Contents

[Keras 1](#_Toc473100297)

[安装 2](#_Toc473100298)

[workflow 3](#_Toc473100299)

[Scipy 4](#_Toc473100300)

[install 4](#_Toc473100301)

[numPy 4](#_Toc473100302)

[scipy 7](#_Toc473100303)

[matplotlib 13](#_Toc473100304)

[pandas 14](#_Toc473100305)

[SymPy 15](#_Toc473100306)

[CS231n Convolutional Neural Networks for Visual Recognition 18](#_Toc473100307)

[RNN 18](#_Toc473100308)

[Convolutional Neural Networks (CNNs / ConvNets) 21](#_Toc473100309)

主成分分析Principal Component Analysis，PCA

通过正交变换将一组可能存在相关性的变量转换为一组线性不相关的变量,前提是尽可能多地反映原来变量的信息

转换后的这组变量叫主成分

从原始变量中导出少数几个主成分，使它们尽可能多地保留原始变量的信息

应用场合：特征降维（比如人脸识别，200\*200图像，将灰度值作为特征，即40000维，邻近像素相关性高，可以用PCA降维，保存识别所需要的信息）

 

数据变化的主方向u1就是协方差矩阵的主特征向量，而u2是次特征向量。

白化

白化的目的就是降低输入的冗余性,即如下：

1. 特征之间相关性较低；(ii)所有特征具有相同的方差。

白化是一种数据预处理方法。 事实证明这也是一种生物眼睛(视网膜)处理图像的粗糙模型。具体而言，当你的眼睛感知图像时，由于一幅图像中相邻的部分在亮度上十分相关，大多数临近的“像素”在眼中被感知为相近的值。因此，如果人眼需要分别传输每个像素值（通过视觉神经）到大脑中，会非常不划算。取而代之的是，视网膜进行一个与ZCA中相似的去相关操作 (这是由视网膜上的ON-型和OFF-型光感受器细胞将光信号转变为神经信号完成的)。由此得到对输入图像的更低冗余的表示，并将它传输到大脑。

# 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

## workflow

Load Data.

Define Model.

Compile Model.

Fit Model.

Evaluate Model.

Tie It All Together.

# Scipy

<http://www.scipy.org/install.html>

## install

Upgrade pip

$ python -m pip install --upgrade pip

$ pip install --user numpy scipy matplotlib ipython jupyter pandas sympy nose

# user install executable directory is on your PATH

# Consider adding this at the end of your ~/.bashrc file

export PATH="$PATH:/home/your\_user/.local/bin"

## 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）

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

c.T # 转置

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

np.zeros((2,2))

np.ones((1,2))

np.full((2,2), 7)

np.eye(2)

np.random.random((2,2))

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

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

np.dot(v, w) # 矢量与矢量内积，矩阵与矢量乘，矩阵与矩阵乘

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")

## scipy

SciPy contains additional routines needed in scientific work: for example, routines for computing integrals numerically, solving differential equations, optimization, and sparse matrices.

Subpackage Description

cluster Clustering algorithms

constants Physical and mathematical constants

fftpack Fast Fourier Transform routines

integrate Integration and ordinary differential equation solvers

interpolate Interpolation and smoothing splines

io Input and Output

linalg Linear algebra

ndimage N-dimensional image processing

odr Orthogonal distance regression

optimize Optimization and root-finding routines

signal Signal processing

sparse Sparse matrices and associated routines

spatial Spatial data structures and algorithms

special Special functions

stats Statistical distributions and functions

weave C/C++ integration

<https://docs.scipy.org/doc/scipy/reference/tutorial/basic.html>

>>> import numpy as np

>>> import matplotlib as mpl

>>> import matplotlib.pyplot as plt

>>> from scipy import linalg, optimize

a = np.r\_[3,[0]\*5,-1:1:10j] row concatenation

np.c\_ column concatenation

meshgrid

produce N, N-d arrays which provide coordinate arrays for an N-dimensional volume

>>> np.mgrid[0:5:4j,0:5:4j]

array([[[ 0. , 0. , 0. , 0. ],

[ 1.6667, 1.6667, 1.6667, 1.6667],

[ 3.3333, 3.3333, 3.3333, 3.3333],

[ 5. , 5. , 5. , 5. ]],

[[ 0. , 1.6667, 3.3333, 5. ],

[ 0. , 1.6667, 3.3333, 5. ],

[ 0. , 1.6667, 3.3333, 5. ],

[ 0. , 1.6667, 3.3333, 5. ]]])

