DeepLearning

Keras

https://keras-cn.readthedocs.io/en/latest/

Keras的核心数据结构是“模型”,Keras的底层库使用Theano或TensorFlow(“符号主义”的库)

与传统的Python代码区别？

符号主义的计算首先定义各种变量，然后建立“计算图”，计算图规定了各个变量之间的计算关系。建立好的计算图需要编译已确定其内部细节，然而，此时的计算图还是一个“空壳子”，里面没有任何实际的数据，只有当你把需要运算的输入放进去后，才能在整个模型中形成数据流，从而形成输出值

深度学习的优化算法，说白了就是梯度下降。每次的参数更新有两种方式：

Batch gradient descent（批梯度下降）：遍历全部数据集算一次损失函数，然后算函数对各个参数的梯度，更新梯度。缺点：计算量开销大，计算速度慢，不支持在线学习

stochastic gradient descent（随机梯度下降）：速度快，但收敛性不好，可能在最优点附近晃来晃去，hit不到最优点。两次参数的更新也有可能互相抵消掉，造成目标函数震荡的比较剧烈

mini-batch gradient decent（小批的梯度下降）：数据分为若干批，按批来更新参数，批中的一组数据共同决定了本次梯度的方向，下降起来就不容易跑偏，减少了随机性

张量可以看作是向量、矩阵的自然推广，我们用张量来表示广泛的数据类型

0阶张量，即标量，也就是一个数

1阶张量，也就是一个向量

2阶张量，也就是一个矩阵

3阶张量，一个立方体

张量的阶数有时候也称为维度，或者轴

Ubuntu 16.04 LTS是Nvidia官方以及绝大多数深度学习框架默认开发环境

Theano: a compiler for mathematical expressions in Python

TensorFlow: a python library for fast numerical computing

Keras: library addresses these concerns by providing a wrapper for both Theano and Tensorflow

scikit-learn library：general purpose machine learning framework in Python built on top

of SciPy

application checkpointing

dropout

convolutional neural networks

安装

# 系统升级

$ sudo apt update

$ sudo apt upgrade

# 安装python基础开发包

$ sudo apt install -y python-dev python-pip python-nose gcc g++ git gfortran vim

# 安装运算加速库

$ sudo apt install -y libopenblas-dev liblapack-dev libatlas-base-dev

# 安装CUDA开发环境

$ sudo dpkg -i cuda-repo-ubuntu1604-8-0-local\_8.0.44-1\_amd64.deb

$ sudo apt update

$ sudo apt install cuda

# 将CUDA路径添加至环境变量

$ sudo gedit /etc/bash.bashrc

在bash.bashrc文件中添加：

export CUDA\_HOME=/usr/local/cuda-8.0

export PATH=/usr/local/cuda-8.0/bin${PATH:+:${PATH}}

export LD\_LIBRARY\_PATH=/usr/local/cuda-8.0/lib64${LD\_LIBRARY\_PATH:+:${LD\_LIBRARY\_PATH}}

$ source gedit /etc/.bashrc

在.bashrc中添加如上相同内容

$ sudo gedit ~/.bashrc

$ nvcc -v 测试nVidia cuda版本号

Keras框架搭建

$ sudo pip install -U --pre pip setuptools wheel

$ sudo pip install -U --pre numpy scipy matplotlib scikit-learn scikit-image

$ sudo pip install -U --pre theano

$ sudo pip install -U --pre keras

注：依赖numpy，scipy, pyyaml, HDF5, h5py（可选，仅在模型的save/load函数中使用）

TensorFlow(当使用TensorFlow为后端时) or Theano(当使用Theano作为后端时), Keras默认使用TensorFlow作为后端来进行张量操作

keras$ sudo python setup.py install

or $ sudo pip install keras

$ python 验证

>>> import theano

>>> import keras

Keras环境设置

修改默认keras后端: gedit ~/.keras/keras.json

配置theano文件: gedit ~/.theanorc

[global]

openmp=False

device = gpu

floatX = float32

allow\_input\_downcast=True

[lib]

cnmem = 0.8

[blas]

ldflags= -lopenblas

[nvcc]

fastmath = True

验证keras是否安装成功

>>>import keras

加速测试

keras/examples/$ python mnist\_mlp.py

Load Data.

Define Model.

Compile Model.

Fit Model.

