Mean Field Inference

Yu Che Wang / yuchecw2

December 4, 2018

1 Introduction

In this project, I used the MNIST dataset to implement Mean Field Inference. I performed 3 operations on each image: 1. Binarize by mapping any value below 0.5 to -1 and any value above to 1 2. Add noise randomly by flipping 2% of the bits 3. Denoise using Boltzman machine model and mean field inference.

2 Average accuracy

The accuracy for each denoised image is obtained by calculating the ratio of pixels with correct value. The average accuracy on first 500 images is 0.9836.

3 Sample images

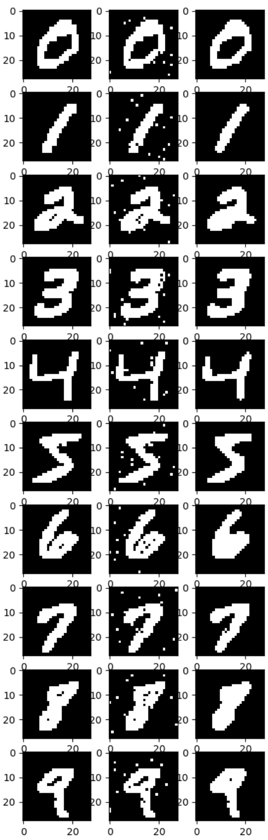

Figure 1: Sample images for each digit. The first column is sample image, the second column is noised version, and the third column is denoised version.