Polynomials

1-d polynomials = poly1d class (coefficients or polynomial roots to initialize a po lynomial)

manipulated in algebraic expressions, integrated, differentiated, and evaluated

>>> from numpy import poly1d

>>> p = poly1d([3,4,5]) 3x\*x + 4x + 5

>>> print p.integ(k=6) 积分

>>> print p.deriv() 微分

Vectorizing functions (vectorize)

# 定义标量函数

>>> def addsubtract(a,b):

... if a > b:

... return a - b

... else:

... return a + b

# 矢量化标量函数

>>> vec\_addsubtract = np.vectorize(addsubtract)

>>> vec\_addsubtract([0,3,6,9],[1,3,5,7])

Special functions (scipy.special)

airy, elliptic, bessel, gamma, beta, hypergeometric, parabolic cylinder, mathieu, spheroidal wave, struve, and kelvin

>>> from scipy import special

>>> def drumhead\_height(n, k, distance, angle, t):

... kth\_zero = special.jn\_zeros(n, k)[-1]

... return np.cos(t) \* np.cos(n\*angle) \* special.jn(n, distance\*kth\_zero)

>>> theta = np.r\_[0:2\*np.pi:50j]

>>> radius = np.r\_[0:1:50j]

>>> x = np.array([r \* np.cos(theta) for r in radius])

>>> y = np.array([r \* np.sin(theta) for r in radius])

>>> z = np.array([drumhead\_height(1, 1, r, theta, 0.5) for r in radius])

Integration (scipy.integrate)

>>> help(integrate)

>>> from scipy.integrate import quad 单变量积分

>>> from scipy.integrate import dblquad 双变量积分

>>> from scipy.integrate import tplquad 三变量积分

>>> from scipy.integrate import nquad 多变量积分

>>> N = 5

>>> def f(t, x):

... return np.exp(-x\*t) / t\*\*N

>>> nquad(f, [[1, np.inf],[0, np.inf]])

>>> from scipy.integrate import simps Integrating using Samples

Ordinary differential equations 常微分方程

>>> from scipy.integrate import odeint

Optimization (scipy.optimize)

1. Unconstrained and constrained minimization of multivariate scalar functions (minimize) using a variety of algorithms (e.g. BFGS, Nelder-Mead simplex, Newton Conjugate Gradient, COBYLA or SLSQP)

2. Global (brute-force) optimization routines (e.g. basinhopping, differential\_evolution)

3. Least-squares minimization (least\_squares) and curve fitting (curve\_fit) algorithms

4. Scalar univariate functions minimizers (minimize\_scalar) and root finders (newton)

5. Multivariate equation system solvers (root) using a variety of algorithms (e.g. hybrid Powell, Levenberg-Marquardt or large-scale methods such as Newton-Krylov).

>>> from scipy.optimize import minimize

# Unconstrained minimization of multivariate scalar functions 无约束

>>> def rosen(x):

... """The Rosenbrock function"""

... return sum(100.0\*(x[1:]-x[:-1]\*\*2.0)\*\*2.0 + (1-x[:-1])\*\*2.0)

>>> x0 = np.array([1.3, 0.7, 0.8, 1.9, 1.2])

>>> res = minimize(rosen, x0, method='nelder-mead',

... options={'xtol': 1e-8, 'disp': True})

# Constrained minimization of multivariate scalar functions (minimize) 有约束

the Sequential Least SQuares Programming optimization algorithm (SLSQP)

Least-squares minimization (least\_squares)

Univariate function minimizers (minimize\_scalar)

# Unconstrained minimization (method='brent')

>>> from scipy.optimize import minimize\_scalar

>>> f = lambda x: (x - 2) \* (x + 1)\*\*2

>>> res = minimize\_scalar(f, method='brent')

# Bounded minimization (method='bounded')

>>> from scipy.special import j1

>>> res = minimize\_scalar(j1, bounds=(4, 7), method='bounded')

Root finding

# Finding a root of a set of non-linear equations can be achieve using the root function

>>> import numpy as np

>>> from scipy.optimize import root

>>> def func(x):

... return x + 2 \* np.cos(x)

>>> sol = root(func, 0.3)

