P2图像恢复

2018年5月25日 11:19

人工智能课程Project 2: 图像恢复

题目要求

问题：

给定3张受损图像，尝试恢复他们的原始图像。

说明：

1. 原始图像包含1张黑白图像（A.png）和2张彩色图像（B.png, C.png）。
2. 受损图像（X）是由原始图像（$I \in \mathcal{R}^{H*W*C}$）添加了不同噪声遮罩（noise masks）（$M \in \mathcal{R}^{H*W*C}$）得到的（$X = I \odot M$），其中$\odot$是逐元素相乘。
3. 噪声遮罩仅包含{0,1}值。对应原图（A/B/C）的噪声遮罩的每行分别用0.8/0.4/0.6的噪声比率产生的，即噪声遮罩每个通道每行80%/40%/60%的像素值为0，其他为1。
4. Code：目前提供的实现代码，利用逐行回归实现图像恢复，效果一般。鼓励大家开发更有效的恢复方法。
5. Data：提供的三幅待恢复图像，提供给大家。要求设计恢复算法，得到恢复图像，和实验报告一起提交。

命名规则：

给定受损图像：{A,B,C}.png

提交图像格式：{学号}_{A,B,C}.png

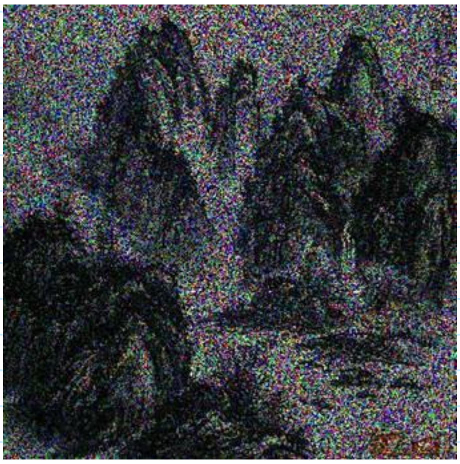
评估：

评估误差为所有恢复图像（R）与原始图像（I）的2-范数之和，此误差越小越好。

$\text{error} = \sum_{i=1}^{3} \text{norm}(R_i(:) - I_i(:), 2)$，其中$(:)$是向量化操作。

示例：

输入受损图像：



输出恢复图像：

原始图像：

计算error：

原始图像与受损图像的误差为349.4518

原始图像与恢复图像的误差为86.1748

Notes：

多变量固定窗口二元高斯回归方法

这道题用了高斯线性回归的方法，其实本质上就是把图片理解成为由多个（题目里面用了50个Phi）不同mean相同方差的高斯函数组合的结果。然后通过没有被噪音影响的像素值，用最小二乘法算出对应每一行的Phi的权重（即每一行的50个函数的权重），然后再用这个权重和被污染像素的分度密度进行计算

1，1	~~1，2~~	1，3	~~1，4~~	1，5

~~2，1~~	2，2	~~2，3~~	2，4	~~2，5~~

*划掉的代表是被污染的

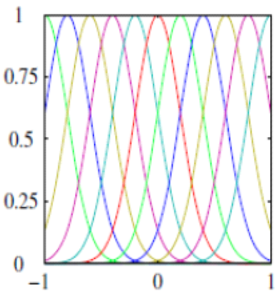
1. 假设有3个高斯函数构成了每一行的图相
也就是说对于每一个像素都是wi*Phii（xi) i=1,2,3 即每一行每个高斯函数对应的权重，每个高斯函数xi时的概率
2. 然后可以通过最小二乘法，利用没有被污染的，也就是说正确的值来计算出权重wi
3. 再用这个权重wi*每个高斯函数xi（被污染的像素）来恢复图相

Basis Function： 在这里用了高斯函数

There are many other possible choices for the basis functions, for example

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\} \tag{3.4}$$

where the μ_j govern the locations of the basis functions in input space, and the parameter s governs their spatial scale. These are usually referred to as 'Gaussian' basis functions, although it should be noted that they are not required to have a probabilistic interpretation, and in particular the normalization coefficient is unimportant because these basis functions will be multiplied by adaptive parameters w_j.



