:
x = torch.zeros(5, 3, dtype=torch.long)
# 打印
print(x)
#从numpy array 创建张量 ?
#从list创建张量 ?
```

Out:
Tensor([[0,0,0], [0,0,0],
[0,0,0], [0,0,0], [0,0,0]])

```
>>> x = torch.zeros(5,3,dtype=torch.long)
>>> print(x)
tensor([[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]])
```

# Tensor 相关的操作

- 计算操作与Numpy的计算操作相同.

```
# 比如, 张量加法
torch.add(x,y)
x+y
```

- 张量变形函数 tensor.view.

```
x = torch.randn(4, 4)
y = x.view(16)
z = x.view(-1, 8) # the size -1 is inferred from other dimensions
print(x.size(), y.size(), z.size())
```

Out:
torch.Size([4, 4])
torch.Size([16])
torch.Size([2, 8])

# 自动求微分：Autograd

- PyTorch 提供在张量上自动求微分的操作.

```
x = torch.ones((2, 2), requires_grad=True)
y = x + 2
z = y * y * 3
out = z.mean()
print(z, out)
# 自动求微分
out.backward()
print(x.grad)
```

Out:
tensor([[27., 27.], [27., 27.]], grad_fn=<MulBackward0>) tensor(27., grad_fn=<MeanBackward0>)

```
tensor([[27., 27.],
        [27., 27.]], grad_fn=<MulBackward0>)
 tensor(27., grad_fn=<MeanBackward0>)
```

Out:
tensor([[4.5000, 4.5000], [4.5000, 4.5000]])

```
tensor([[4.5000, 4.5000],
        [4.5000, 4.5000]])
```

# 构建块:例子

- 构建 LeNet-5 网络

```
import torch.nn as nn
import torch.nn.functional as F
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        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)

    def forward(self, x):
        out = F.relu(self.conv1(x))
        out = F.max_pool2d(out, 2)
        out = F.relu(self.conv2(out))
        out = F.max_pool2d(out, 2)
        out = out.view(out.size(0), -1)
        out = F.relu(self.fc1(out))
        out = F.relu(self.fc2(out))
        out = self.fc3(out)
        return out
```

import
引入关键的库

__init__
定义网络层
初始化权重

forward
构建网络
定义网络层之间的连接和张量数据流

# 构建块: 模板

- **定制网络**

```
import torch.nn as nn
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        …

    def forward(self, x):
        …
```

- 继承父类 **nn.Module** 以被能在pytorch的深度神经网络中使用.
- 变量在__init__中定义和初始化
- 必须有一个forward函数. 计算图在forward函数中构建.

# 构建块: 模块化

```
import torch.nn as nn
class Block(nn.Module):
    def __init__(self):
        super(Block, self).__init__()
        ...
        pass

    def forward(self, x):
        ...
        pass


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.layer = Block()
        ...
        pass

    def forward(self, x):
        output = self.layer()
        ...
        return output
```

用super来继承父类.

只要用这个模板创建模块，我们可以用之前定义的模块创建更大的module

Note:
模块化在实践中非常好用, 尤其是对重复结构的网络. (如VGG-16, ResNet等) 另外, 模块化减少了调试难度，代码更易读

# nn.Module

- 模块(Module）是所有神经网络模型的基类。
- 创建模型的时候继承这个类.
- nn.Module 还有很多非常有趣的属性，大家在使用的过程中可以慢慢了解。

# 数据处理

- PyTorch 提供用于数据处理的函数.

首先, import 必要模块来做变换:

```
import torchvision.transforms as transforms
```

**用torchvision.transforms中的函数做数据预处理和数据扩充**

| **函数** | **功能** |
| --- | --- |
| torchvision.transforms.ToTensor | NumPy 数组转换为 torch张量。 所有数据标准化到 [0,1]范围 |
| torchvision.transforms.Normalize | 用给出的均值和标准层来标准化输入. |
| torchvision.transforms.RandomHorizontalFlip | 对输入图片做一个随机水平翻转，用来扩充数据. |
| torchvision.transforms.RandomCrop | 随机剪裁都目标大小. 先在图片边界填充零，再随机剪裁到目标大小. |

# 数据处理

- 例子: 对CIFAR-10 数据集的数据处理

