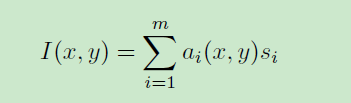
### 实验六 TopoICA模型实现

拓扑ICA（拓扑结构的独立成分分析）是一个生成模型，他将拓扑映射和ICA结合在一起。在所有的拓扑映射中，表征空间中的距离（在拓扑‘网格’）是相似于表征的成分的距离。在拓扑ICA中，表征成分的距离通过高阶相关性的互信息定于得到的，这给出了ICA上下文中的自然距离测量。在自然图像数据上的应用，拓扑ICA给出了类Gabor线性特征中的线性分解。与普通的ICA相比，这个高阶依赖性显示线性ICA不能移除定义一个拓扑顺序使得近邻的细胞倾向于在同一时间激活。这同样暗示这邻居有着与复杂细胞一样的特性。这个方法因此可以展现复杂细胞特性和拓扑组织的同时发生。这两个特性通过同时邻居的激活值定义的拓扑的原则而得到的。

拓扑ICA对经典ICA模型进行了一个修正，使得成分之间的依赖性被显式的表达。还要特别提出，独立成分的残余依赖性结构，也就是ICA无法消除的依赖性，能够用于定义成分之间的一种拓扑序。这种拓扑序很容易用可视化方式表示，因其与脑建模之间的联系，拓扑序随图像特征提取具有重要意义。

**实验目的：**加深对TopoICA模型的理解，能够使用TopoICA模型解决简单问题

**实验原理：**假定一个静态灰度化图像I（x,y），他是由许多特征或基函数ai（x,y）的线性组合得到的：



si是随机系数，在每个图像I（x,y）中都是不同的。这里关键的假设是si是非高斯的，而且是互相独立的。这样的分解就叫做独立成分分析（ICA），或者换个观点来说，就是稀疏编码。

在给定足够数量的图像或者说图像碎片I（x,y）的观测值的情况下，估算上述等式中的模型需要对于所有的i和（x,y）的情况下决定si，ai（x，y）的值。我们假定基础的情况下，ai(x,y)来自于一个可逆的线性系统。然后我们就能对这个系统求逆：





wi表示逆过滤器，可以认为是模型简单的细胞的感受野，而且si是当输入是I（x，y）的激活值。

在经典的ICA中，独立成分si并没有特别的顺序，或者其他的关系。独立成分内部固有顺序或关系的缺失，与我们假定他们是完全的统计独立是有关系的。当在自然图像统计上使用ICA进行建模时，会清晰的发现在很多地方是违反独立假设的前提的，可以很容易发现所估计的独立成分之间是有着清晰的依赖（非独立）关系的。在这部分中，我们通过使用一个生成模型来定义拓扑ICA，这个模型是普通ICA模型的分层的版本。这个想法是放宽I（x,y）等式中成分的独立性假设，这样就能使得拓扑中相互靠近的成分不再假定具有独立性。例如，拓扑是由点阵或网格定义的，成分的依赖性就是网格上成分的距离函数。相反的，在拓扑中不相互靠近的成分还是独立的，至少近似独立；因此大多数成分对还是独立的。

近邻成分之间应该选择高阶的依赖性，即能量相关：

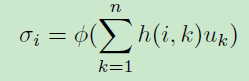


这里假定si和sj是拓扑中的近邻，而且协方差是正的。直观上，这样一个相关性意味着成分倾向于在同一时刻是活动的，即非零，但是si和sj的真实值却不是那么容易预测的。即有可能si和sj是不相关的，但是他们的能量却不是。

