AI Expert 프로그램 실습

9/11 Graphical Models, Gaussian Process, Hawkes Process

Overview

This tutorial is three-fold as follows:

1. Graphical Models - 80 min.
2. Gaussian Process (GP) - 80 min.
3. Hawkes Process (HP) - 80 min.

* 10 minutes break between each part.

Environments

1. Python 3.6
2. virtualenv
3. Gpy
4. Matplotlib
5. Scipy
6. Image
7. Tqdm
8. Ipykernel
9. Tick

   Module for statistical learning, with a particular emphasis on time-dependent modelling

Part 0. Environment Setting

Download the source code

```
cogito@digits-1:~$ git clone https://github.com/cogito288/samsung-ds-kaist.git
```

Part 0. Environment Setting

Create the virtual environment and activate it

```
cogito@digits-1:~$ virtualenv -p python3 samdung-ds-0710-env
Already using interpreter /usr/bin/python3
Using base prefix '/usr'
New python executable in /home/cogito/samdung-ds-0710-env/bin/python3
Also creating executable in /home/cogito/samdung-ds-0710-env/bin/python
Installing setuptools, pkg_resources, pip, wheel...done.
cogito@digits-1:~$ source samdung-ds-0710-env/bin/activate
(samdung-ds-0710-env) cogito@digits-1:~$
```

Install ipykernel package

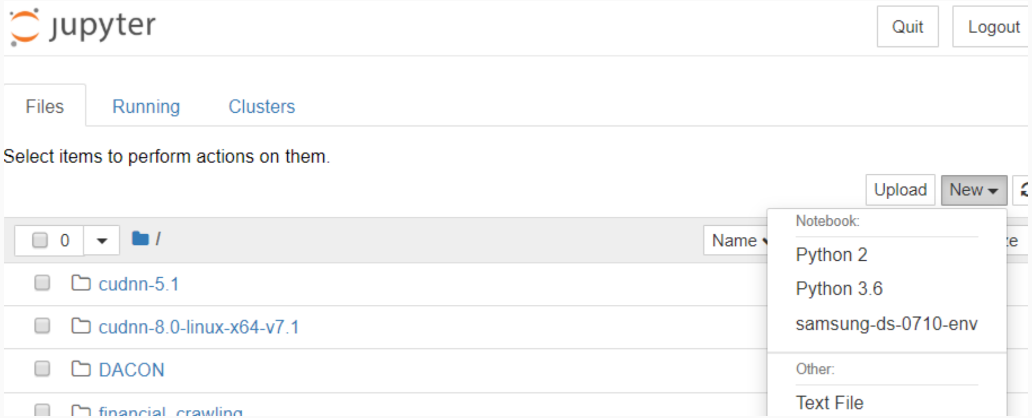
```
(samdung-ds-0710-env) cogito@digits-1:~$ pip install ipykernel
(samdung-ds-0710-env) cogito@digits-1:~$ pip install matplotlib
(samdung-ds-0710-env) cogito@digits-1:~$ pip install scipy==1.0.0
(samdung-ds-0710-env) cogito@digits-1:~$ pip install image
(samdung-ds-0710-env) cogito@digits-1:~$ pip install tqdm
```

Part 0. Environment Setting

Add virtualenv to Jupyter kernel

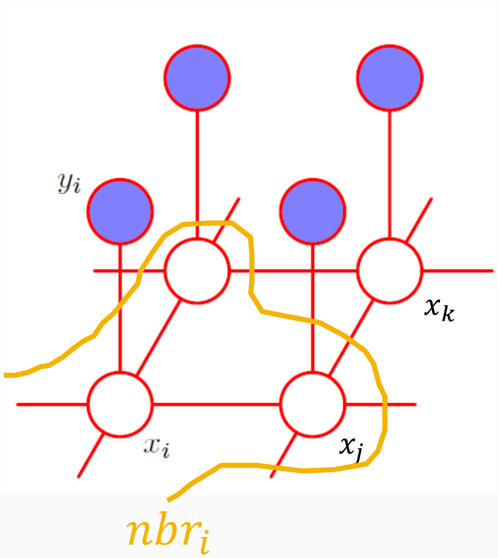
```
cogito@digits-1:~$ python3 -m ipykernel install --user  \
--name samsung-ds-0710-env --display-name "samsung-ds-0710-env“
Installed kernelspec samsung-ds-0710-env in /home/cogito/.local/share/jupyter/kernels/samsung-ds-0710-env
```

```
cogito@digits-1:~$ jupyter notebook
```