Interpolation (scipy.interpolate)

# 1-D interpolation (interp1d)

>>> from scipy.interpolate import interp1d

>>> x = np.linspace(0, 10, num=11, endpoint=True)

>>> y = np.cos(-x\*\*2/9.0)

>>> f = interp1d(x, y)

>>> f2 = interp1d(x, y, kind='cubic')

# Multivariate data interpolation (griddata)

>>> from scipy.interpolate import griddata

>>> points = np.random.rand(1000, 2)

>>> values = func(points[:,0], points[:,1])

>>> grid\_x, grid\_y = np.mgrid[0:1:100j, 0:1:200j]

>>> grid\_z0 = griddata(points, values, (grid\_x, grid\_y), method='nearest')

>>> grid\_z1 = griddata(points, values, (grid\_x, grid\_y), method='linear')

>>> grid\_z2 = griddata(points, values, (grid\_x, grid\_y), method='cubic')

# Spline interpolation in 1-d: Procedural (interpolate.splXXX)

Spline interpolation requires two essential steps: (1) a spline representation of the curve is computed, and (2) the spline is evaluated at the desired points. In order to find the spline representation, there are two different ways to represent a curve and obtain (smoothing) spline coefficients: directly and parametrically

>>> from scipy.interpolate import splrep， splev

>>> x = np.arange(0, 2\*np.pi+np.pi/4, 2\*np.pi/8)

>>> y = np.sin(x)

>>> tck = splrep(x, y, s=0)

>>> xnew = np.arange(0, 2\*np.pi, np.pi/50)

>>> ynew = splev(xnew, tck, der=0)

Two-dimensional spline representation: Procedural (bisplrep)

>>> from scipy.interpolate import bisplrep, bisplev

>>> x, y = np.mgrid[-1:1:20j, -1:1:20j]

>>> z = (x+y) \* np.exp(-6.0\*(x\*x+y\*y))

>>> xnew, ynew = np.mgrid[-1:1:70j, -1:1:70j]

>>> tck = interpolate.bisplrep(x, y, z, s=0)

>>> znew = interpolate.bisplev(xnew[:,0], ynew[0,:], tck)

Fourier analysis

discrete Fourier transform (DFT)

Fast Fourier Transform (FFT)

>>> from scipy.fftpack import fft, ifft

>>> x = np.array([1.0, 2.0, 1.0, -1.0, 1.5])

>>> y = fft(x)

>>> yinv = ifft(y)

Two and n-dimensional discrete Fourier transforms

>>> from scipy.fftpack import fft2, ifft2, fftn, ifftn

Discrete Cosine Transforms

>>> from scipy.fftpack import dct, idct

Discrete Sine Transforms

>>> from scipy.fftpack import dst, idst

Signal Processing (scipy.signal)

Filtering: Convolution/Correlation

Time-discrete filters: FIR(finite response filters) and IIR(infinite response filters)

Other filters: Median Filter, Order Filter, Wiener filter, Hilbert filter

Spectral Analysis, spectral density

Analog Filter Design

Linear Algebra (scipy.linalg)

scipy.linalg vs numpy.linalg

numpy.matrix vs 2D numpy.ndarray

Basic routines: Inverse, Solving linear system, Determinant, norms, Solving linear least-squares problems and pseudo-inverses, Decompositions(Eigenvalues and eigenvectors), SVD(Singular value decomposition)

Singular Value Decomposition (SVD) can be thought of as an extension of the eigenvalue problem to matrices that are not square, Every matrix has a singular value decomposition. Sometimes, the singular values are called the spectrum of A

Sparse Eigenvalue Problems with ARPACK

Spatial data structures and algorithms (scipy.spatial)

Delaunay triangulations

The Delaunay triangulation is a subdivision of a set of points into a non-overlapping set of triangles,

Convex hulls

Convex hull is the smallest convex object containing all points in a given point set.

Voronoi diagrams

A Voronoi diagram is a subdivision of the space into the nearest neighborhoods of a given set of points.