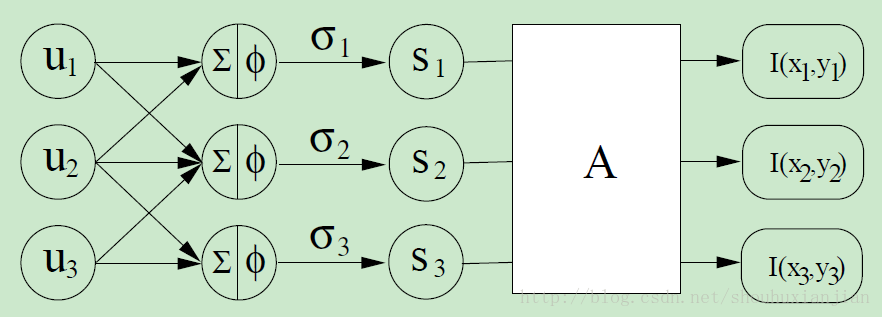
定义一个生成模型，这个模型能够表示拓扑网格中邻近成分的能量相关性。在这个模型中，观测到的图像碎片作为成分si的线性转换而生成，就像I（x,y）等式中的基础ICA模型一样。关键的点是如何定义一个s的联合密度去表示这个拓扑。 我们通过以下的方式来定义s的联合密度。si的方差sigma（i）^2不是常量，他们被假定成是随机变量，有具体的模型生成。在生成方差后，变量si是由具体的条件分布来相互之间独立生成的，换句话来说，si是在给定方差下独立的。si之间的依赖性隐含在他们的方差的依赖性中。根据拓扑的原则，近邻成分的相对应的方差应该是（正）相关的，而且相互不靠近的成分的方差是独立的，至少是近似的。

为了具体说明模型的方差sigma（i）^2，我们首先需要定义拓扑。这可以通过函数h（i,j）来完成，这个函数能够表示第i个和第j个成分的相关程度。这个函数能够以自组织映射的同样的方法来定义。所以h（i，j）是一个超参数矩阵。在本文中，我们认为他是未知的，但是是固定的。

     为了使用拓扑相关性h（i,j），需要许多不同模型的方差sigma（i）^2，我们通过一个非线性ICA模型来定义他们：



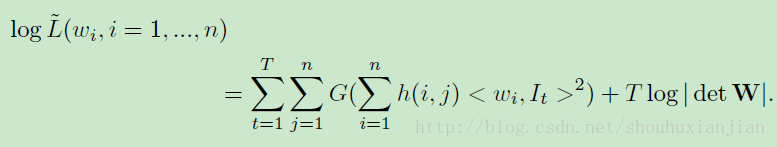
这里uk是高阶独立成分，可以用作去生成方差，而且Φ是标量非线性的。uk的分布和Φ的实际形式是模型的附加超参数。限制uk为非负的。这个函数Φ可以被认为是在非负实数集中的一个单调转换。这确保了sigma（i）是非负的。生成的拓扑ICA模型如下所示：

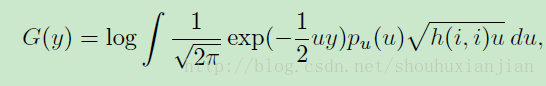


为了估计这个模型，我们采用最大释然估计。为了简化估计，我们通过一个易处理的似然的逼近。为了获得这个逼近，我们首先做出下面的假设。首先，我们固定成分si的条件密度是高斯的。第二，我们定义非线性Φ是

http://img.blog.csdn.net/20140906143952888?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvc2hvdWh1eGlhbmppYW4=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

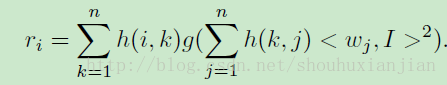
第三，我们简单的假定si的条件密度对于所有的i来说都是相同的。模型的似然度函数定义如下：