Part 1. Graphical Models (recap)

- Markov Random Field

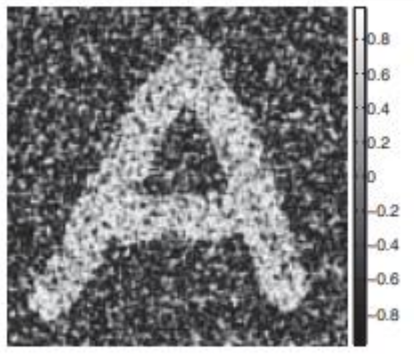


- Markov Property
  - $p(x_i, x_j) \neq p(x_i)p(x_j)$ for $x_j \in nbr_i$
  - $p(x_i, x_k) = p(x_i)p(x_k)$ for $x_k \notin nbr_i$

- Energy Based Model
  - $p(x) = \frac{1}{Z}\exp(-E(x))$

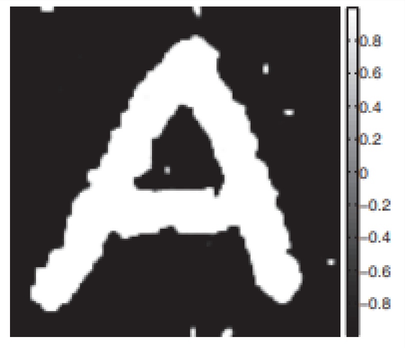
Part 1. Graphical Models (recap)

- Image Denoising using Ising Model



Noisy Image

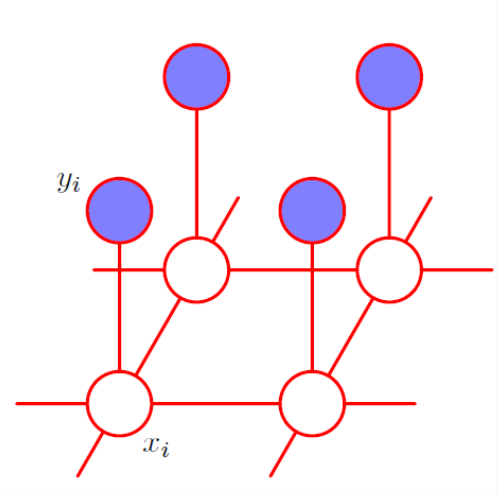
Denoising



Clean Image

Part 1. Graphical Models (recap)

- Ising Model



- y_i: noisy pixel value for i^{th} pixel
- x_i: binary state value for i^{th} pixel $\in \{-1, 1\}$
- $nbr_i = \{x_{i\leftarrow}, x_{i\rightarrow}, x_{i\uparrow}, x_{i\downarrow}\}$

- $p(x) = \frac{1}{Z_0} \exp\left(-E_0(x)\right)$
- $E_0(x) = -\sum_{i=1}^{D} \sum_{j \in nbr_i} W_{ij} x_i x_j$

Part 1. Graphical Models (recap)

- Ising Model
    - $p(x) = \frac{1}{Z_0}\exp\left(-E_0(x)\right)$
    - $E_0(x) = -\sum_{i=1}^{D}\sum_{j \in nbr_i} W_{ij}x_i x_j$

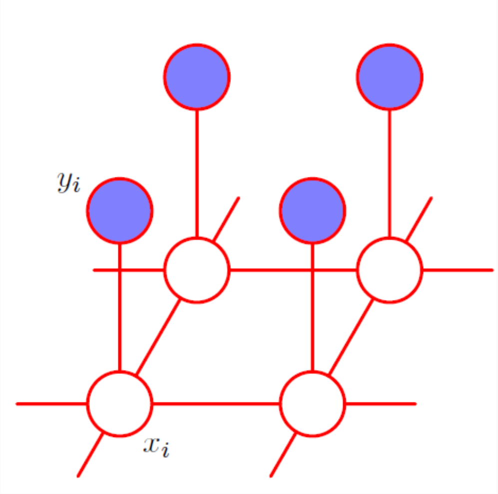
$x_j \backslash x_i$	-1	1
-1	$-W_{ij}$	W_{ij}
1	W_{ij}	$-W_{ij}$

$E_0(x)$

- $W_{ij} > 0$
    - When $x_i = x_j$, $p(x)$ is high.
    - When $x_i \neq x_j$, $p(x)$ is small
- $W_{ij} < 0$
    - When $x_i \neq x_j$, $p(x)$ is high.
    - When $x_i = x_j$, $p(x)$ is small