```
import torchvision.transforms as transforms
# 训练数据集的数据变换
transform= transforms.Compose([
    transforms.ToTensor(), # 转换成tensor
    transforms.Normalize(mean=(0.4914, 0.4822, 0.4465), std=(0.2023, 0.1994, 0.2010)),
         #Normalized_image=(image-mean)/std
])
#例子
from PIL import Image
import matplotlib.pyplot as plt
img = Image.open('test.jpg’)
img_transform=transform(img)

data = np.random.rand(10, 10,3)
img = data *255
img_transform = transform(img)
plt.imshow(img)
plt.show()
```

# 数据装载

- PyTorch 为大多CV任务提供数据装载器.
- CIFAR-10数据集的例子

```
import torchvision
transform_train= transforms.Compose([
    transforms.ToTensor(), # 转换成tensor
    transforms.Normalize(mean=(0.4914, 0.4822, 0.4465), std=(0.2023, 0.1994, 0.2010)),
])
transform_test= transforms.Compose([
    transforms.ToTensor(), # 转换成tensor
    transforms.Normalize(mean=(0.4914, 0.4822, 0.4465), std=(0.2023, 0.1994, 0.2010)),
])
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True,
transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=256, shuffle=True,
num_workers=16)
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True,
transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=100, shuffle=False,
num_workers=16)
```

# 数据装载

- 这里给出ImageNet的例子

## TORCHVISION.DATASETS

All datasets are subclasses of `torch.utils.data.Dataset` i.e, they have `__getitem__` and `__len__` methods implemented. Hence, they can all be passed to a `torch.utils.data.DataLoader` which can load multiple samples parallelly using `torch.multiprocessing` workers. For example:

```
imagenet_data = torchvision.datasets.ImageNet('path/to/imagenet_root/')
data_loader = torch.utils.data.DataLoader(imagenet_data,
                                          batch_size=4,
                                          shuffle=True,
                                          num_workers=args.nThreads)
```

# 损失函数

- 大部分问题, 用交叉熵损失函数（cross-entropy 损失）

```
import torch.nn as nn
criterion = nn.CrossEntropyLoss()
```

- 建议看一下Pytorch的实现源码. 损失函数用两个输入参数

```
class CrossEntropyLoss(_WeightedLoss):
    def __init__(self, weight=None, size_average=None, ignore_index=-100,
                 reduce=None, reduction='mean'):
        super(CrossEntropyLoss, self).__init__(weight, size_average, reduce, reduction)
        self.ignore_index = ignore_index

    def forward(self, input, target):
        return F.cross_entropy(input, target, weight=self.weight,
                               ignore_index=self.ignore_index, reduction=self.reduction)
```

- 所有,正确的调用方式是

```
loss = nn.CrossEntropyLoss(outputs, targets)
#OR
loss = criterion(outputs, targets)
```

Note:
在任何不确定的时候就去看
pytorch 文档和使用教程

# 训练方法 Optimizer

Optimizer是训练方法. optimizers 在**torch.optim包中定义**.

| **函数名** | **功能** |
| --- | --- |
| torch.optim.Adadelta | Implements Adadelta. |
| torch.optim.Adagrad | Implements Adagrad. |
| torch.optim.Adam | Implements Adam. |
| torch.optim.ASGD | Implements Averaged Stochastic Gradient Descent. |
| torch.optim.RMSprop | Implements RMSprop. |
| torch.optim.SGD | 随机梯度下降(optionally with momentum). |

**大部分情况用torch.optim.SGD**

# 训练方法 Optimizer

- Optimizer 在计算图之外、训练开始前定义好.
- 比如 我们定义并初始化好一个网络**net**.
- **定义一个 优化器用带动量的随机梯度下降 （SGD with momentum）**