简单的得到一个梯度算法，第i个向量wi的更新规则如下：





**实验内容：**根据TopoICA模型的相关知识，使用Python语言实现一个简单的TopoICA模型。简单程序示例如下所示：

model.py

import chainer

import chainer.functions as F

import chainer.links as L

import cupy as cp

import numpy as np

class ReconstractionTICA(chainer.Chain):

def \_\_init\_\_(self):

super(ReconstractionTICA, self).\_\_init\_\_()

with self.init\_scope():

self.f = L.Linear(1 \* 32 \* 32, 1 \* 22 \* 22)

self.p = F.average\_pooling\_2d

self.g = L.Linear(1 \* 22 \* 22, 1 \* 32 \* 32)

self.mask = cp.zeros((1, 22, 22, 1, 32, 32), dtype=float)

for dy in range(22):

for dx in range(22):

self.mask[0, dy, dx, 0, dy:dy+11, dx:dx+11] = 1.0

self.mask = self.mask.reshape((1 \* 22 \* 22, 1 \* 32 \* 32))

def \_\_call\_\_(self, x):

self.f.W.data = self.f.W.data \* self.mask

self.g.W.data = self.g.W.data \* self.mask.T

h = self.f(x)

h1 = F.square(h)

h1 = F.reshape(h1, (-1, 1, 22, 22))

h1 = self.p(h1, 5, stride=1, pad=0)

h1 = F.sqrt(h1 + 0.0001)

h2 = self.g(h)

return (h1, h2)

train.py

import chainer

import chainer.functions as F

import chainer.links as L

import chainer.cuda

import datetime

import numpy as np

import os

import pickle

import shutil

import tarfile

import urllib.request

from model import ReconstractionTICA

from visualize import visualize, visualize\_kernel

# Define constants

λ = 1.0 # Reconstruction coefficient

N = 100 # Minibatch size

SNAPSHOT\_INTERVAL = 10

def main():

# (Make directories)

os.mkdir('dataset/') if not os.path.isdir('dataset') else None

os.mkdir('train/') if not os.path.isdir('train') else None

# (Download dataset)

if not os.path.exists('dataset/cifar-10-gray.npy'):

url =

'https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz'

response = urllib.request.urlopen(url)

with open('dataset/cifar-10-python.tar.gz', 'wb') as stream:

stream.write(response.read())

with tarfile.open('dataset/cifar-10-python.tar.gz', 'r') as

stream:

stream.extractall('dataset/')

train = []

for path in ['data\_batch\_1', 'data\_batch\_2', 'data\_batch\_3',

'data\_batch\_4', 'data\_batch\_5', 'test\_batch']:

path = 'dataset/cifar-10-batches-py/' + path

with open(path, 'rb') as stream:

\_ = pickle.load(stream, encoding='bytes')

\_ = np.frombuffer(\_[b'data'],

dtype=np.uint8).reshape((-1, 3, 32, 32))

for rgb in \_:

r = rgb[0, :, :].astype('f') / 255.

g = rgb[1, :, :].astype('f') / 255.

b = rgb[2, :, :].astype('f') / 255.

gray = 0.2989 \* r + 0.5870 \* g + 0.1140 \* b

train.append(gray)

train = np.array(train, dtype='f')

np.save('dataset/cifar-10-gray', train)

os.remove('dataset/cifar-10-python.tar.gz') if

os.path.exists('dataset/cifar-10-python.tar.gz') else None

shutil.rmtree('dataset/cifar-10-batches-py',

ignore\_errors=True)

# (Download CIFAR100 dataset)

if not os.path.exists('dataset/cifar-100-gray.npy'):

url =

'https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz'

response = urllib.request.urlopen(url)

with open('dataset/cifar-100-python.tar.gz', 'wb') as stream:

stream.write(response.read())

with tarfile.open('dataset/cifar-100-python.tar.gz', 'r') as

stream:

stream.extractall('dataset/')

train = []

for path in ['train', 'test']:

path = 'dataset/cifar-100-python/' + path

with open(path, 'rb') as stream:

\_ = pickle.load(stream, encoding='bytes')

\_ = np.frombuffer(\_[b'data'],

dtype=np.uint8).reshape((-1, 3, 32, 32))

for rgb in \_:

r = rgb[0, :, :].astype('f') / 255.

g = rgb[1, :, :].astype('f') / 255.

b = rgb[2, :, :].astype('f') / 255.