Part 1. Graphical Models (recap)

- Ising Model



- y_i: noisy pixel value for i^{th} pixel
- x_i: binary state value for i^{th} pixel $\in \{-1, 1\}$
- $nbr_i = \{x_{i\leftarrow}, x_{i\rightarrow}, x_{i\uparrow}, x_{i\downarrow}\}$

- $p(x) = \frac{1}{Z_0} \exp\left(-E_0(x)\right)$
- $E_0(x) = -\sum_{i=1}^{D} \sum_{j \in nbr_i} W_{ij} x_i x_j \quad (\text{set } W_{ij} = 1)$
  $= -\sum_{i=1}^{D} \sum_{j \in nbr_i} x_i x_j$
- $p(y|x) = \prod_i p(y_i|x_i) \qquad (\text{Markov property})$
  $= \prod_i N(y_i|\, x_i)$

Part 1. Graphical Models (recap)

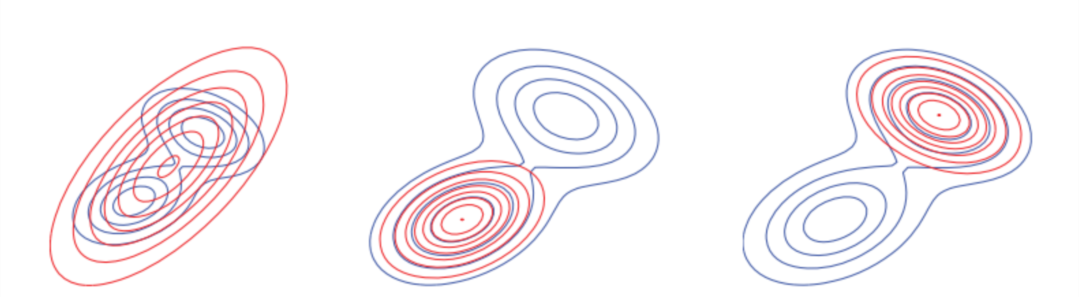
- $p(x) = \frac{1}{Z_0}\exp\left(-E_0(x)\right)$
- $E_0(x) = -\sum_{i=1}^{D}\sum_{j\in nbr_i} x_i x_j$
- $p(y|x) = \prod_i N(y_i|\, x_i)$

- $p(x|y) = \frac{1}{Z} p(y|x)p(x) = \frac{1}{Z}\exp(-\mathrm{E}_0(x) + \log\prod_i N(y_i|\, x_i))$

$$= \frac{1}{Z}\exp\left(\sum_{i=1}^{D}\sum_{j\in nbr_i} x_i x_j + \log\prod_i N(y_i|\, x_i)\right)$$

Part 1. Graphical Models (recap)

- Variational Inference
  - Approximate intractable distribution $p(x)$ using tractable distribution $q(x)$.
    Target distribution　　Proposal distribution
- Mean field approximation
  - $q(x) = \prod_i q_i(x_i)$
- Example:
  - $p(x)$: Unknown distribution
  - $q(x)$: Normal distribution



Part 1. Graphical Models (recap)

- Variational inference for Ising model
  - Target distribution: $p(x|y)$
  - Proposal distribution: $q(x)$
- Mean field approximation
  - $q(x) = \prod_i q(x_i, \mu_i)$ where μ_i is mean value for x_i
- $q_i(x_i) = \frac{1}{Z_i}\exp\left(\mathrm{E}_{-q_i}[\log p(x|y)]\right)$
  - $\log(p(x|y)) = \sum_{i=1}^{D}\sum_{j\in nbr_i} x_i x_j + \log\prod_i N(y_i|\, x_i) + const$
  - $\mathrm{E}_{-q_i}[\log p(x|y)] = \mathrm{E}_{-q_i}[\sum_{i=1}^{D}\sum_{j\in nbr_i} x_i x_j + \log\prod_i N(y_i|\, x_i) + const]$

$$= x_i \sum_{j\in nbr_i} \mathrm{E}_{\mathrm{q_j}}[x_j] + \log N(y_i|x_i) + const = x_i \sum_{j\in nbr_i} \mu_j + \log N(y_i|x_i) + const$$

Part 1. Graphical Models (recap)