4 Best reconstruction

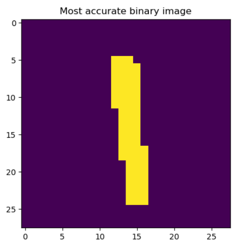

Figure 2: Original image of the best reconstruction

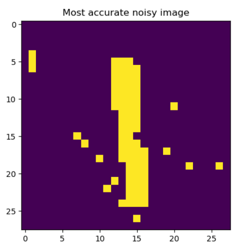

Figure 3: Noised version of the best reconstruction

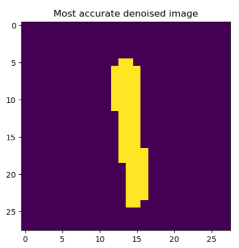

Figure 4: Denoised version of the best reconstruction

5 Worst reconstruction

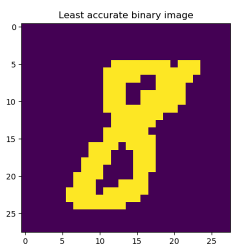

Figure 5: Original image of the worst reconstruction

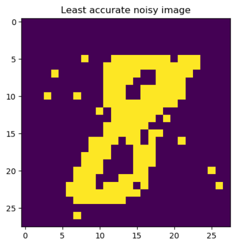

Figure 6: Noised version of the worst reconstruction

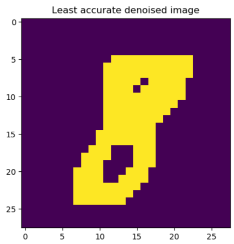

Figure 7: Denoised version of the worst reconstruction

6 ROC Curve

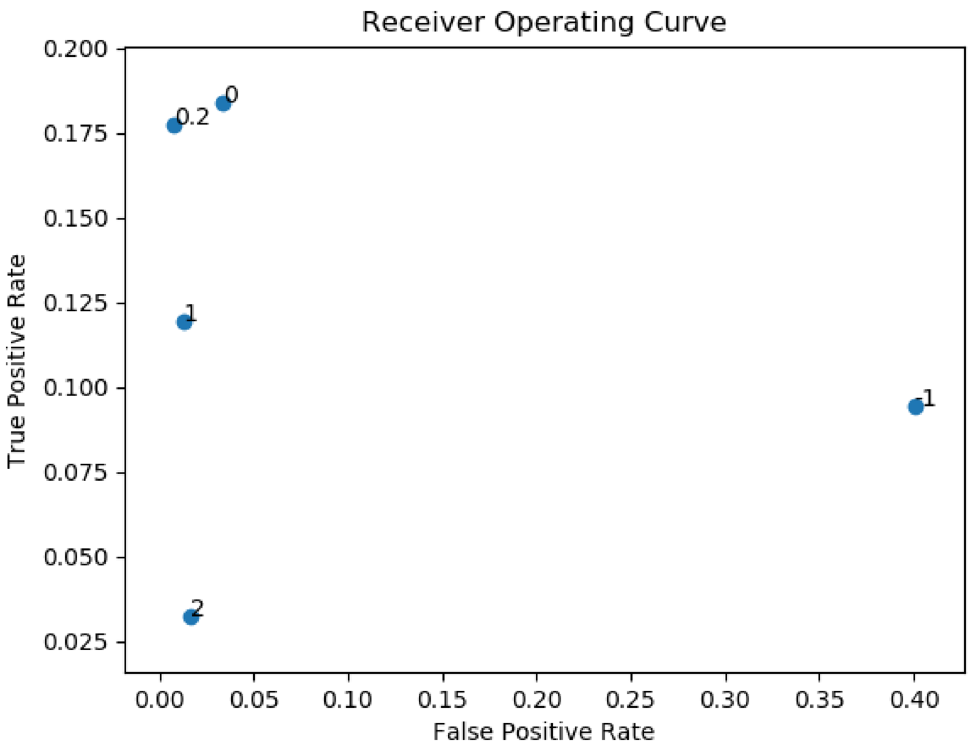

Figure 8: Receiver operating curve. The label for each data point is the value of θ_{ij} for the H_i, H_j terms.

7 Code

7.1 Helper functions

```python
import numpy as np
import gzip
import struct
import matplotlib.pyplot as plt
from mlxtend.data import loadlocal_mnist

def binarize(images):
    bi_images = np.zeros(images.shape)
    for k in range(images.shape[0]):
        for i in range(images.shape[1]):
            for j in range(images.shape[2]):
                if images[k,i,j] >= 0.5:
                    bi_images[k,i,j] = 1.0
                else:
                    bi_images[k,i,j] = -1.0
    return bi_images

def create_noisy(images):
    noisy_images = np.copy(images)
    size = images.shape[1] * images.shape[2]
    flip_size = (int)(size*0.02)
    for k in range(images.shape[0]):
        choice = np.random.permutation(np.arange(size))[:flip_size].tolist()
        for i in range(size):
            if i in choice:
                noisy_images[k, i//images.shape[1], i%images.shape[1]] \
                            = -images[k, i//images.shape[1], i%images.shape[1]]
    return noisy_images
```

```python
def denoise(noisy_images, theta_hh=0.2):
    # theta_hh = 0.2
    theta_hx = 0.2
    epsilon = 0.001
    num_epochs = 20
    # Initialize diff list
    diff = [[] for _ in range(noisy_images.shape[0])]
    for i in diff:
        i.append(0)
    length = noisy_images.shape[1]
    images = np.copy(noisy_images)
    for k in range(images.shape[0]):
        # Initialize edge weights pi
        pi = np.random.rand(length, length)
        prev_pi = np.copy(pi)
        for epoch in range(num_epochs):
            exponent = np.zeros((length, length))
            for i in range(images.shape[1]):
                for j in range(images.shape[2]):
                    if i is not 0: # Not on the top edge
                        exponent[i,j] += theta_hh*(2*pi[i-1,j]-1) + theta_hx*noisy_images[k,i-1,j]
                    if i is not images.shape[1]-1: # Not on the bottom edge
                        exponent[i,j] += theta_hh*(2*pi[i+1,j]-1) + theta_hx*noisy_images[k,i+1,j]
                    if j is not 0: # Not on the left edge
                        exponent[i,j] += theta_hh*(2*pi[i,j-1]-1) + theta_hx*noisy_images[k,i,j-1]
                    if j is not images.shape[1]-1: # Not on the right edge
                        exponent[i,j] += theta_hh*(2*pi[i,j+1]-1) + theta_hx*noisy_images[k,i,j+1]
                    # Update edge weights
                    pi[i,j] = np.exp(exponent[i,j]) / (np.exp(exponent[i,j]) + np.exp(-exponent[i,j]))
                    if pi[i,j] < 0.5:
                        images[k,i,j] = -1.0
                    else:
                        images[k,i,j] = 1.0

            # diff[k].append(np.linalg.norm(pi-prev_pi,2))
            diff[k].append(np.sum(np.power(pi-prev_pi,2)))
            prev_pi = np.copy(pi)
            if diff[k][-1] < epsilon:
                break
    return images
```

```python
def accuracy(binary_images,denoise_images):
    accuracy_list = np.zeros((binary_images.shape[0],1))
    for k in range(binary_images.shape[0]):
        n_incorrect = np.count_nonzero(binary_images[k,:,:]-denoise_images[k,:,:])
        accuracy_list[k] = 1 - (n_incorrect / (binary_images.shape[1]*binary_images.shape[2]))
    return accuracy_list

def confusion(binary_images, denoise_images):
    true_positive_list = np.zeros((binary_images.shape[0],1))
    false_positive_list = np.zeros((binary_images.shape[0],1))
    for k in range(binary_images.shape[0]):
        true_positive = 0
        false_positive = 0
        for i in range(binary_images.shape[1]):
            for j in range(binary_images.shape[2]):
                if denoise_images[k,i,j] == 1.0:
                    if binary_images[k,i,j] == 1.0:
                        true_positive += 1
                    else:
                        false_positive += 1
        true_positive_list[k] = true_positive / (binary_images.shape[1]**2)
        false_positive_list[k] = false_positive / (binary_images.shape[1]**2)
    return np.mean(true_positive_list), np.mean(false_positive_list)
```