Statistics (scipy.stats)

continuous random variables and discrete random variables . Over 80 continuous random variables (RVs) and 10 discrete random variables

The main public methods for continuous RVs are:

rvs: Random Variates

pdf: Probability Density Function

cdf: Cumulative Distribution Function

sf: Survival Function (1-CDF)

ppf: Percent Point Function (Inverse of CDF)

isf: Inverse Survival Function (Inverse of SF)

stats: Return mean, variance, (Fisher’s) skew, or (Fisher’s) kurtosis

moment: non-central moments of the distribution

Multidimensional image processing (scipy.ndimage)

from scipy.misc import imread, imsave, imresize

import matplotlib.pyplot as plt

# Read an JPEG image into a numpy array

img = imread('assets/cat.jpg')

# Resize the image to be 300 by 300 pixels.

img\_tinted = imresize(img, (300, 300))

# Write the tinted image back to disk

imsave('assets/cat\_tinted.jpg', img\_tinted)

# Show the original image

plt.subplot(1, 2, 1)

plt.imshow(img)

Correlation and convolution

Fourier domain filters

Distance transforms

Interpolation functions

Morphology: Binary morphology, Grey-scale morphology

Segmentation and labeling

Object measurements

File IO (scipy.io)

MATLAB files

>>> import scipy.io as sio

sio.loadmat(file\_name[, mdict, appendmat]) Load MATLAB file.

sio.savemat(file\_name, mdict[, appendmat, ...]) Save a dictionary of names and arrays into a MATLAB-style .mat file.

sio.whosmat(file\_name[, appendmat]) List variables inside a MATLAB file.

Weave (scipy.weave)

The scipy.weave (below just weave) package provides tools for including C/C++ code within in Python code

## matplotlib

import matplotlib.pyplot as plt

plt.plot([1,2,3,4]) # array of y-axis, the default x [0, .. len(y)-1]

or plt.plot(t, t, 'r--', t, t\*\*2, 'bs', t, t\*\*3, 'g^')

plt.ylabel('some numbers')

plt.show()

Controlling line properties （matplotlib.lines.Line2D）

linewidth, dash style, antialiased, ...

same to matlab, pyplot has concept of current figure and axes

matplotlib.axes.Axes = gca()

matplotlib.figure.Figure = gcf()

clf(), cla(), close()

xlabel(), ylabel(), title() , text(), annotate(), xscale(‘log’), yscale9'log')

import numpy as np

import matplotlib.pyplot as plt

def f(t):

return np.exp(-t) \* np.cos(2\*np.pi\*t)

t1 = np.arange(0.0, 5.0, 0.1)

t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)

plt.subplot(211)

plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)

plt.plot(t2, np.cos(2\*np.pi\*t2), 'r--')

plt.show()

TeX markup in any matplotlib text

<http://matplotlib.org/users/mathtext.html>

## pandas

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

dates = pd.date\_range('20130101', periods=6)

df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))

df.head()

df.tail(3)

df.index

df.columns

df.values

df.T

df.sort\_index(axis=1, ascending=False) #sorting by an axis

df.sort\_values(by='B') #sorting by values

selection

df['A'] # select a single column

df.loc['20130102':'20130104',['A','B']] # both endpoints are included

df.at[dates[0],'A'] # 获取单元素

df[0:3] # slices the rows.

df.iloc[3:5,0:2] # By integer slices

df.iat[1,1] # 获取单元素

Boolean Indexing

df[df.A > 0]

df[df > 0]

df2[df2['E'].isin(['two','four'])]

df2[df2 > 0] = -df2 #只对df2>0的部分进行操作，别的元素不动

Missing Data

pandas primarily uses the value np.nan to represent missing data. It is by default not included in computations

df1.dropna(how='any') # To drop any rows that have missing data.

df1.fillna(value=5) # Filling missing data

pd.isnull(df1) # To get the boolean mask where values are nan

df.describe() # 描述性统计

df.apply(lambda x: x.max() - x.min()) # 列处理

s = pd.Series(np.random.randint(0, 7, size=10))

s.value\_counts() # 直方图

pd.concat([df[:3], df[3:7], df[7:]])

pd.merge(left, right, on='key') # SQL style merges

df.append(s, ignore\_index=True) # Append rows to a dataframe

Grouping：

Splitting the data into groups based on some criteria

Applying a function to each group independently

Combining the results into a data structure

df.groupby('A').sum()

Time Series

Converting between period and timestamp

rng = pd.date\_range('3/6/2012 00:00', periods=5, freq='D')

rng = pd.date\_range('1/1/2012', periods=5, freq='M')

