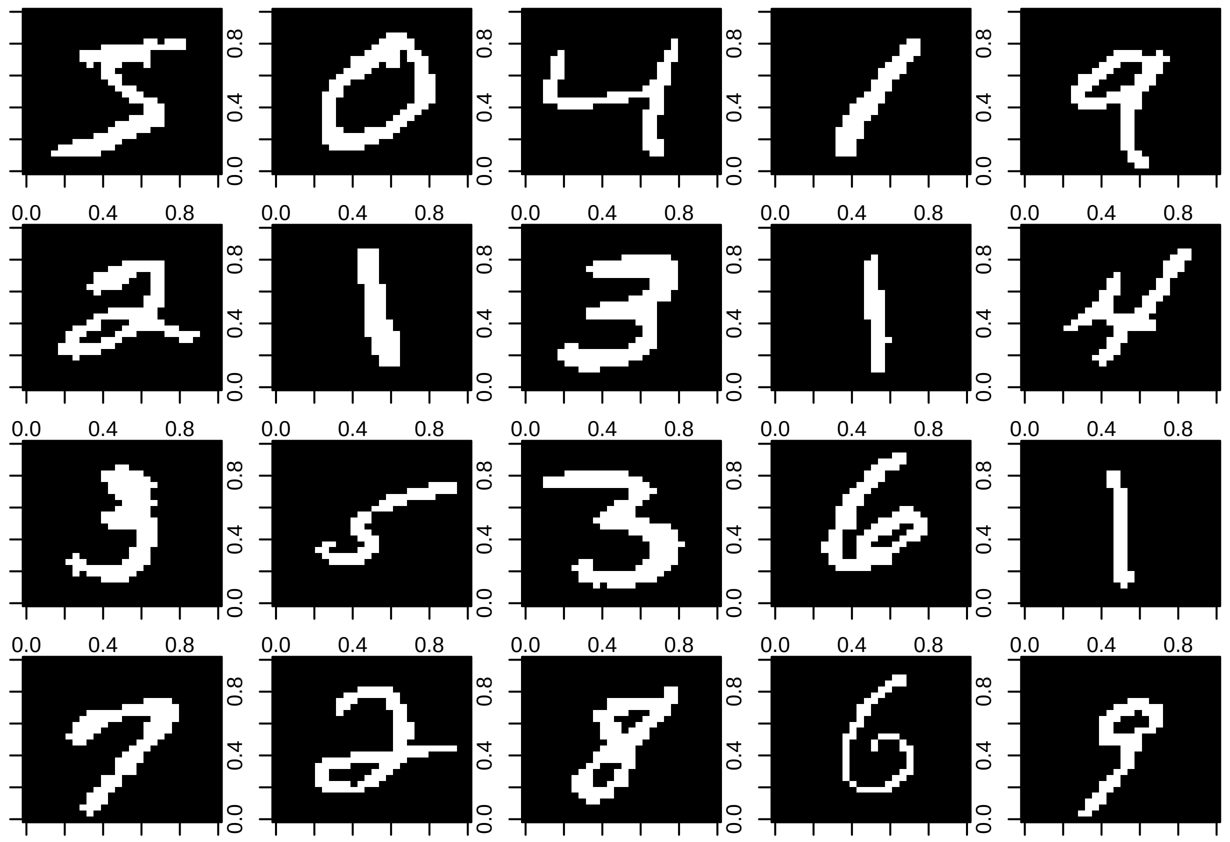
Homework 8

CS 498, Spring 2018, Xiaoming Ji

Obtaining the dataset

```
IMAGE_COUNT = 20

load_image_data = function(file.name, img.count){
  img_file = file(file.name, "rb")
  header = readBin(img_file, integer(), n = 4,  endian = "big")

  if(img.count == -1) img.count = header[2]

  data = matrix(0, img.count, 28 * 28)
  img = readBin(img_file, integer(), size = 1, n = img.count* 28*28, endian = "big",
                    signed = FALSE)

  data = matrix(img, ncol=28 * 28, byrow=T)
  close(img_file)

  return (data)
}

load_label_data = function(file.name){
  label_file = file(file.name, "rb")
  header = readBin(label_file, integer(), n = 2,  endian = "big")

  data = readBin(label_file, integer(), size = 1, n = header[2], endian = "big",
                   signed = FALSE)
  close(label_file)

  return (data)
}

train_data = load_image_data("./MNIST/train-images-idx3-ubyte", IMAGE_COUNT)

#Binarize data
train_data[(train_data / 255) < 0.5] = -1
train_data[train_data != -1] = 1
```

