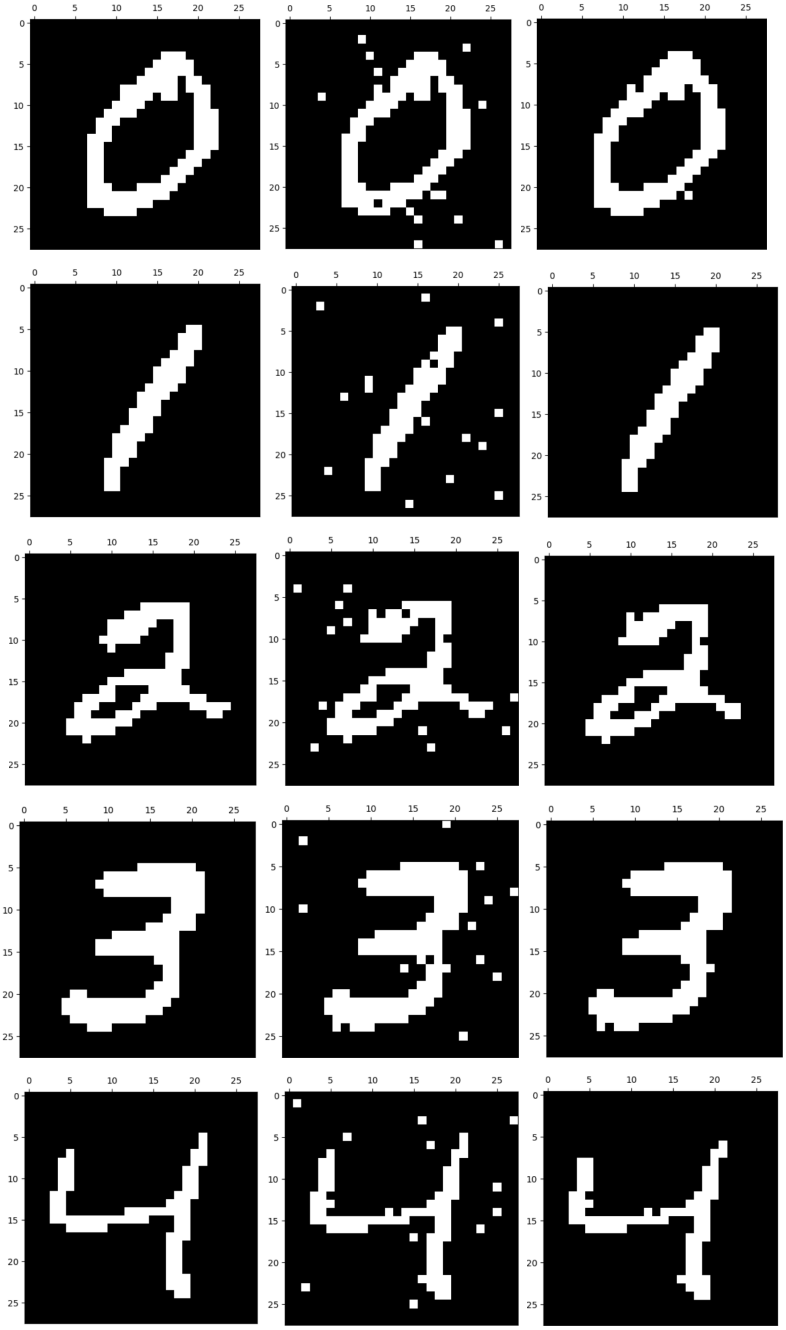
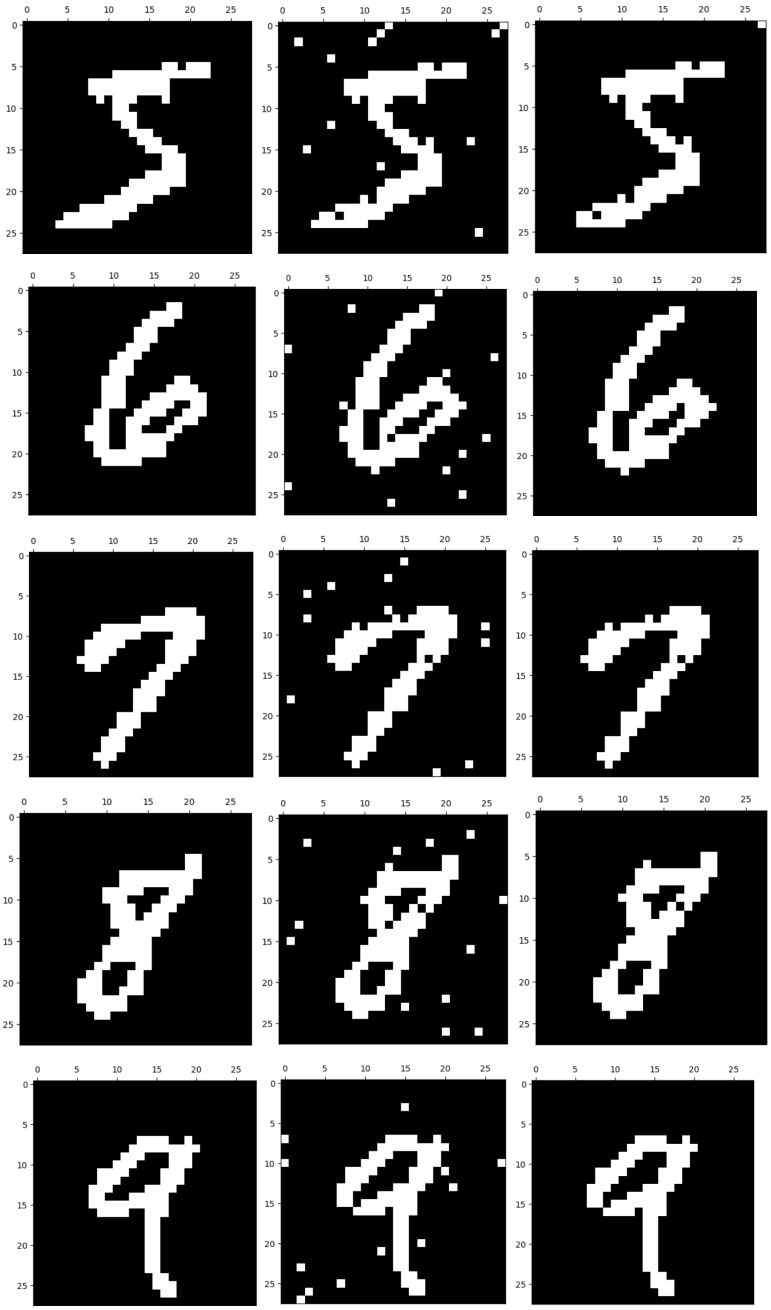
CS498 AML HW9