Categoricals

df = pd.DataFrame({"id":[1,2,3,4,5,6], "raw\_grade":['a', 'b', 'b', 'a', 'a', 'e']})

df["grade"] = df["raw\_grade"].astype("category")

df.to\_csv('foo.csv') # Writing to a csv file

pd.read\_csv('foo.csv') # Reading from a csv file

df.to\_hdf('foo.h5','df') # Writing to a HDF5 Store

pd.read\_hdf('foo.h5','df') # Reading from a HDF5 Store

df.to\_excel('foo.xlsx', sheet\_name='Sheet1') # Writing to an excel file

pd.read\_excel('foo.xlsx', 'Sheet1', index\_col=None, na\_values=['NA']) # Reading from an excel file

## SymPy

Symbolic computation deals with the computation of mathematical objects symbolically

Symbolic computation systems (which by the way, are also often called computer algebra systems, or just CASs) such as SymPy are capable of computing symbolic expressions with variables.

SymPy can simplify expressions, compute derivatives, integrals, and limits, solve equations, work with matrices, it includes modules for plotting, printing (like 2D pretty printed output of math formulas, or LATEXLATEX), code generation, physics, statistics, combinatorics, number theory, geometry, logic, and more

Whenever you combine a SymPy object and a SymPy object, or a SymPy object and a Python object, you get a SymPy object, but whenever you combine two Python objects, SymPy never comes into play, and so you get a Python object.

import math

math.sqrt(8)

import sympy

sympy.sqrt(8)

易犯错误点：

x = symbols('x')

expr = x + 1

expr.subs(x, 2) 符号表达式求值

expr.subs(x, x\*y) 符号替换

expr = x\*\*3 + t\*x\*y - z

expr.subs([(x, 2), (y, 4), (z, 0)])

expr = x\*\*4 - 4\*x\*\*3 + 4\*x\*\*2 - 2\*x + 3

replacements = [(x\*\*i, y\*\*i) for i in range(5) if i%2 == 0]

expr.subs(replacements) 结果y\*\*4 - 4\*x\*\*3 + 4\*y\*\*2 - 2\*x + 3

注意：SymPy expressions are immutable, no function will change them in-place. All functions will return new expressions.

条件测试

a = (x+1)\*\*2

b = x\*\*2 + 2\*x + 1

if simplify(a - b) == 0:

if a.equals(b):

字符串转符号表达式converting strings to sympy expressions

str\_expr = "x\*\*2 + 3\*x - 1/2"

expr = sympify(str\_expr)

convert a SymPy expression to an expression

expr = sin(x)

f = lambdify(x, expr, "numpy")

f(numpy.arange(10))

Simplification

uses heuristics to determine the simplest result

>>> simplify(sin(x)\*\*2 + cos(x)\*\*2)

>>> simplify((x\*\*3 + x\*\*2 - x - 1)/(x\*\*2 + 2\*x + 1))

Polynomial/Rational Function Simplification

x, y =symbols('x y z')

>>> expand((x + 1)\*\*2)

>>> factor(x\*\*2\*z + 4\*x\*y\*z + 4\*y\*\*2\*z)

expr.coeff(x, n) gives the coefficient of x\*\*n in expr:

>>> expr = x\*y + x - 3 + 2\*x\*\*2 - z\*x\*\*2 + x\*\*3

>>> collected\_expr = collect(expr, x)

>>> collected\_expr.coeff(x, 2)

>>> trigsimp(sin(x)\*tan(x)/sec(x)) #simplify trigonometric

>>> expand\_trig(sin(x + y)) #expand trigonometric functions

By default, SymPy Symbols are assumed to be complex

Symbols can be given different assumptions by passing the assumption to symbols()

>>> x, y = symbols('x y', positive=True)

>>> a, b = symbols('a b', real=True)

Special Functions

>>> factorial(n)

>>> binomial(n, k) n choose k

>>> gamma(z)

Calculus

>>> diff(x\*\*4, x, 3) #the third derivative 微分

>>> diff(exp(x\*y\*z), x, y, 2, z, 4) #多阶偏微分

integrate(exp(x)\*sin(x) + exp(x)\*cos(x), x) 积分

integrate(sin(x\*\*2), (x, -oo, oo)) 定积分

>>> integrate(exp(-x\*\*2 - y\*\*2), (x, -oo, oo), (y, -oo, oo)) #多变量积分

limit(sin(x)/x, x, 0) 极限

>>> limit(1/x, x, 0, '+') 单边极限

>>> expr = exp(sin(x))

