认知科学与类脑计算

实验报告六

日期：2019/5/31

班级：16 人工智能

姓名：贾乘兴

学号：201600301304

一.需求分析：

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。本实验目的是加深对 TopoICA 模型的理解，能够使用 TopoICA 模型解决简单问题

二.概要设计：

利用下载好的 MNIST 数据集，构建拓扑 ica 模型。使用 MNIST 数据集中的训练集训练参数 w，并对分解的独立成分和重建效果以及权重进行可视化

三.详细设计：

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 a_i（x,y）的线性组合得到的：

$$I(x,y)=\sum_{i=1}^{m}a_i(x,y)s_i$$

s_i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 s_i，a_i（x，y）的值。我们假定基础的情况下，a_i(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：

$$s_i=<w_i,I>$$

$$<w_i,I>=\sum_{x,y}w_i(x,y)I(x,y)$$

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

在经典的 ICA 中，独立成分 s_i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：

$$\mathrm{cov}\,(s_i^2, s_j^2) = E\{s_i^2 s_j^2\} - E\{s_i^2\}E\{s_j^2\} \neq 0$$

这里假定 s_i 和 s_j 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 s_i 和 s_j 的真实值却不是那么容易预测的。即有可能 s_i 和 s_j 是不相关的，但是他们的能量却不是。

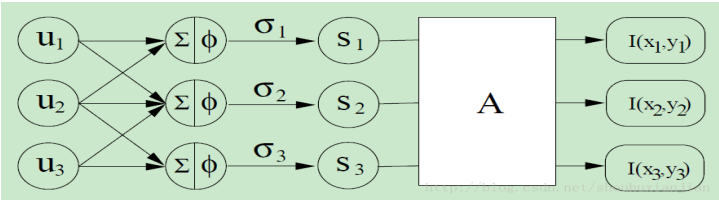
定义一个生成模型，这个模型能够表示拓扑网格中邻近成分的能量相关性。在这个模型中，观测到的图像碎片作为成分 s_i 的线性转换而生成，就像 I（x,y）等式中的基础 ICA 模型一样。关键的点是如何定义一个 s 的联合密度去表示这个拓扑。 我们通过以下的方式来定义 s 的联合密度。s_i 的方差 sigma（i）^2 不是常量，他们被假定成是随机变量，有具体的模型生成。在生成方差后，变量 s_i 是由具体的条件分布来相互之间独立生成的，换句话来说，s_i 是在给定方差下独立的。s_i 之间的依赖性隐含在他们的方差的依赖性中。根据拓扑的原则，近邻成分的相对应的方差应该是（正）相关的，而且相互不靠近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

$$\sigma_i = \phi(\sum_{k=1}^{n} h(i,k)u_k)$$

这里 u_k 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。u_k 的分布和Φ的实际形式是模型的附加超参数。限制 u_k 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 s_i 的条件密度是高斯的。第二，我们定义非线性Φ是

$$\phi(y) = y^{-1/2}$$

第三，我们简单的假定 s_i 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：

$$\log \tilde{L}(w_i, i = 1, ..., n) = \sum_{t=1}^{T} \sum_{j=1}^{n} G(\sum_{i=1}^{n} h(i,j) < w_i, I_t >^2) + T \log |\det \mathbf{W}|.$$

$$G(y) = \log \int \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2} uy) p_u(u) \sqrt{h(i,i)u}\, du,$$