- $q_i(x_i) = \frac{1}{Z_i} \exp\left(\mathrm{E}_{-q_i}[\log p(x|y)]\right)$
  - $\mathrm{E}_{-q_i}[\log p(x|y)] = x_i \sum_{j\in nbr_i} \mu_j + \log N(y_i|x_i) + const$
- $q_i(x_i) \propto \exp\left(x_i \sum_{j\in nbr_i} \mu_j + \log N(y_i|x_i)\right)$
- $q_I(x_i = 1) = \frac{\exp\left(\sum_{j\in nbr_i} \mu_j + \log N(y_i|1)\right)}{\exp\left(\sum_{j\in nbr_i} \mu_j + \log N(y_i|1)\right) + \exp\left(-\sum_{j\in nbr_i} \mu_j + \log N(y_i|-1)\right)}$

$$= \frac{1}{1 + \exp\left(-2\sum_{j\in nbr_i} \mu_j + \log N(y_i|-1) - \log N(y_i|1)\right)} = \mathrm{sigmoid}(2\mathrm{a_i})$$

$$\mathrm{a_i} = \sum_{j\in nbr_i} \mu_j + 0.5 * (\log N(y_i|1) \ - \log N(y_i|-1))$$

Part 1. Graphical Models (recap)

- $q_i(x_i = 1) = \mathrm{sigmoid}(2\mathrm{a_i})$
  - $\mathrm{a_i} = \sum_{j \in nbr_i} \mu_j + 0.5 * (\log N(y_i|1) \ - \log N(y_i|-1))$
- $q_i(x_i = -1) = \mathrm{sigmoid}(-2\mathrm{a_i})$
- $\mu_i = \mathrm{E}_{q_i}[x_i] = (+1) \cdot q_i(x_i = 1) + (-1) \cdot q_i(x_i = -1)$

$$= \frac{1}{1+\exp(-2a_i)} - \frac{1}{1+\exp(2a_i)} = \frac{\exp(a_i)}{\exp(a_i)+\exp(-a_i)} - \frac{\exp(-a_i)}{\exp(-a_i)+\exp(a_i)}$$

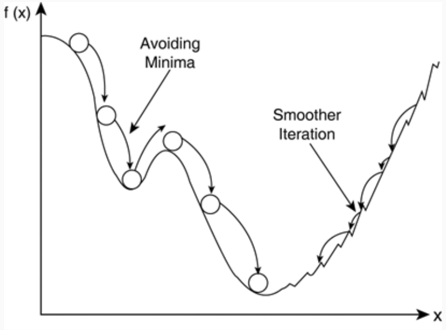
$$= \frac{\exp(a_i)-\exp(-a_i)}{\exp(a_i)+\exp(-a_i)} = \tanh a_i$$

Part 1. Graphical Models (recap)

- Update variational parameter μ_i
  - $\mu_i = \tanh(a_i) = \tanh\left(\sum_{j \in nbr_i} \mu_j + 0.5 * (\log N(y_i|1) - \log N(y_i|-1))\right)$
  - $\mu_i^t = \tanh\left(\sum_{j \in nbr_i} \mu_j^{t-1} + 0.5 * (\log N(y_i|1) - \log N(y_i|-1))\right)$

(fixed point algorithm)

  - $\mu_i^t = (1-\lambda)\mu_j^{t-1} + \lambda \tanh\left(\sum_{j \in nbr_i} \mu_j^{t-1} + 0.5 * (\log N(y_i|1) - \log N(y_i|-1))\right)$



(damped update)

Part 1. Graphical Models (code)

- Import library

```
%matplotlib inline
import matplotlib.pyplot as plt
from PIL import Image

import numpy as np
from scipy.special import expit as sigmoid
from scipy.stats import multivariate_normal

from tqdm import tqdm

import copy
```

To visualize data (e.g. plot)

To handle image data (e.g. open)

Mathematical library/function/class

To visualize progress (e.g. iteration of loop)

For deep copy of array

Part 1. Graphical Models (code)

- Load image file as array

```
print('Loading Image ...')
img_orig = np.double(Image.open('./samsung.jpg').resize((288, 140)))[:,:,3]

plt.figure()
plt.imshow(img_orig, cmap='gray')
plt.title("original image")
```

Load image from jpg file

Part 1. Graphical Models (code)

- Load image file as array

Resize image for fast experiment

```
print('Loading Image ...')
img_orig = np.double(Image.open('./samsung.jpg').resize((288, 140)))[:,:,3]

plt.figure()
plt.imshow(img_orig, cmap='gray')
plt.title("original image")
```

Load image from jpg file

Part 1. Graphical Models (code)

- Load image file as array

Type cast from image to double array

Resize image for fast experiment

```
print('Loading Image ...')
img_orig = np.double(Image.open('./samsung.jpg').resize((288, 140)))[:,:,3]
```