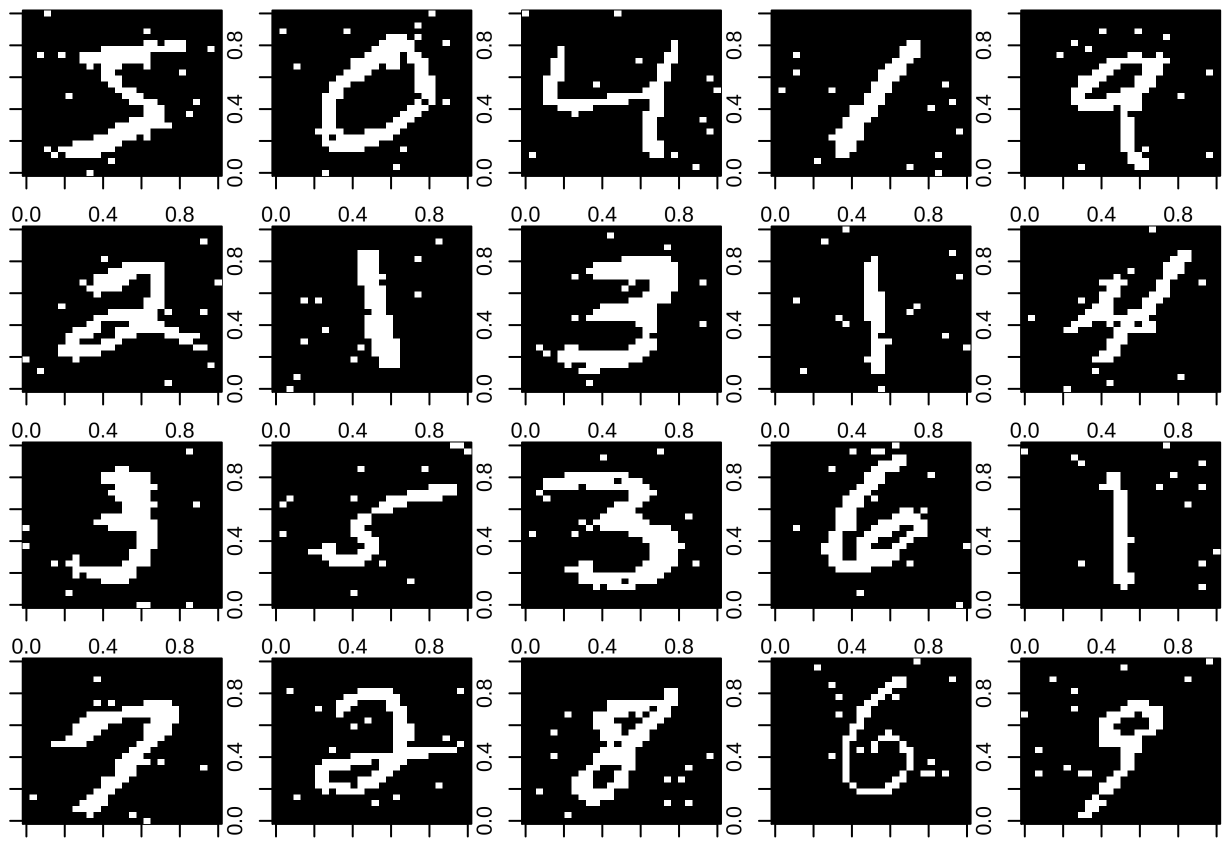
```
par(mfrow=c(4,5))
par(mar = c(1,1,1,1))

for(i in 1:20)
{
  image(matrix(train_data[i,], 28, 28)[, 28:1], col  = gray(0:1))
}
```



Adding pre-determined noise to the dataset

```r
library(readr)

read.coordinates = function(file.name) {
  RawData = read_csv(file.name)
  coor_count = dim(RawData)[1] / 2 * (dim(RawData)[2] - 1)
  coordinates = matrix(0, coor_count, 3)
  col_count = dim(RawData)[2] - 1

  n_i = 1
  for(r_i in 1:(dim(RawData)[1] / 2)) {
    i = 2 * r_i - 1
    m = regexec("[[:digit:]]+", as.character(RawData[i, 1]))
    img_num = as.integer(regmatches(RawData[i, 1], m)) + 1

    for (k in 1:col_count) {
      coordinates[n_i, 1] = img_num
      coordinates[n_i, 2] = as.integer(RawData[i, k + 1]) + 1
      coordinates[n_i, 3] = as.integer(RawData[i + 1, k + 1]) + 1

      n_i = n_i + 1
    }
  }
  coordinates
}
```

```
NoiseCoordinates = read.coordinates("./SupplementaryAndSampleData/NoiseCoordinates.csv")

noise_train_data = train_data
for (i in 1:dim(NoiseCoordinates)[1]){
  img_i = NoiseCoordinates[i, 1]
  bit_i = (NoiseCoordinates[i, 2] - 1) * 28 + NoiseCoordinates[i, 3]

  if(noise_train_data[img_i, bit_i] > 0) noise_train_data[img_i, bit_i] = -1
  else noise_train_data[img_i, bit_i] = 1
}

par(mfrow=c(4,5))
par(mar = c(1,1,1,1))

for(i in 1:20)
{
  image(matrix(noise_train_data[i,] ,28, 28)[, 28:1], col  = gray(0:1))
}
```