最小二乘

Maximum likelihood and least squares

- Assume: $t = y(\mathbf{x}, \mathbf{w}) + \epsilon$ $\quad y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \phi(\mathbf{x})$
- Thus:

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}) \longrightarrow \mathbb{E}[t|\mathbf{x}] = \int t p(t|\mathbf{x})\,\mathrm{d}t = y(\mathbf{x}, \mathbf{w})$$

- For data set $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and target vector $\mathbf{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$, the likelihood function:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n), \beta^{-1})$$

$$\ln p(\mathbf{t}|\mathbf{w}, \beta) = \sum_{n=1}^{N} \ln \mathcal{N}(t_n|\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n), \beta^{-1}) = \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta \boxed{E_D(\mathbf{w})}$$

SSE: sum-of-squares error function

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n)\}^2 = \frac{1}{2}\|\mathbf{t} - \Phi\mathbf{w}\|^2$$

Maximum likelihood and least squares

Maximum likelihood and least squares

$$\ln p(\mathbf{t}|\mathbf{w},\beta) = \sum_{n=1}^{N} \ln \mathcal{N}(t_n|\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n),\beta^{-1}) = \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta E_D(\mathbf{w})$$

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 = \frac{1}{2}\|\mathbf{t}-\boldsymbol{\Phi}\mathbf{w}\|^2 \qquad \mathbf{w} = (w_0,\ldots,w_{M-1})^{\mathrm{T}}$$

- Solving w by ML:

$$\nabla \ln p(\mathbf{t}|\mathbf{w},\beta) = \sum_{n=1}^{N}\left\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\right\}\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}$$

$$0 = \sum_{n=1}^{N} t_n\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} - \mathbf{w}^{\mathrm{T}}\left(\sum_{n=1}^{N}\boldsymbol{\phi}(\mathbf{x}_n)\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}\right) \Rightarrow \mathbf{w}_{\mathrm{ML}} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t}$$

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \boxed{\phi_0(\mathbf{x}_2) \quad \phi_1(\mathbf{x}_2) \quad \cdots \quad \phi_{M-1}(\mathbf{x}_2)} & & & \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix} = \begin{pmatrix} \boldsymbol{\phi}(\mathbf{x}_1)^{\mathrm{T}} \\ \boldsymbol{\phi}(\mathbf{x}_2)^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\phi}(\mathbf{x}_N)^{\mathrm{T}} \end{pmatrix}$$

N×M design matrix

$$\boldsymbol{\Phi}^{\dagger} \equiv \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}$$

Moore-Penrose pseudo-inverse

1. 线性回归

https://www.cnblogs.com/GuoJiaSheng/p/3928160.html

https://www.jianshu.com/p/4d2f6052ab7d

从只含一个自变量X的情况开始，成为一元线性回归。假定回归模型为
Y=b0+b1X+e
b0,b1为未知参数，b0为常数项或截距，b1为回归系数，e为随机误差。实际工作中，也常使用变量变换法。即在散点图与直线趋势差距较大时，设法对自变量乃至因变量进行适当的变换，使变换后的散点图更加接近与直线。这样对变化后的新变量进行线性回归分析，再回到原变量。
假定Y = b0+b1X+e
л和э分别为x和y的算术平均，故可以改写为Yi=β0+β1(Xi-л)+ei (i = 1,....,n)
其关系是β0=b0+b1л,β1=b1，故如估计了β0和β1，可以得到b0和b1的估计。
当X=Xi处取值为Yi,估计值我们有Г表示，这样我们就有偏离Yi-Г，我们当然希望偏离越小越好。衡量这种偏离大小的一个合理的单一指标为他们的平方和（通过平方去掉符号的影响，若简单求和，则正负偏离抵消了）