简单的得到一个梯度算法，第 i 个向量 w_i 的更新规则如下：

$$\Delta w_i(x,y) \propto E\{I(x,y) < w_i, I > r_i)\}$$

$$r_i = \sum_{k=1}^{n} h(i,k) g(\sum_{j=1}^{n} h(k,j) < w_j, I >^2).$$

四.调试分析：

使用 chainer 框架实现该模型，利用上述更新法则进行独立成分和权重的的训练，不断返回并得到新的参数，将其可视化

五.测试结果：

训练过程中，正则化误差（独立成分的均值）和重建误差（利用训练的权重重建图像的误差）如下变化（格式：epoch 正则化误差 重建误差）

0 0.035932262831677 99 0.038585356880600254

1 0.023885682507728538 0.016111786564191183

2 0.021652699634432792 0.010833577974699438

3 0.020065714344382285 0.00830004913189138

4 0.018900230874617896 0.0069067700199472405

5 0.017886487049981952 0.005693603341157238

6 0.017111703899378577 0.005012595478134851

7 0.01645896610493461 0.004455533623695373

8 0.01589131131923447 0.003941977928237369

9 0.015411768766740958 0.0035615346929989754

10 0.014966872424508134 0.0031093906929406025

11 0.014651160937113066 0.0030683491949457676

12 0.014277999273811777 0.0025242847312862676

13 0.014007060471922l6 0.0024014468359140058

14 0.013764460808597506 0.0022671102875998863

15 0.013570098620839417 0.0021910528514611847

16 0.013374947727036972 0.00211165567394346

17 0.013179478580132126 0.0018209118533801909

18 0.013035939284600317 0.001777527179995862

19 0.012952060267950097 0.0020575873402412983

20 0.012787287486717105 0.0016156381574304152

21 0.012684447645830611 0.0016481882250324513

22 0.012588968474107483 0.0016460316043230706

23 0.012516744743722 18 0.0016712106170598418

24 0.012436259812675416 0.0016069989195481563

25 0.012365568635674814 0.0015613818009539196

26 0.012302884308931727 0.0015581149139325134

27 0.012274049241095781 0.0017023185840419803

28 0.012204137419660886 0.0015266430373109567

29 0.012156451462457578 0.0014936629966056595

30 0.01212851781398058 0.001610289132707597

31 0.012071213509577017 0.0014522986875575346

32 0.012053008990672727 0.0015621402253358004

33 0.012011841032654047 0.0014597516699965732

34 0.011990493315582475 0.0015770643193779204

35 0.011954121176774303 0.001499418238333116

36 0.011936877459908525 0.0015856434738573929

37 0.011906243219661217 0.0014902768457735267

38 0.011878996719606221 0.001467744380837151

39 0.011859957490426799 0.001512353336244511

40 0.011853709874364238 0.0016484902066683086

41 0.011817336988945801 0.0014301978093377936

42 0.011804800885729492 0.0015148697438417003

43 0.011773228724487127 0.001359649564352973

44 0.011783483269003 53 0.0016267582650956076

45 0.011759506914143762 0.001462800440688928

46 0.011729988181032 24 0.001438633999302207

47 0.011715639365526538 0.001364115001730776

48 0.011718624581893286 0.001557607182379191

49 0.011716219064158697 0.0016350008107838222

50 0.011687074773944914 0.001434840627577311

51 0.01167863752382497 0.001494942930585239

52 0.01166752969690909 0.0014482160111462387

53 0.01166021294426173 0.0015007127813684443

54 0.011655128790686527 0.001550960160869484

55 0.01163541253656149 0.001420557917347954

56 0.01163053317150722 0.0014825452333510232

57 0.011619872886997958 0.0014856300952184635

58 0.011615197779610753 0.0015259733610825303

59 0.011609001606702805 0.0014923929552120778

60 0.011593749552654723 0.0014235138857232718

61 0.01159503692916284 0.001541170952259563

62 0.011582947011726599 0.0014767773771503319

63 0.011569258814367156 0.001448691239541707

64 0.011565795028582215 0.001465522782042778

65 0.011557429201590518 0.0014145221614550489

66 0.011560439650590221 0.0015513211442157626

67 0.01153536823267738 0.0013490080671423735

68 0.011533799056584637 0.0014287245177547448

69 0.01153676297981292 0.0014869418586992349

70 0.011532367838857074 0.001535292079206556

71 0.01154288194142282 0.0016004445976189649

72 0.011523299409697454 0.0014794031473381135

73 0.01151011060612897 0.0014014885751142477

74 0.011517290025949478 0.0015041712403763086

75 0.01150606885086745 0.001484638568557178

76 0.011504702701543768 0.001456115321101

77 0.011503460955185195 0.00151523165453303

78 0.011488565797917545 0.0013892744579546464

79 0.011489679743535817 0.001444956588287217