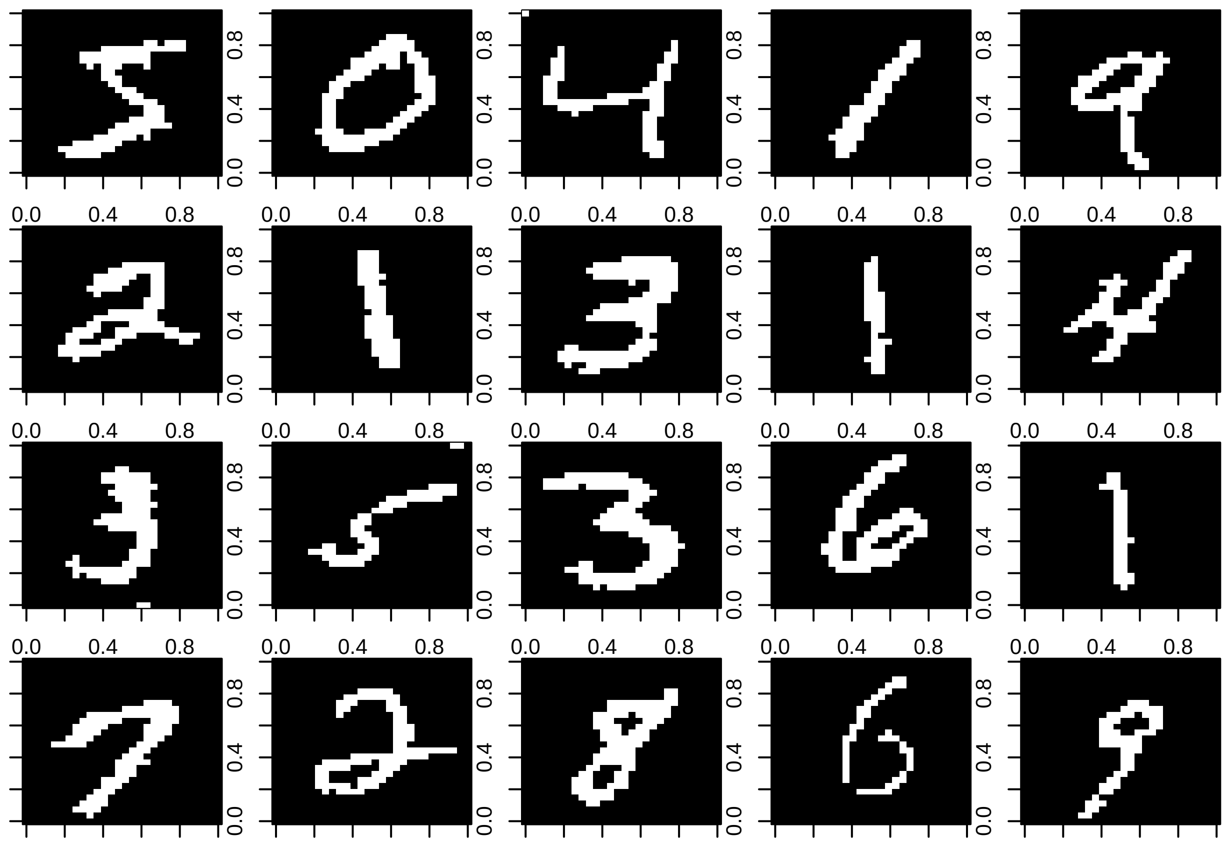


Building a Boltzman Machine

```
UpdateOrders = read.coordinates("./SupplementaryAndSampleData/UpdateOrderCoordinates.csv")
InitialQ = read.csv("./SupplementaryAndSampleData/InitialParametersModel.csv", header=FALSE)
```

```
MFI = function(init.Q, update.orders, theta.hh.ij, theta.hx.ij, img.data, iterations){
  All_Q = list()
  Q = init.Q
  All_Q[[1]] = Q

  for (iter in 1:iterations){
    for (c in 1:dim(update.orders)[1]) {
      i = update.orders[c, 1]
      j = update.orders[c, 2]
      log_q_ij_1 = 0

      neighbors = matrix(c(i, j-1,i,j+1,i-1,j,i+1,j), 4, 2, byrow = TRUE)
      for(n in 1:dim(neighbors)[1]){
        n_i = neighbors[n, 1]
        n_j = neighbors[n, 2]

        if(n_i > 0 && n_i <= 28 && n_j > 0 && n_j <= 28)
          log_q_ij_1 = log_q_ij_1 + theta.hh.ij * (2 * Q[n_i, n_j] - 1)
      }


      log_q_ij_1 = log_q_ij_1 + theta.hx.ij * img.data[i, j]

      Q[i, j] = exp(log_q_ij_1) / (exp(log_q_ij_1) + exp(-log_q_ij_1))
    }

    All_Q[[iter + 1]] = Q
  }

  img = as.matrix(Q)
  img[img < 0.5] = -1
  img[img != -1] = 1

  return (list(img=img, All_Q = All_Q))
}
```

```
par(mfrow=c(4,5))
par(mar = c(1,1,1,1))

for (i in 1:IMAGE_COUNT){
  img_data = matrix(noise_train_data[i,], 28, 28, byrow = TRUE)
  update_orders = UpdateOrders[UpdateOrders[,1] == i,2:3]

  denoised_img_data = MFI(InitialQ, update_orders, 0.8, 2, img_data, 10)$img

  image(t(denoised_img_data[28:1,]), col  = gray(0:1))
}
```



Turning in the energy function values computed initially and after each iteration

```r
EQ = function(init.Q, theta.hh.ij, theta.hx.ij, img.data){
  Q = init.Q
  epsilon = 1e-10

  eq_log_q = sum((Q * log(Q + epsilon)) + ((1 - Q) * log(1 - Q + epsilon)))