Yidi Yang (yyang160)
Huiyun Wu (hwu63)

1. Average accuracy on the first 500 images: 0.9946836734693877

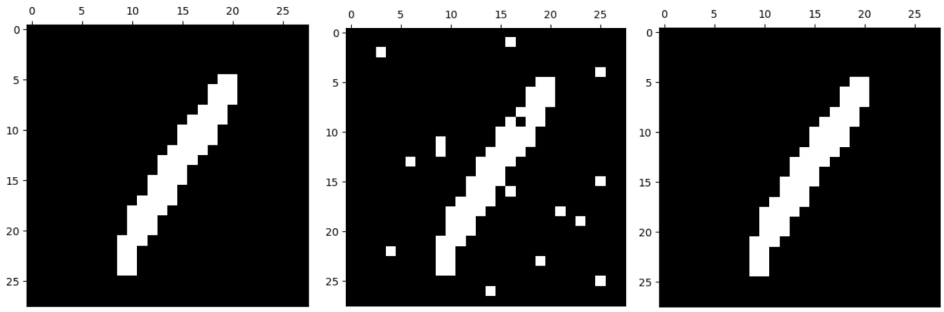
2. One set of sample images for each digit -- For each digit you should put up a row of a (sample image, noised version, denoised version via MFI). This means should have a total of 30 images (10 rows, 3 columns).





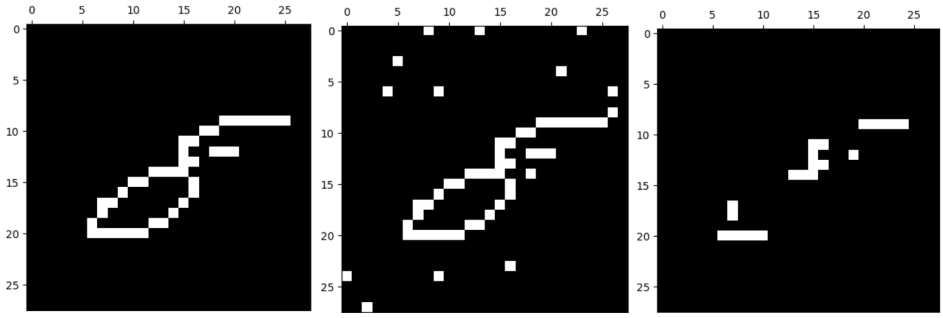
3. Best reconstruction (original, noisy, denoised)