2. 高斯过程回归

https://www.cnblogs.com/tornadomeet/archive/2013/06/15/3137239.html

https://zhuanlan.zhihu.com/p/24388992

3. 课程参考：

https://blog.csdn.net/Woolseyyy/article/details/72663125

4. python实现

https://www.cnblogs.com/lyrichu/p/7814651.html
Numpy's array 类被称为ndarray。 这个对象常用而重要的属性如下：

- ndarray.ndim： 输出矩阵（数组）的维度
- ndarray.shape： 输出矩阵的各维数大小，相当于matlab中的size()函数
- ndarray.size： 输出矩阵（数组）元素的总个数，相当于各维数之积
- ndarray.dtype： 输出矩阵元素的类型，如int16, int32, float64等
- ndarray.itemsize： 输出矩阵中每个元素所占字节数

来自 <https://www.zybuluo.com/chanvee/note/89078>

5. 掩码矩阵
https://docs.scipy.org/doc/numpy/reference/generated/numpy.ma.masked_values.html
https://blog.csdn.net/GZHermit/article/details/76163934

```
a = np.array([0,1,4,3,4,-9999,33,34,-9999])
am = np.ma.MaskedArray(a)
am.mask = (am==-9999)
np.ma.set_fill_value(am, 0)

z = np.arange(35)

print z[am.filled()]
```

```
import numpy as np

arr = np.random.randint(1, 10, size=[1, 5, 5])
mask = arr<5
arr[mask] = 0 # 把标记为True的值记为0
print(arr)

#[[[9 9 7 6 0]
#  [0 0 6 9 0]
#  [8 0 8 5 0]
#  [0 5 5 8 9]
#  [0 7 0 0 6]]]
```

6. linspace
>>X=linspace(5,100,20) % 产生从5到100范围内的20个数据，相邻数据跨度相同
来自 <https://zhidao.baidu.com/question/110822829.html>

https://blog.csdn.net/yuan1125/article/details/69925720

7. find

python的第三方库numpy（用于矩阵运算，需要import numpy as np）中可以使用np.where，如

```
>>a = np.array(a)
>>a
array([1, 2, 3, 1, 2, 3, 1, 2, 3])
>>idx = np.where(a > 2)
>>idx
(array([2, 5, 8], dtype=int32),)
```

8.矩阵组合

1.水平组合

```
>>> np.hstack((a,b))
array( [ 0, 1, 2, 0, 2, 4],
       [ 3, 4, 5, 6, 8, 10],
       [ 6, 7, 8, 12, 14, 16] )
>>> np.concatenate((a,b),axis=1)
array( [ 0, 1, 2, 0, 2, 4],
       [ 3, 4, 5, 6, 8, 10],
       [ 6, 7, 8, 12, 14, 16] )
```

2.垂直组合

```
>>> np.vstack((a,b))
array( [ 0, 1, 2],
       [ 3, 4, 5],
       [ 6, 7, 8],
       [ 0, 2, 4],
       [ 6, 8, 10],
       [12, 14, 16] )
>>> np.concatenate((a,b),axis=0)
array( [ 0, 1, 2],
       [ 3, 4, 5],
       [ 6, 7, 8],
       [ 0, 2, 4],
       [ 6, 8, 10],
       [12, 14, 16] )
```

xy等于x,y的合并,冒号表示所有元素,则xy=[X(:) Y(:)];表示将x的所有元素作为第一列，y的所有元素作为第二列，形成的xy是一个2列的矩阵；比如例子：

```
 x=[1 2 3;4 5 6;7 8 9]

x =

     1     2     3
     4     5     6
     7     8     9
```

x是个3*3的矩阵，在存储时,按列存储为[1 4 7 2 5 8 3 6 9]

所以x(5)=x(2,2)=5,所以以下面的合并会按这个顺序来。

```
>> y=rand(3)

y =

    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575

>> xy=[x(:),y(:)]
```

9. 图片的显示和存储

https://blog.csdn.net/zywvvd/article/details/72810360