Load image from jpg file

```
plt.figure()
plt.imshow(img_orig, cmap='gray')
plt.title("original image")
```

Part 1. Graphical Models (code)

- Load image file as array

Type cast from image to double array

Resize image for fast experiment

```
print('Loading Image ...')
img_orig = np.double(Image.open('./samsung.jpg').resize((288, 140)))[:,:,3]

plt.figure()
plt.imshow(img_orig, cmap='gray')
plt.title("original image")
```

Load image from jpg file

Height x Width X RGBA

Part 1. Graphical Models (code)

- Load image file as array

Type cast from image to double array

Resize image for fast experiment

```
print('Loading Image ...')
img_orig = np.double(Image.open('./samsung.jpg').resize((288, 140)))[:,:,3]

plt.figure()
plt.imshow(img_orig, cmap='gray')
plt.title("original image")
```

Load image from jpg file

Height x Width X RGBA

1. Open new window
2. Plot image gray color map
3. Set title to plot

Part 1. Graphical Models (code)

- Binarize pixel value of image

Compute mean pixel value of image

```
print('Binarize image ...')
img_mean = np.mean(img_orig)
img_binary = (+1)*(img_orig>img_mean) + (-1)*(img_orig<img_mean)
[H, W] = img_binary.shape

plt.figure()
plt.imshow(img_binary, cmap='gray')
plt.title("binary image")
```

Binarize image based on mean pixel value

Get image size (Height and Width)

Part 1. Graphical Models (code)

- Generate noisy image

Set the noise level
(standard deviation=2.0)

```
print('Generate noisy image ...')
sigma  = 2.0
y = img_binary + sigma*np.random.randn(H, W)

plt.figure()
plt.imshow(y, cmap='gray')
plt.title("noisy image")
```

Add noise ($N(0,\sigma^2)$)
to binarized image

$N(0,\sigma^2) = \sigma N(0,1)$

Part 1. Graphical Models (code)

- Build Ising model

x_j \ x_i	-1	1
-1	$-W_{ij}$	W_{ij}
1	W_{ij}	$-W_{ij}$

$E_0(x)$

```
W_ij = 1.0
dampling_factor = 0.5
max_iter = 15

normal_pos = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=+1, cov=sigma**2), (H, W))
normal_neg = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=-1, cov=sigma**2), (H, W))
normal_all = normal_pos - normal_neg

mu = y

a = mu + 0.5*normal_all
prob_x_pos = sigmoid(+2*a)
prob_x_neg = sigmoid(-2*a)
```

- $\mu_i^t = \tanh\left(\sum_{j\in nbr_i}\mu_j^{t-1} + 0.5 * (\log N(y_i|1) - \log N(y_i|-1))\right)$

(fixed point algorithm)

- $\mu_i^t = (1-\lambda)\mu_j^{t-1} + \lambda\tanh\left(\sum_{j\in nbr_i}\mu_j^{t-1} + 0.5 * (\log N(y_i|1) - \log N(y_i|-1))\right)$

(damped update)

Part 1. Graphical Models (code)

- Build Ising model

$x_j \backslash x_i$	-1	1
-1	$-W_{ij}$	W_{ij}
1	W_{ij}	$-W_{ij}$

$E_0(x)$

```
W_ij = 1.0
dampling_factor = 0.5
max_iter = 15

normal_pos = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=+1, cov=sigma**2), (H, W))
normal_neg = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=-1, cov=sigma**2), (H, W))
normal_all = normal_pos - normal_neg

mu = y

a = mu + 0.5*normal_all
prob_x_pos = sigmoid(+2*a)
prob_x_neg = sigmoid(-2*a)
```

- $\mu_i^t = \tanh\left(\sum_{j\in nbr_i} \mu_j^{t-1} + 0.5 * (\log N(y_i|1) - \log N(y_i|-1))\right)$

(fixed point algorithm)

- $\mu_i^t = (1-\lambda)\mu_j^{t-1} + \lambda \tanh\left(\sum_{j\in nbr_i} \mu_j^{t-1} + 0.5 * (\log N(y_i|1) - \log N(y_i|-1))\right)$

(damped update)

Part 1. Graphical Models (code)