Evaluate Model.

Tie It All Together.

NumPy

<http://old.sebug.net/paper/books/scipydoc/numpy_intro.html>

NumPy提供了两种基本的对象：ndarray（N-dimensional array object）和 ufunc（universal function object）

$ pip install numpy

创建

# 通过给array函数传递Python的序列对象创建数组，如果传递的是多层嵌套的序列，将创建多维数组

>>> a = np.array([1, 2, 3, 4])

>>> b = np.array((5, 6, 7, 8), dtype=np.float)

>>> c = np.array([[1, 2, 3, 4],[4, 5, 6, 7], [7, 8, 9, 10]], dtype=np.complex)

c.dtype 数组元素类型

c.shape 数组尺寸

c.shape = 4,3 #原数组尺寸改变

d = a.reshape((2, 2)) #原数组尺寸不改变，但生成指定尺寸的新数组，注意a and d共享数据存储内存区域

np.arange(0, 1, 0.1) #类似python的range，指定开始值，终值和步长，注意不包括终值

np.linspace(0, 1, 12) #等差数列，包括终值

np.logspace(0, 2, 20) #等比数列

np.fromstring("abcdefgh", dtype=np.int8)

# 用C语言的二进制方式写了一组double类型的数值到某个文件中，那们可以从此文件读取相应的数据，并通过fromstring函数将其转换为float64类型的数组

def func(i):

return i%4+1

np.fromfunction(func, (10,)) #第一个参数为计算每个元素的函数，第二个参数为数组大小

存取元素

a[5]

a[3:5] #通过下标范围获取的新的数组是原始数组的一个视图。它与原始数组共享同一块数据空间

a[:5]

a[:-1]

a[1:-1:2] 第三个参数表示步长

a[::-1] 数组逆序

a[[3, 3, 1, 8]] 使用整数序列作为下标获得的数组不和原始数组共享数据空间

a[a>5] 使用布尔数组作为下标获得的数组不和原始数组共享数据空间

多维数组

NumPy采用组元(tuple)作为数组的下标

也有整数序列或布尔数组作为下标，注意是多维

内存结构



ufunc运算： 对数组的每个元素进行操作的函数

result = np.sin(a)

result = a1 + a2 类似matlab点运算

+, -, \*, /, //, \*\*, %

函数式编程

def triangle\_func(c, c0, hc):

# 数组元素函数

def trifunc(x):

x = x - int(x) # 三角波的周期为1，因此只取x坐标的小数部分进行计算

if x >= c: r = 0.0

elif x < c0: r = x / c0 \* hc

else: r = (c-x) / (c-c0) \* hc

return r

# 用trifunc函数创建一个ufunc函数，可以直接对数组进行计算, 不过通过此函数

# 计算得到的是一个Object数组，需要进行类型转换

return np.frompyfunc(trifunc, 1, 1)

y2 = triangle\_func(0.6, 0.4, 1.0)(x)

广播

# array([[ 0],

[10],

[20],

[30],

[40],

[50]])

a = np.arange(0, 60, 10).reshape(-1, 1)

# array([0, 1, 2, 3, 4])

b = np.arange(0, 5)

c = a + b

array([[ 0, 1, 2, 3, 4],

[10, 11, 12, 13, 14],

[20, 21, 22, 23, 24],

[30, 31, 32, 33, 34],

[40, 41, 42, 43, 44],

[50, 51, 52, 53, 54]])

x, y = np.ogrid[0:1:4j, 0:1:3j] 开始值：结束值： 数组长度

x =

array([[ 0. ],

[ 0.33333333],

[ 0.66666667],

[ 1. ]])

y = array([[ 0. , 0.5, 1. ]])

利用ogrid的返回值，我能很容易计算x, y网格面上各点的值，或者x, y, z网格体上各点的值

x, y = np.ogrid[-2:2:20j, -2:2:20j]

z = x \* np.exp( - x\*\*2 - y\*\*2)

文件存取

numpy.load和numpy.save函数以NumPy专用的二进制类型保存数据，这两个函数会自动处理元素类型和shape等信息

>>> np.save("a.npy", a)

>>> c = np.load( "a.npy" )

将多个数组保存到一个文件中的话，可以使用numpy.savez函数

np.savez("result.npz", a, b, sin\_array = c)

>>> r = np.load("result.npz")