a. Original image b.Noisy image c. Reconstruction

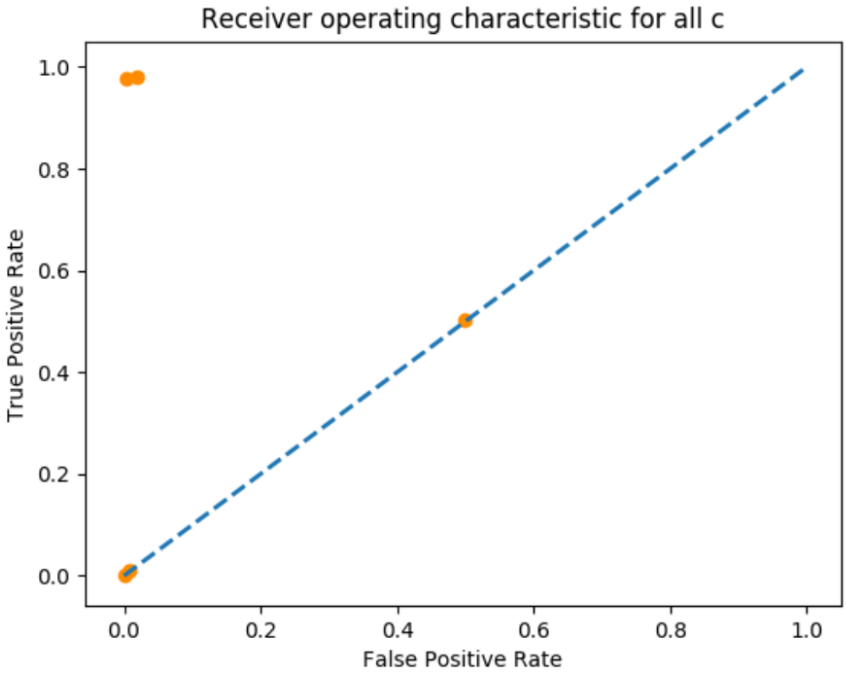


4. Worst reconstruction (original, noisy, denoised)

a. Original image b.Noisy image c. Reconstruction



5. ROC curve



6. Code snippet

Convert images back to 2D arrays and the Boltzmann model:

```
def to_2d(images):
    res_images = np.ndarray((500, 28, 28))
    for i in range(500):
        res_images[i] = np.reshape(images[i], (28, 28))
    return res_images

def boltzmann(image, theta_ij):
    pi = np.asarray([[0.5]*28]*28)
    prev_pi = np.asarray([[1]*28]*28)
    # theta H,H = 0.2, theta H,X = 2
    while True:
        for i in range(28):
            for j in range(28):
                nom = 0
                if i >= 1:
                    nom += theta_ij * (2 * pi[i - 1, j] - 1)
                if i <= 26:
                    nom += theta_ij * (2 * pi[i + 1, j] - 1)
                if j >= 1:
                    nom += theta_ij * (2 * pi[i, j - 1] - 1)
                if j <= 26:
                    nom += theta_ij * (2 * pi[i, j + 1] - 1)
                nom += 0.2 * image[i, j]
                pi[i, j] = np.exp(nom)/(np.exp(nom) + np.exp(-nom))
        error = np.abs(np.sum((prev_pi - pi)))
        if error < 0.01:
            break
        prev_pi = np.copy(pi)
    out = np.asarray([[(pi[i, j] > 0.5).astype(int) - (pi[i, j] <= 0.5).astype(int)
                       for j in range(28)] for i in range(28)])
    return out
```

Step-by-Step Driver Code:

```
if __name__ == "__main__":
    training_data = list(read(dataset='training', path='.'))

    # Get first 500 images
    training_data = training_data[:500]
    print(len(training_data))

    # Normalize
    images = np.ndarray((500, 784))
    noisy_images = np.ndarray((500, 784))
    for i in range(500):
        image = np.reshape(training_data[i][1].astype(float), (1,784))
        image[0] /= 256

        # Binarize
        for j in range(784):
            image[0, j] = (image[0, j] > 0.5).astype(int) - (image[0, j] <= 0.5).astype(int)
        images[i] = image[0]

        # Flip 2% bits
        flip = np.random.randint(0, 783, int(0.02 * 784))
        noisy_images[i] = images[i].copy()
        for idx in flip:
            noisy_images[i][idx] = -images[i][idx]

    # To 2-D image
    noisy_images = to_2d(noisy_images)
    images = to_2d(images)

    # Denoise using boltzmann model
    fpr = np.zeros(5)
    tpr = np.zeros(5)
    k = 0
    for c in [-1, 0, 0.2, 1, 2]:
    #for c in [0.2]:
        denoised_images = np.zeros((500, 28, 28))
        for i in range(500):
            denoised_images[i] = boltzmann(noisy_images[i], c)
```