gray = 0.2989 \* r + 0.5870 \* g + 0.1140 \* b

train.append(gray)

train = np.array(train, dtype='f')

np.save('dataset/cifar-100-gray', train)

os.remove('dataset/cifar-100-python.tar.gz') if

os.path.exists('dataset/cifar-100-python.tar.gz') else None

shutil.rmtree('dataset/cifar-100-python', ignore\_errors=True)

# Create samples.

train = np.load('dataset/cifar-10-gray.npy').reshape((-1, 1, 32,

32)) # cifar-10 or cifar-100

train = np.random.permutation(train)

validation = train[0:100]

# Create the model

nn = ReconstractionTICA()

# (Use GPU)

chainer.cuda.get\_device(0).use()

nn.to\_gpu()

# Setup optimizers

optimizer = chainer.optimizers.Adam()

optimizer.setup(nn)

# (Change directory)

os.chdir('train/')

time = datetime.datetime.today().strftime("%Y-%m-%d %H.%M.%S")

os.mkdir(time)

os.chdir(time)

# (Validate input images)

visualize(validation, 'validation.png', (32, 32))

# Training

for epoch in range(200):

# (Validate generated images)

if (epoch % SNAPSHOT\_INTERVAL == 0):

os.mkdir('%d' % epoch)

os.chdir('%d' % epoch)

y, z = nn(chainer.cuda.to\_gpu(validation))

visualize(chainer.cuda.to\_cpu(z.data), 'z.png', (32, 32))

visualize(chainer.cuda.to\_cpu(y.data), 'y.png', (18, 18))

visualize\_kernel(chainer.cuda.to\_cpu(nn.f.W.data),

'W.png')

os.chdir('..')

# (Random shuffle samples)

train = np.random.permutation(train)

total\_loss\_reg = 0.0

total\_loss\_rec = 0.0

for n in range(0, len(train), N):

x = chainer.cuda.to\_gpu(train[n:n + N].reshape((N, 1 \* 32

\* 32)))

y, z = nn(x)

# loss\_reg = F.sum(y)

# loss\_rec = F.sum((x - z) \*\* 2)

loss\_reg = F.mean(y)

loss\_rec = F.sum((x - z) \*\* 2) / np.prod(x.shape)

loss = loss\_reg + λ \* loss\_rec

nn.cleargrads()

loss.backward()

optimizer.update()

total\_loss\_reg += loss\_reg.data

total\_loss\_rec += loss\_rec.data

# (View loss)

total\_loss\_reg /= len(train) / N

total\_loss\_rec /= len(train) / N

print(epoch, total\_loss\_reg, total\_loss\_rec)

if \_\_name\_\_ == '\_\_main\_\_':

main()

visualize.py

import matplotlib.pyplot as plt

def visualize(X, fname, shape):

plt.figure(num=None, figsize=(10, 10), dpi=80, facecolor='w',

edgecolor='k')

for i in range(0, 100):

plt.subplot(10, 10, i + 1)

plt.tick\_params(labelleft='off', top='off', bottom='off')

plt.tick\_params(labelbottom='off', left='off', right='off')

plt.imshow(X[i].reshape(shape), cmap='gray')

plt.savefig(fname)

plt.close()

def visualize\_kernel(W, fname):

plt.figure(num=None, figsize=(22, 22), dpi=60, facecolor='w',

edgecolor='k')

for y in range(22):

for x in range(22):

plt.subplot(22, 22, x + y \* 22 + 1)

plt.tick\_params(labelleft='off', top='off', bottom='off')

plt.tick\_params(labelbottom='off', left='off',

right='off')

plt.imshow(W.reshape((1, 22, 22, 1, 32, 32))[0, y, x, 0,

y:y+11, x:x+11], cmap='gray')

plt.savefig(fname)

plt.close()

**实验要求：**

1. 下载MNIST数据集。
2. 构建TopoICA模型。
3. 使用MNIST数据集中的训练集训练网络，使用测试集测试训练好的网络。