```
import torch.optim as optim

optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9, weight_decay=1e-4)
```

- 为了取得最好的性能，推荐用默认值.
- 以后我们会介绍如何调节参数。

# Optimizer

- 在训练方法中调节学习率learning rate

```
import torch.optim as optim
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9, weight_decay=1e-4)
new_lr = 0.1
for param_group in optimizer.param_groups:
    param_group['lr'] = new_lr
```

- 用torch.optim.lr_scheduler方法

```
import torch.optim as optim
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9, weight_decay=1e-4)
# Apply 0.1 learning rate decay for every 30 epochs.
optimizer = optim.lr_scheduler.StepLR(optimizer,step_size=30, gamma=0.1)
```

# 例子: 动态网络

- 创建一个网络，深度动态变化
  - 前向时随机选择0-3隐藏层.
  - 隐藏层的权重是共享的

# 例子: 动态网络

- Import基本的库

```
import torch
import random
```

- 对于更复杂的网络, 推荐import 下面这些库:

```
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torch.backends.cudnn as cudnn #if you have GPU installed
import torchvision
import torchvision.transforms as transforms
```

# 例子: 动态网络

- 创建动态网络模块

```
class DynamicNet(torch.nn.Module):
    def __init__(self, D_in, H, D_out):
        super(DynamicNet, self).__init__()
        self.input_linear = torch.nn.Linear(D_in, H)
        self.middle_linear = torch.nn.Linear(H, H)
        self.output_linear = torch.nn.Linear(H, D_out)

    def forward(self, x):
        h_relu = self.input_linear(x).clamp(min=0)
        for _ in range(random.randint(0, 3)):
            h_relu = self.middle_linear(h_relu).clamp(min=0)
        y_pred = self.output_linear(h_relu)
        return y_pred
```

重要：初始化父类.

初始化网络层和权重设置

指定网络连接关系；
随机选取0-3层

# 例子: 动态网络

- 准备一些简单数据

```
# N 是 batch 大小; D_in 是输入维度;
# H 是隐藏层维度; D_out 是 输入维度.
N, D_in, H, D_out = 64, 1000, 100, 10

# 创建随机Tensors 作为输入和输出
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
```

# 例子: 动态网络

- 创建模型实例损失函数 和训练方法操作

```
# 实例化刚才定义好的类，我们就得到了这个模型
model = DynamicNet(D_in, H, D_out)

# 构建损失函数和优化器. 均方差做loss function, SGD momentum做优化器
criterion = torch.nn.MSELoss(reduction='sum')
optimizer = torch.optim.SGD(model.parameters(), lr=1e-4, momentum=0.9)
```

数据并没有做标准化，用小一些的学习率1e-4避免梯度爆炸. 通常,标准化数据，SGD momentum方法的学习率选择1e-2.

# 例子: 动态网络

- 开始前向和后向传播

```
for t in range(500):
    # 前向传播:传入x给模型, 计算预测的y
    y_pred = model(x)

    #计算并打印
    loss = criterion(y_pred, y)
    print(t, loss.item())

    #初始化梯度, 计算后向, 更新网络.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

# 进阶PyTorch课题

- 训练/Eval 模式
- GPU上训练
- 模型加载和存储
- 数据并行
- 学习率调整器

# 训练/评估模式

- 一些网络层(比如dropout, batch normalization)在训练和评估模式下的计算完全不同.
  - 所有一定要设置正确的模式.

```
import torch
…
#我们已经定义好的类进行实例化就得到了模型
model = DynamicNet(D_in, H, D_out)
…
# 在开始训练前设置train 模式
model.train()
… # Training code
# 在开始评估之前设置eval 模式
model.eval()
… # Evaluation code
```

# GPU 上训练

- GPU 训练速度有明显提升相比CPU.