7.2 Main function

```python
def main():
    # img_filename = 'train-images-idx3-ubyte.gz'
    # images = extract_images(img_filename, 500)
    images, labels = loadlocal_mnist(images_path='train-images-idx3-ubyte', labels_path='train-labels-idx1-ubyte')
    images = images.reshape(-1,28,28)[:500,:,:]
    labels = labels[:500]
    label_img_dict = {}
    for i in range(labels.shape[0]):
        if labels[i] not in label_img_dict.keys():
            label_img_dict[labels[i]] = images[i,:,:].reshape(-1,28,28)
        else:
            label_img_dict[labels[i]] = np.concatenate((label_img_dict[labels[i]], images[i,:,:].reshape(-1,28,28)), axis=0)

    # Sample images for each digit
    plt.figure(figsize=(4.5,15))
    for num in range(10):
        orig_img = label_img_dict[num]
        binary_img = binarize(orig_img)
        plt.subplot(10,3,num*3+1)
        plt.imshow(binary_img[0,:,:], cmap='gray')
        noisy_img = create_noisy(binary_img)
        plt.subplot(10,3,num*3+2)
        plt.imshow(noisy_img[0,:,:], cmap='gray')
        denoise_img = denoise(noisy_img)
        plt.subplot(10,3,num*3+3)
        plt.imshow(denoise_img[0,:,:], cmap='gray')
    plt.savefig('samples.png')

    binary_images = binarize(images)
    noisy_images = create_noisy(binary_images)
    denoise_images = denoise(noisy_images)
    accuracy_list = accuracy(binary_images, denoise_images)
    avg_accuracy = sum(accuracy_list[:500]) / 500
    print('Average accuracy on the first 500 images: {}'.format(avg_accuracy))
```

```python
    # Most accurate
    max_idx = np.argmax(accuracy_list)
    plt.figure()
    plt.imshow(binary_images[max_idx,:,:])
    plt.title('Most accurate binary image')
    plt.savefig('most_accurate_binary_image.png')
    plt.figure()
    plt.imshow(noisy_images[max_idx,:,:])
    plt.title('Most accurate noisy image')
    plt.savefig('most_accurate_noisy_image.png')
    plt.figure()
    plt.imshow(denoise_images[max_idx,:,:])
    plt.title('Most accurate denoised image')
    plt.savefig('most_accurate_denoised_image.png')

    # Least accurate
    min_idx = np.argmin(accuracy_list)
    plt.figure()
    plt.imshow(binary_images[min_idx,:,:])
    plt.title('Least accurate binary image')
    plt.savefig('least_accurate_binary_image.png')
    plt.figure()
    plt.imshow(noisy_images[min_idx,:,:])
    plt.title('Least accurate noisy image')
    plt.savefig('least_accurate_noisy_image.png')
    plt.figure()
    plt.imshow(denoise_images[min_idx,:,:])
    plt.title('Least accurate denoised image')
    plt.savefig('least_accurate_denoised_image.png')
```

```python
    # ROC
    denoise_images_list = []
    accuracies = []
    true_positive_list = []
    false_positive_list = []
    for theta_hh in [-1,0,0.2,1,2]:
        denoise_images_list.append(denoise(noisy_images, theta_hh))
        accuracies.append(accuracy(binary_images, denoise_images))
        true_positive, false_positive = confusion(binary_images, denoise_images_list[-1])
        true_positive_list.append(true_positive)
        false_positive_list.append(false_positive)

    txt_list = [-1,0,0.2,1,2]
    fig, ax = plt.subplots()
    ax.scatter(false_positive_list, true_positive_list)
    for i, txt in enumerate(txt_list):
        ax.annotate(txt, (false_positive_list[i], true_positive_list[i]))
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    plt.title('Receiver Operating Curve')
    plt.savefig('roc.png')
```