- Build Ising model

$x_j \backslash x_i$	-1	1
-1	$-W_{ij}$	W_{ij}
1	W_{ij}	$-W_{ij}$

$E_0(x)$

```
W_ij = 1.0
damping_factor = 0.5
max_iter = 15

normal_pos = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=+1, cov=sigma**2), (H, W))
normal_neg = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=-1, cov=sigma**2), (H, W))
normal_all = normal_pos - normal_neg

mu = y

a = mu + 0.5*normal_all
prob_x_pos = sigmoid(+2*a)
prob_x_neg = sigmoid(-2*a)
```

1. $\log N(y_i|1)$
2. $\log N(y_i|-1)$
3. $\log N(y_i|1) - \log N(y_i|-1)$

Part 1. Graphical Models (code)

- Build Ising model

$x_j \backslash x_i$	-1	1
-1	$-W_{ij}$	W_{ij}
1	W_{ij}	$-W_{ij}$

$E_0(x)$

```
W_ij = 1.0
dampling_factor = 0.5
max_iter = 15

normal_pos = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=+1, cov=sigma**2), (H, W))
normal_neg = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=-1, cov=sigma**2), (H, W))
normal_all = normal_pos - normal_neg

mu = y

a = mu + 0.5*normal_all
prob_x_pos = sigmoid(+2*a)
prob_x_neg = sigmoid(-2*a)
```

Initial μ = y

1. $\log N(y_i|1)$
2. $\log N(y_i|-1)$
3. $\log N(y_i|1) - \log N(y_i|-1)$

Part 1. Graphical Models (code)

- Build Ising model

$x_j \backslash x_i$	-1	1
-1	$-W_{ij}$	W_{ij}
1	W_{ij}	$-W_{ij}$

$E_0(x)$

```
W_ij = 1.0
dampling_factor = 0.5
max_iter = 15

normal_pos = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=+1, cov=sigma**2), (H, W))
normal_neg = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=-1, cov=sigma**2), (H, W))
normal_all = normal_pos - normal_neg

mu = y

a = mu + 0.5*normal_all
prob_x_pos = sigmoid(+2*a)
prob_x_neg = sigmoid(-2*a)
```

Initial μ = y

1. $\log N(y_i|1)$
2. $\log N(y_i|-1)$
3. $\log N(y_i|1) - \log N(y_i|-1)$

- $q_i(x_i = 1) = \mathrm{sigmoid}(2a_i)$
  - $a_i = \sum_{j \in nbr_i} \mu_j + 0.5 * (\log N(y_i|1) - \log N(y_i|-1))$
- $q_i(x_i = -1) = \mathrm{sigmoid}(-2a_i)$

Part 1. Graphical Models (code)

- Mean field variational inference

Visualize loop progress
(e.g. time per loop)

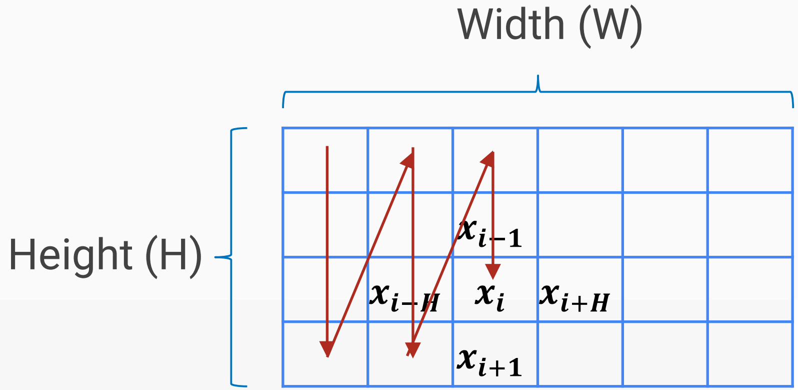
Deep copy for array

```
for i in tqdm(range(max_iter)):
    prev_mu = copy.deepcopy(mu)

    for w in range(W):
        for h in range(H):
            position = H*w + h
            nbr = position + np.array([-1,1,-H,H])
            boundary_idx = [h!=0,h!=H-1,w!=0,w!=W-1]
            nbr = nbr[np.where(boundary_idx)[0]]
            xi_h, xi_w = np.unravel_index(position, (H,W), order='F')
            nbr_h, nbr_w = np.unravel_index(nbr, (H,W), order='F')
            mu[xi_h,xi_w] = (1-dampling_factor)*prev_mu[xi_h,xi_w] ₩
                        + dampling_factor*np.tanh(W_ij*np.sum(prev_mu[nbr_h,nbr_w]) + 0.5*normal_all[xi_h,xi_w])
```

```
>>> foo = [0, 1, 2]
>>> bar = foo
>>> foo[0] = 9
>>> print bar
[9, 1, 2]
```