80 0.011496397548665603 0.001572806273761671

81 0.011481871965030829 0.001456904747368147

82 0.011471754683492085 0.0013798712188145146

83 0.011471378055090705 0.0014088641608638378

84 0.011479732743464411 0.0016010244474940311

85 0.01145786564797163 0.0013594609734718687

86 0.011456223969968657 0.0014036959190464888

87 0.011463854770796995 0.0015382962111228455

88 0.011455644491749505 0.0014677930007261845

89 0.011436202344484628 0.0012976482913169699

90 0.011453003557398915 0.0015193255330086686

91 0.011445312600893279 0.0014465029393128741

92 0.011438253602633874 0.0014047414414623443

93 0.011446606925067802 0.0015359348416677677

94 0.01143996107392013 0.001509926500536191

95 0.011416009568298857 0.0012686008288680265

96 0.011432749745436012 0.0014885651567601598

97 0.011425680462270976 0.0014690325307310558

98 0.011428086018810669 0.0014588926536574338

99 0.011423216161007683 0.0014674571575596929

100 0.011412743056813876 0.0013815404985992549

101 0.011420872438078126 0.0014738113530135403

102 0.011413774656442305 0.0014382588864342931

103 0.011411812187482913 0.0014419699643622153

104 0.011400692782675227 0.0014173452311661095

105 0.011405025514153143 0.001436905803081269

106 0.011398423789069056 0.0013603808126451138

107 0.011405768364978334 0.0015108998725190759

108 0.011398759962369998 0.0014536250193486922

109 0.011390029303729534 0.0013742651533296642

110 0.011390304109081626 0.001404998198170991

111 0.011394711992082496 0.0014974850180442445

112 0.011381267557541529 0.001328239756306478

113 0.011383061882418891 0.0014204780457657761

114 0.011377625355186561 0.0013723582414483342

115 0.011388037853563826 0.0015534079817977424

116 0.011391466753557324 0.0015068597540569802

117 0.011374463210813701 0.0013512230755683655

118 0.011366692548617 72 0.0013353261441807263

119 0.01137324741575867 0.0014044517005095257

120 0.011370959229146441 0.0014228316153942918

121 0.011364438420472045 0.0013773885465343484

122 0.011374675196905932 0.0015035559077902388

123 0.01135029997676611 0.0012563267944885107

124 0.011359908486095567 0.0013656543650237532

125 0.01136795238746951 0.0015199309569046211

126 0.011355711854994297 0.0013864135069889016

127 0.01134759460731099 0.001332206829683855

128 0.011358492728322744 0.0014819426836523537

129 0.011355886260668437 0.0014075819542631507

130 0.011345925124672552 0.0013643022282243086

131 0.011350420106512805 0.0014130391847963135

132 0.011347031616605819 0.0013696784886027065

133 0.011346341781318188 0.0014173771663142057

134 0.011339064557105302 0.0013629725525970571

135 0.011345120629606147 0.0014050406829725641

136 0.011353178961823383 0.001569373154003794

137 0.011339628848557671 0.0013224939272428553

138 0.011339813671074808 0.0013877887012980258

139 0.011330513328624269 0.0013307699638729295

140 0.011339934797336658 0.0014414410424069503

141 0.011346730973261098 0.0014928099311267335

142 0.011329206034230689 0.001352308799396269

143 0.011331958519294858 0.0013824934849981218

144 0.011335550555959344 0.0014229442037564392

145 0.011324344219950338 0.0013516237964116347

146 0.011340733584947884 0.0014953654950174192

147 0.011322845788672566 0.0013395829126238823

148 0.011329546418661872 0.0014226180788440008

149 0.011315980435659487 0.0012845286822024112

150 0.011321757615854343 0.0013912820855815273

151 0.011331721671546498 0.0014868033128247287

152 0.011320297333101432 0.0013645173484110274

153 0.011313716714891295 0.0013361004470304277

154 0.011316531241560976 0.0014146494654899773

155 0.011314934108716746 0.0013439281517639756

156 0.011314551325825354 0.0014088902349855441

157 0.011310544020185867 0.0013574751674120004

158 0.01130710041616112 0.001328647897268335

159 0.011310698045417667 0.0014017553106532431

160 0.011316956092293063 0.0014708291736315006

161 0.01130272597540 17 0.0012865492983837612

162 0.011306610523412626 0.0014068341478317356

163 0.011294860659788053 0.0012766673090906504