**在GPU上部署模型**

```
import torch
# 先检查可用的GPU
device = 'cuda' if torch.cuda.is_available() else 'cpu'
# 我们已经定义好的类进行实例化就得到了模型
model = DynamicNet(D_in, H, D_out)
# 把模型复制到到CUDA设备 这步很重要.
model.to(device)
```

# GPU 上训练

- 训练的输入数据也要复制到GPU

```
for t in range(500):
    # 复制输入到GPU. 非常重要.
    x, y = x.to(device), y.to(device)
    # 前向计算: 把x 传给模型, 计算预测的y
    y_pred = model(x)

    # Compute and print loss
    loss = criterion(y_pred, y)
    print(t, loss.item())

    # 零梯度, 执行后向传播, 根据梯度更新网络.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

# 模型存储/加载

- 存储/加载完整的网络

```
import torch
#我们已经定义好的类进行实例化就得到了模型
model = DynamicNet(D_in, H, D_out)
# 配置训练 优化器和训练过程
…
#模型保存
torch.save(model, “dynamic_net.pth”)
```

Note:
模型序列化为pickle 对象.
序列化数据，包含了对象的类和目录结构等信息。当数据反序列化时， 模块、类和函数会自动按需导入进来。

# 模型存储/加载

- 模型加载

```
import torch
# 加载
model = torch.load(''dynamic_net.pth'')
```

Note:模型序列化为pickle 对象.
序列化数据，包含了对象的类和目录结构等信息。当数据反序列化时， 模块、类和函数会自动按需导入进来。

# 模型存储/加载

- 存储模型的权重

```
import torch
#我们已经定义好的类进行实例化就得到了模型
model = DynamicNet(D_in, H, D_out)
# 配置训练优化器和训练过程
…
# 保存权重参数
torch.save(model.state_dict(), ''dynamic_net.pt'')
```

Note: 这个方法更好
权重参数不依赖特定类或者代码

# 模型存储/加载

- 加载模型的权重参数

```
import torch
#我们已经定义好的类进行实例化就得到了模型
model = DynamicNet(D_in, H, D_out)
# 配置训练 优化器和训练过程
…
# 加载权重参数
model.load_state_dict(torch.load("dynamic_net.pt"))
```

Note: 这个方法更好。
权重参数不依赖特定类、模块

# 数据并行

- 多个GPU卡获得更多的加速

```
import torch
# 检查GPU 是否可用
device = 'cuda' if torch.cuda.is_available() else 'cpu'
# 实例化模型
model = DynamicNet(D_in, H, D_out)
# 模型复制到GPU. 非常重要.
model.to(device)
# 用数据并行语法.
model = torch.nn.DataParallel(model)
```

# 学习率调度/调整

- torch.optim.lr_scheduler库来调整学习率.

例子: 指定学习率指数级衰减

CLASS torch.optim.lr_scheduler.StepLR(*optimizer*, *step_size*, *gamma=0.1*, *last_epoch=-1*) [SOURCE]

Sets the learning rate of each parameter group to the initial lr decayed by gamma every step_size epochs. When last_epoch=-1, sets initial lr as lr.

Parameters

- **optimizer** (*Optimizer*) – Wrapped optimizer.
- **step_size** (*int*) – Period of learning rate decay.
- **gamma** (*float*) – Multiplicative factor of learning rate decay. Default: 0.1.
- **last_epoch** (*int*) – The index of last epoch. Default: -1.

Example

```
>>> # Assuming optimizer uses lr = 0.05 for all groups
>>> # lr = 0.05     if epoch < 30
>>> # lr = 0.005    if 30 <= epoch < 60
>>> # lr = 0.0005   if 60 <= epoch < 90
>>> # ...
>>> scheduler = StepLR(optimizer, step_size=30, gamma=0.1)
>>> for epoch in range(100):
>>>     train(...)
>>>     validate(...)
>>>     scheduler.step()
```

学习率对训练的成功非常重要，之后的课程会具体介绍

# 参考资料

- **NumPy 教程**

http://cs231n.github.io/python-numpy-tutorial/

- **PyTorch 文档**

https://pytorch.org/docs/stable/index.html