Produce Images.

```
    # Fraction of correctly denoised
    accuracy = np.zeros(500)
    for i in range(500):
        accuracy[i] = np.sum(denoised_images[i] == images[i])/784
    fraction = np.sum(accuracy)/500
    print("Overfall fraction of correction: ", fraction)

    # Max accuracy
    max_acc = np.max(accuracy)
    max_index = np.argmax(accuracy)
    print("Max accuracy: ", max_acc)
    #
    # # original image
    show(images[max_index])
    show(noisy_images[max_index])
    show(denoised_images[max_index])
    #
    # # Min accuracy
    min_acc = np.min(accuracy)
    min_index = np.argmin(accuracy)
    print("Min accuracy: ", min_acc)
    #
    # # original image
    show(images[min_index])
    show(noisy_images[min_index])
    show(denoised_images[min_index])

    labels = [1,2,3,4,5,6, 7, 8, 9, 0]
    ind = []
    while len(labels) > 0:
        for i, item in enumerate(training_data):
            label, image = item
            if label in labels:
                labels.remove(label)
                ind.append(i)
                continue
    for i in ind:
        show(images[i])
        show(noisy_images[i])
        show(denoised_images[i])
```

Produce ROC curve

```
        fp = 0
        tp = 0
        N = 0
        for i in range(500):
            accuracy[i] = np.sum(denoised_images[i] == images[i])/784
            for j in range(28):
                for s in range(28):
                    if images[i, j, s] == -1 and denoised_images[i, j, s] == 1:
                        fp += 1
                    if images[i, j, s] == 1 and denoised_images[i, j, s] == 1:
                        tp += 1
                    if images[i, j, s] == -1:
                        N += 1
```

```
        print(fp/N)
        fpr[k] = fp/N
        tpr[k] = tp/(500*784-N)
        k += 1
    print(fpr, tpr)
    plt.figure()
    lw = 2
    plt.plot([0, 1], [0, 1], lw=lw, linestyle='--')
    plt.scatter(fpr, tpr, color='darkorange')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic for all c')
    plt.show()
```

7. Any other relevant code

We modified a piece of code from GitHub (https://gist.github.com/akesling/5358964) to read in the MNIST data.

```
def read(dataset = "training", path = "."):
    """
    Python function for importing the MNIST data set.  It returns an iterator
    of 2-tuples with the first element being the label and the second element
    being a numpy.uint8 2D array of pixel data for the given image.
    """

    if dataset is "training":
        fname_img = os.path.join(path, 'train-images-idx3-ubyte')
        fname_lbl = os.path.join(path, 'train-labels-idx1-ubyte')
    elif dataset is "testing":
        fname_img = os.path.join(path, 't10k-images-idx3-ubyte')
        fname_lbl = os.path.join(path, 't10k-labels-idx1-ubyte')
    else:
        raise ValueError("dataset must be 'testing' or 'training'")

        # Load everything in some numpy arrays
    with open(fname_lbl, 'rb') as flbl:
        magic, num = struct.unpack(">II", flbl.read(8))
        lbl = np.fromfile(flbl, dtype=np.int8)

    with open(fname_img, 'rb') as fimg:
        magic, num, rows, cols = struct.unpack(">IIII", fimg.read(16))
        img = np.fromfile(fimg, dtype=np.uint8).reshape(len(lbl), rows, cols)

    get_img = lambda idx: (lbl[idx], img[idx])

        # Create an iterator which returns each image in turn
    for i in range(len(lbl)):
        yield get_img(i)
```

```
def show(image):
    """
    Render a given numpy.uint8 2D array of pixel data.
    """
    from matplotlib import pyplot
    import matplotlib as mpl

    fig = pyplot.figure()
    ax = fig.add_subplot(1, 1, 1)
    imgplot = ax.imshow(image, cmap=plt.get_cmap('gray'))
    imgplot.set_interpolation('nearest')
    ax.xaxis.set_ticks_position('top')
    ax.yaxis.set_ticks_position('left')
    pyplot.show()
```