- Mean field variational inference




```
for i in tqdm(range(max_iter)):
    prev_mu = copy.deepcopy(mu)

    for w in range(W):
        for h in range(H):
            position = H*w + h
            nbr = position + np.array([-1,1,-H,H])
            boundary_idx = [h!=0,h!=H-1,w!=0,w!=W-1]
            nbr = nbr[np.where(boundary_idx)[0]]
            xi_h, xi_w = np.unravel_index(position, (H,W), order='F')
            nbr_h, nbr_w = np.unravel_index(nbr, (H,W), order='F')
            mu[xi_h,xi_w] = (1-dampling_factor)*prev_mu[xi_h,xi_w] ₩
                        + dampling_factor*np.tanh(W_ij*np.sum(prev_mu[nbr_h,nbr_w]) + 0.5*normal_all[xi_h,xi_w])
```

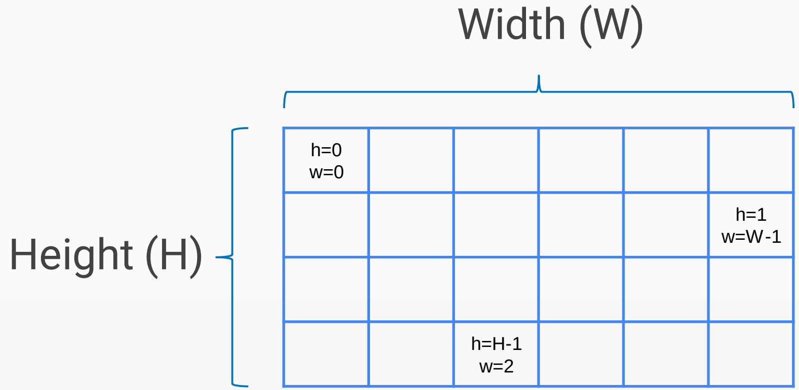
Loop for each pixel

Current pixel position (x_i)

nbr_i pixel position

Part 1. Graphical Models (code)

- Mean field variational inference




```python
for i in tqdm(range(max_iter)):
    prev_mu = copy.deepcopy(mu)

    for w in range(W):
        for h in range(H):
            position = H*w + h
            nbr = position + np.array([-1,1,-H,H])
            boundary_idx = [h!=0,h!=H-1,w!=0,w!=W-1]
            nbr = nbr[np.where(boundary_idx)[0]]
            xi_h, xi_w = np.unravel_index(position, (H,W), order='F')
            nbr_h, nbr_w = np.unravel_index(nbr, (H,W), order='F')
            mu[xi_h,xi_w] = (1-dampling_factor)*prev_mu[xi_h,xi_w] ₩
                              + dampling_factor*np.tanh(W_ij*np.sum(prev_mu[nbr_h,nbr_w]) + 0.5*normal_all[xi_h,xi_w])
```

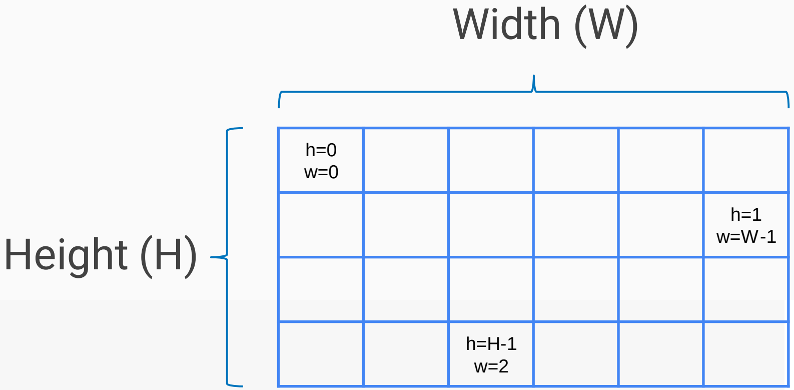
Loop for each pixel

Current pixel position (x_i)

Check whether current position is boundary

Part 1. Graphical Models (code)