>>> expr.series(x, 0, 4) 泰勒级数展开

Solvers

Eq(x+1, 4) x+1 = 4 方程等式

>>> solveset(Eq(x\*\*2, 1), x) #solving algebraic equations

注：solve(x\*\*2 - 2, x)求根不被推荐

>>> f = symbols('f', cls=Function)

>>> diffeq = Eq(f(x).diff(x, x) - 2\*f(x).diff(x) + f(x), sin(x))

>>> dsolve(diffeq, f(x)) 解微分方程

Matrix([[1, 2], [2, 2]]).eigenvals()

besselj(nu, z).rewrite(jn)

latex(Integral(cos(x)\*\*2, (x, 0, pi)))

# CS231n Convolutional Neural Networks for Visual Recognition

## RNN

网络权值的解释？W = 模板

Interpretation of linear classifiers as template matching. Another interpretation for the weights WW is that each row of WW corresponds to a template (or sometimes also called a prototype) for one of the classes. The score of each class for an image is then obtained by comparing each template with the image using an inner product (or dot product) one by one to find the one that “fits” best. With this terminology, the linear classifier is doing template matching, where the templates are learned. Another way to think of it is that we are still effectively doing Nearest Neighbor, but instead of having thousands of training images we are only using a single image per class (although we will learn it, and it does not necessarily have to be one of the images in the training set), and we use the (negative) inner product as the distance instead of the L1 or L2 distance.

for example: the ship template (W) contains a lot of blue pixels as expected. This template will therefore give a high score once it is matched against images of ships on the ocean with an inner product.

Additionally, note that the horse template seems to contain a two-headed horse, which is due to both left and right facing horses in the dataset. The linear classifier merges these two modes of horses in the data into a single template. Similarly, the car classifier seems to have merged several modes into a single template which has to identify cars from all sides, and of all colors. In particular, this template ended up being red, which hints that there are more red cars in the CIFAR-10 dataset than of any other color. The linear classifier is too weak to properly account for different-colored cars, but as we will see later neural networks will allow us to perform this task. Looking ahead a bit, a neural network will be able to develop intermediate neurons in its hidden layers that could detect specific car types (e.g. green car facing left, blue car facing front, etc.), and neurons on the next layer could combine these into a more accurate car score through a weighted sum of the individual car detectors.

First-layer Visualizations



Examples of visualized weights for the first layer of a neural network. Left: Noisy features indicate could be a symptom: Unconverged network, improperly set learning rate, very low weight regularization penalty. Right: Nice, smooth, clean and diverse features are a good indication that the training is proceeding well.

score function: mapping the raw image pixels to class scores (e.g. a linear function)

loss(cost) function: measured the quality of a particular set of parameters based on how well the induced scores agreed with the ground truth labels in the training data (e.g. Softmax/SVM).

Optimization: the process of finding the set of parameters WW that minimize the loss function.

linear mapping -> Neural Networks -> Convolutional Neural Networks

SVM cost function is an example of a convex function --- convex optimization

Neural Networks cost functions will become non-convex

--- Random search (bad idea)

W = np.random.randn(10, 3073) \* 0.0001 # generate random parameters

--- Random Local Search

Core idea: iterative refinement, refining a specific set of weights W to be slightly better is significantly less difficult

Wtry = W + np.random.randn(10, 3073) \* step\_size

--- Following the Gradient

weights += - step\_size \* weights\_grad # perform parameter update

The gradient tells us the direction in which the function has the steepest rate of increase, but it does not tell us how far along this direction we should step

choosing the step size (also called the learning rate) will become one of the most important (and most headache-inducing) hyperparameter settings in training a neural network

Computing the gradient: numerical gradient and analytic gradient (e.g. chain rule == backpropagation)

Backpropagation allow us to efficiently optimize relatively arbitrary loss functions that express all kinds of Neural Networks, including Convolutional Neural Networks.