164 0.01130984014676263 0.0014318479583016596

165 0.011298908425184587 0.0013548922535846942

166 0.011295088438006739 0.0013155273819575085

167 0.011306566406662265 0.0014641256190952845

168 0.011300965043095251 0.0013612292656519762

169 0.011296684698512157 0.0013738872618220437

170 0.011295210442816218 0.001347694998451819

171 0.011297893269608419 0.0013611001098373283

172 0.011297138937128087 0.0013972059857527105

173 0.011296407973083357 0.001416569430148229

174 0.011292008891080816 0.0013387702638283372

175 0.01129081844817847 0.0013541192498329716

176 0.011291040463062623 0.0013878470101432565

177 0.011291460199281574 0.0013579843286424876

178 0.011290511271605888 0.001368268989220572

179 0.011294107783275347 0.0013550992969733973

180 0.011286264547767738 0.0013702136334419871

181 0.01128400648944 08 0.0013713198464635448

182 0.011287701072481771 0.0013570371494279242

183 0.011293396460823715 0.0014933938102331013

184 0.011273260107263922 0.0012308476008668853

185 0.011291424608789385 0.0014427462949727972

186 0.01127670811334004 0.0013261182645025354

187 0.011275790949972967 0.0013277332723373547

188 0.011285599463929733 0.0014071354799671098

189 0.011278938236646354 0.001384671822403713

190 0.011270243738157054 0.0012957440583462206

191 0.011277115875855089 0.00132743906074514

192 0.011280698138289153 0.0014067449319797257

193 0.011266219685785473 0.001278120671728781

194 0.011273384938637415 0.0013673053929232992

195 0.01127080481344213 0.001355632471095305

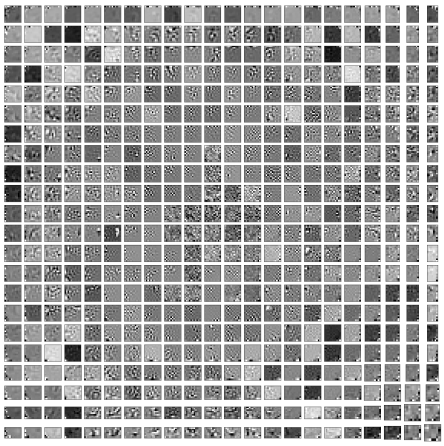
196 0.011279484311429163 0.0014220558867479363

197 0.011260900722506146 0.0012698514487904807

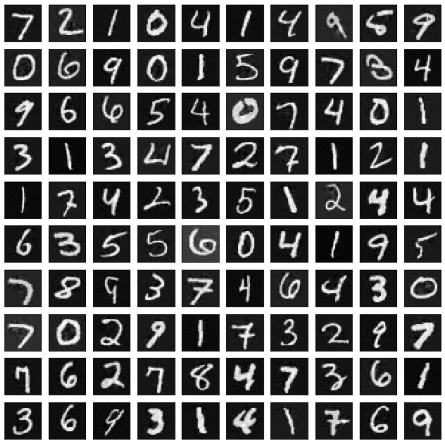
198 0.01128082985834529 0.001444455494347494

199 0.011260258434340358 0.0012516380220768042

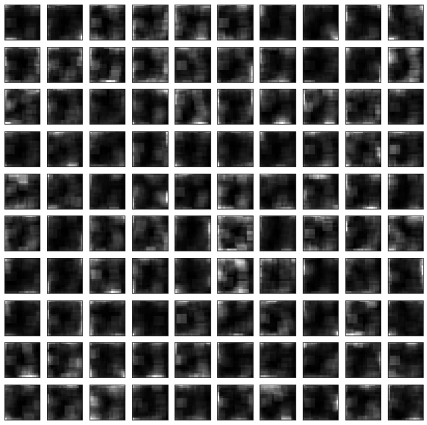
训练得到的参数矩阵 W 可视化结果如下



重建得到的数字图像如下



独立成分的可视化如下



可见重建效果较好，但从独立成分来观察，难以找到肉眼可见的区别

六.附录：

附录代码如下 文件为 train.py 以及 model.py、visualize.py

train.py

```
import chainer
import chainer.functions as F
import chainer.links as L
import chainer.cuda
import numpy as np

from model import ReconstractionTICA
from visualize import visualize, visualize_kernel

# Define constants
l = 1.0  # Reconstruction coefficient
N = 100  # Minibatch size
SNAPSHOT_INTERVAL = 10


def main():
    train, test = chainer.datasets.get_mnist(ndim=3, withlabel=True)

    nn = ReconstractionTICA()

    # Setup optimizers
    optimizer = chainer.optimizers.Adam()
```

```
optimizer.setup(nn)

# (Validate input images)
to_be_show = np.array([t[0] for t in test[:100]])
visualize(to_be_show, 'validation.png', (28, 28))

train = np.array([t[0] for t in train])

# Training
for epoch in range(200):

    # (Validate generated images)
    if (epoch % SNAPSHOT_INTERVAL == 0):
        y, z = nn(chainer.cuda.to_cpu(to_be_show))
        visualize(chainer.cuda.to_cpu(z.data), 'z.png', (28, 28))
        visualize(chainer.cuda.to_cpu(y.data), 'y.png', (18, 18))
        visualize_kernel(chainer.cuda.to_cpu(nn.f.W.data), 'W.png')

    # (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_cpu(train[n:n + N].reshape((N, 1 * 28 * 28)))
        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 + l * 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()
```

model.py

```
import chainer
import chainer.functions as F
import chainer.links as L

import numpy as np

class ReconstractionTICA(chainer.Chain):

    def __init__(self):
        super(ReconstractionTICA, self).__init__()
        with self.init_scope():
            self.f = L.Linear(1 * 28 * 28, 1 * 22 * 22)
            self.p = F.average_pooling_2d
            self.g = L.Linear(1 * 22 * 22, 1 * 28 * 28)
            self.mask = np.zeros((1, 22, 22, 1, 28, 28), 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 * 28 * 28))

    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)
```

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, 28, 28))[0, y, x, 0, y:y+11,
x:x+11], cmap='gray')
    plt.savefig(fname)
    plt.close()
```