- Mean field variational inference




```python
for i in tqdm(range(max_iter)):
    prev_mu = copy.deepcopy(mu)

    for w in range(W):
        for h in range(H):
            position = H*w + h
            nbr = position + np.array([-1,1,-H,H])
            boundary_idx = [h!=0,h!=H-1,w!=0,w!=W-1]
            nbr = nbr[np.where(boundary_idx)[0]]
            xi_h, xi_w = np.unravel_index(position, (H,W), order='F')
            nbr_h, nbr_w = np.unravel_index(nbr, (H,W), order='F')
            mu[xi_h,xi_w] = (1-dampling_factor)*prev_mu[xi_h,xi_w] \
                                    + dampling_factor*np.tanh(W_ij*np.sum(prev_mu[nbr_h,nbr_w]) + 0.5*normal_all[xi_h,xi_w])
```

Loop for each pixel

Current pixel position (x_i)

Delete meaningless element of nbr_i

Part 1. Graphical Models (code)

- Mean field variational inference

```
[[ 0,   1,   2,   3,   4,   5],
 [ 6,   7,   8,   9,  10,  11],
 [12,  13,  14,  15,  16,  17],
 [18,  19,  20,  21, *22*, 23],   <- (3, 4)
 [24,  25,  26,  27,  28,  29],
 [30,  31,  32,  33,  34,  35],
 [36, *37*, 38,  39,  40, *41*]]
      (6, 1)              (6,5)
```

```
>>> np.unravel_index([22, 41, 37], (7,6))
(array([3, 6, 6]), array([4, 5, 1]))
```

```python
for i in tqdm(range(max_iter)):
    prev_mu = copy.deepcopy(mu)

    for w in range(W):
        for h in range(H):
            position = H*w + h
            nbr = position + np.array([-1,1,-H,H])
            boundary_idx = [h!=0,h!=H-1,w!=0,w!=W-1]
            nbr = nbr[np.where(boundary_idx)[0]]
            xi_h, xi_w = np.unravel_index(position, (H,W), order='F')
            nbr_h, nbr_w = np.unravel_index(nbr, (H,W), order='F')
            mu[xi_h,xi_w] = (1-dampling_factor)*prev_mu[xi_h,xi_w] \
                          + dampling_factor*np.tanh(W_ij*np.sum(prev_mu[nbr_h,nbr_w]) + 0.5*normal_all[xi_h,xi_w])
```

Loop for each pixel

Current pixel position (x_i)

1. i (current position, 1D) → (h, w) (2D coordinate)
2. nbr_i (nbr_i, 1D) → (h, w) (2D coordinate)

Part 1. Graphical Models (code)

- Mean field variational inference

```
[[ 0,   1,   2,   3,   4,   5],
 [ 6,   7,   8,   9,  10,  11],
 [12,  13,  14,  15,  16,  17],
 [18,  19,  20,  21, *22*, 23],   <- (3, 4)
 [24,  25,  26,  27,  28,  29],
 [30,  31,  32,  33,  34,  35],
 [36, *37*, 38,  39,  40, *41*]]
      (6, 1)                (6,5)

>>> np.unravel_index([22, 41, 37], (7,6))
(array([3, 6, 6]), array([4, 5, 1]))
```

```
for i in tqdm(range(max_iter)):
    prev_mu = copy.deepcopy(mu)

    for w in range(W):
        for h in range(H):
            position = H*w + h
            nbr = position + np.array([-1,1,-H,H])
            boundary_idx = [h!=0,h!=H-1,w!=0,w!=W-1]
            nbr = nbr[np.where(boundary_idx)[0]]
            xi_h, xi_w = np.unravel_index(position, (H,W), order='F')
            nbr_h, nbr_w = np.unravel_index(nbr, (H,W), order='F')
            mu[xi_h,xi_w] = (1-dampling_factor)*prev_mu[xi_h,xi_w] \
                          + dampling_factor*np.tanh(W_ij*np.sum(prev_mu[nbr_h,nbr_w]) + 0.5*normal_all[xi_h,xi_w])
```

Loop for each pixel

Current pixel position (x_i)

1. i (current position, 1D) $\rightarrow (h, w)$ (2D coordinate)
2. nbr_i (nbr_i, 1D) $\rightarrow (h, w)$ (2D coordinate)

$$\mu_i^t = (1-\lambda)\mu_j^{t-1} + \lambda \tanh\left(\Sigma_{j\in nbr_i} \mu_j^{t-1} + 0.5 * (\log N(y_i|1) \ - \log N(y_i|-1))\right)$$

(damped update)

Reference

[1] K. Murphy, “Machine Learning: A Probabilistic Perspective”, The MIT Press, 2012
[2] https://towardsdatascience.com/variational-inference-ising-model-6820d3d13f6a
[3] https://github.com/vsmolyakov/experiments_with_python/blob/master/chp02/mean_field_mrf.ipynb