Batch梯度下降法(BGD)：it seems wasteful to compute the full loss function over the entire training set in order to perform only a single parameter update

Mini-batch 梯度下降法(MGD)：compute the gradient over batches of the training data

假定120万张图像，由1000 labels组成，那么120万张的平均data loss与1000张等值，the gradient from a mini-batch is a good approximation of the gradient of the full objective (cost)

The size of the mini-batch is a hyperparameter but it is not very common to cross-validate it. It is usually based on memory constraints (if any), or set to some value, e.g. 32, 64 or 128. We use powers of 2 in practice because many vectorized operation implementations work faster when their inputs are sized in powers of 2.

Stochastic Gradient Descent (SGD) :on-line gradient descent

Tips:

if loss barely changing means Learning rate is probably too low

mini-batch size: 256

step size: 10\*\*(-5) (ideally, h->0)

2% improvement: train multiple independent models, at test time average their results

Mini-batch SGD

Loop:

1. Sample a batch of data
2. Forward prop it through the graph, get loss
3. Backprop to calculate the gradients
4. Update the parameters using the gradient

Regularization (dropout)

Randomly set some neurons to zero in the forward pass



解释1：



解释2:

Dropout is training a large ensemble of models (that share parameters). Each binary mask is one model gets trained on only one datapoint

Ideally, want to integrate out all the noise.

## Convolutional Neural Networks (CNNs / ConvNets)

ConvNet architectures:

Images -> Convolutional Layer -> Pooling Layer -> Fully-Connected Layer -> output labels

ConvNet architectures make the explicit assumption that the inputs are images

Regular Neural Nets don’t scale well to full images, for example, image 200\*200\*3 -> 120,000 weights

full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

3D volumes of neurons: unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth.

Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network, we will have an entire set of filters in each CONV layer (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.



For example, if the first Convolutional Layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edged, or blobs of color. We will refer to a set of neurons that are all looking at the same region of the input as a depth column (some people also prefer the term fibre).

Real-world example

the input volume size (W1 \* H1 \* D1): 227\*227\*3

the receptive field size of the Conv Layer neurons (F): 11

the number of filters (K): 96

the stride with which they are applied (S): 4

the amount of zero padding used (P) on the border: 0

the spatial size of the output volume (W2 \* H2 \* D2): 55\*55\*96

W2 = (W1-F+2P)/S+1 = 55

H2 = (H1−F+2P)/S+1 = 55

D2 = K = 96

With parameter sharing, each filter has F\*F\*D1 (11\*11\*3 = 363) weights, total weights (F\*F\*D1)\*K (363x96) and K biases

理论上：参数有55x55x96 \* 11\*11\*3 多个

假设：Notice that the parameter sharing assumption is relatively reasonable: If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally-invariant structure of images.

假设不一定对：Note that sometimes the parameter sharing assumption may not make sense. This is especially the case when the input images to a ConvNet have some specific centered structure, where we should expect, for example, that completely different features should be learned on one side of the image than another. One practical example is when the input are faces that have been centered in the image. You might expect that different eye-specific or hair-specific features could (and should) be learned in different spatial locations. In that case it is common to relax the parameter sharing scheme, and instead simply call the layer a Locally-Connected Layer.

若假设对，则图像空间共用同一个filter（理由如上）,则参数：96 \* 11\*11\*3

Backpropagation: The backward pass for a convolution operation (for both the data and the weights) is also a convolution (but with spatially-flipped filters).

all 55\*55 neurons in each depth slice will now be using the same parameters. In practice during backpropagation, every neuron in the volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.

Pooling Layer



reduce the amount of parameters and computation in the network, and hence to also control overfitting

how? filters of size 2x2 applied with a stride of 2 downsamples every depth slice

input volumn size: W1\*H1\*D1

filter (F): 2\*2

stride (S): 2

output volumn size: W2\*H2\*D2

W2 = (W1-F)/S + 1

H2 = (H1-F)/S + 1

D2 = D1

general pooling: max, average, L2-norm pooling

Backpropagation: Recall from the backpropagation chapter that the backward pass for a max(x, y) operation has a simple interpretation as only routing the gradient to the input that had the highest value in the forward pass. Hence, during the forward pass of a pooling layer it is common to keep track of the index of the max activation (sometimes also called the switches) so that gradient routing is efficient during backpropagation.

Fully-connected layer

Converting FC layers to CONV layers

input volume of size: 7\*7\*512

F = 7

P = 0

S = 1

K = 4096

output: 1\*1\*4096

Layer Patterns

INPUT -> [[CONV -> RELU]\*N -> POOL?]\*M -> [FC -> RELU]\*K -> FC

usually

0 <= N <= 3

0 <= M

0 <= K < 3

INPUT -> FC, implements a linear classifier. Here N = M = K = 0.

INPUT -> CONV -> RELU -> FC

INPUT -> [CONV -> RELU -> POOL]\*2 -> FC -> RELU -> FC. Here we see that there is a single CONV layer between every POOL layer.

INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]\*3 -> [FC -> RELU]\*2 -> FC Here we see two CONV layers stacked before every POOL layer. This is generally a good idea for larger and deeper networks, because multiple stacked CONV layers can develop more complex features of the input volume before the destructive pooling operation

多个小filter级别优于大filter

Suppose that you stack three 3x3 CONV layers on top of each other (with non-linearities in between, of course). In this arrangement, each neuron on the first CONV layer has a 3x3 view of the input volume. A neuron on the second CONV layer has a 3x3 view of the first CONV layer, and hence by extension a 5x5 view of the input volume. Similarly, a neuron on the third CONV layer has a 3x3 view of the 2nd CONV layer, and hence a 7x7 view of the input volume. Suppose that instead of these three layers of 3x3 CONV, we only wanted to use a single CONV layer with 7x7 receptive fields.

3个3\*3filter等效于7\*7filter，但有如下优点

First, the neurons would be computing a linear function over the input, while the three stacks of CONV layers contain non-linearities that make their features more expressive.

Second, if we suppose that all the volumes have CC channels, then it can be seen that the single 7x7 CONV layer would contain C×(7×7×C)=49C2C×(7×7×C)=49C2 parameters, while the three 3x3 CONV layers would only contain 3×(C×(3×3×C))=27C23×(C×(3×3×C))=27C2 parameters.

Intuitively, stacking CONV layers with tiny filters as opposed to having one CONV layer with big filters allows us to express more powerful features of the input, and with fewer parameters

INPUT: [224x224x3] memory: 224\*224\*3=150K weights: 0

CONV3-64: [224x224x64] memory: 224\*224\*64=3.2M weights: (3\*3\*3)\*64 = 1,728

CONV3-64: [224x224x64] memory: 224\*224\*64=3.2M weights: (3\*3\*64)\*64 = 36,864

POOL2: [112x112x64] memory: 112\*112\*64=800K weights: 0

CONV3-128: [112x112x128] memory: 112\*112\*128=1.6M weights: (3\*3\*64)\*128 = 73,728

CONV3-128: [112x112x128] memory: 112\*112\*128=1.6M weights: (3\*3\*128)\*128 = 147,456

POOL2: [56x56x128] memory: 56\*56\*128=400K weights: 0

CONV3-256: [56x56x256] memory: 56\*56\*256=800K weights: (3\*3\*128)\*256 = 294,912

CONV3-256: [56x56x256] memory: 56\*56\*256=800K weights: (3\*3\*256)\*256 = 589,824

CONV3-256: [56x56x256] memory: 56\*56\*256=800K weights: (3\*3\*256)\*256 = 589,824

POOL2: [28x28x256] memory: 28\*28\*256=200K weights: 0

CONV3-512: [28x28x512] memory: 28\*28\*512=400K weights: (3\*3\*256)\*512 = 1,179,648

CONV3-512: [28x28x512] memory: 28\*28\*512=400K weights: (3\*3\*512)\*512 = 2,359,296

CONV3-512: [28x28x512] memory: 28\*28\*512=400K weights: (3\*3\*512)\*512 = 2,359,296

POOL2: [14x14x512] memory: 14\*14\*512=100K weights: 0

CONV3-512: [14x14x512] memory: 14\*14\*512=100K weights: (3\*3\*512)\*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14\*14\*512=100K weights: (3\*3\*512)\*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14\*14\*512=100K weights: (3\*3\*512)\*512 = 2,359,296

POOL2: [7x7x512] memory: 7\*7\*512=25K weights: 0

FC: [1x1x4096] memory: 4096 weights: 7\*7\*512\*4096 = 102,760,448

FC: [1x1x4096] memory: 4096 weights: 4096\*4096 = 16,777,216

FC: [1x1x1000] memory: 1000 weights: 4096\*1000 = 4,096,000

TOTAL memory: 24M \* 4 bytes ~= 93MB / image (only forward! ~\*2 for bwd)

TOTAL params: 138M parameters