  eq_log_p = 0
  for (i in 1:dim(img.data)[1]){
    for (j in 1:dim(img.data)[2]){
      neighbors = matrix(c(i, j-1,i,j+1,i-1,j,i+1,j), 4, 2, byrow = TRUE)

      for(n in 1:dim(neighbors)[1]){
        n_i = neighbors[n, 1]
        n_j = neighbors[n, 2]

        if(n_i > 0 && n_i <= 28 && n_j > 0 && n_j <= 28)
          eq_log_p = eq_log_p + theta.hh.ij * (2 * Q[i, j] - 1) * (2 * Q[n_i, n_j] - 1)
      }

      eq_log_p = eq_log_p + theta.hx.ij * (2 * Q[i, j] - 1) * img.data[i, j]
    }
  }
  return (eq_log_q - eq_log_p)
}
```

```r
image_eq = matrix(0, 20, 11)
image = matrix(0, 28, 28 * 20)

for (i in 1:20){
  img_data = matrix(noise_train_data[i,], 28, 28, byrow = TRUE)
  update_orders = UpdateOrders[UpdateOrders[,1] == i,2:3]

  r = MFI(InitialQ, update_orders, 0.8, 2, img_data, 10)
  All_Q = r$All_Q
  image[1:28, (28 * i - 27):(28 * i)] = r$img

  for (j in 1:11){
    image_eq[i, j] = EQ(All_Q[[j]], 0.8, 2, img_data)
  }
}

image[image < 0] = 0

write.table(image_eq, file="eq.csv", row.names = FALSE, col.names = FALSE, sep = ",")
write.table(image[1:28, 281:560], file="denoised_11_20.csv",
            row.names = FALSE, col.names = FALSE, sep = ",")
```

Displaying the reconstructed images

```r
original_image = matrix(0, 28, 28 *20)
noise_image = matrix(0, 28, 28 *20)

for (i in 1:20) {
  img_data = matrix(train_data[i,], 28, 28, byrow = TRUE)
  original_image[1:28, (28 * i - 27):(28 * i)] = img_data

  img_data = matrix(noise_train_data[i,], 28, 28, byrow = TRUE)
  noise_image[1:28, (28 * i - 27):(28 * i)] = img_data

}

par(mfrow=c(3,1))
par(mar = c(0,0,0,0))
image(t(original_image[1:28, 281:560][28:1,]), col  = gray(0:1))
image(t(noise_image[1:28, 281:560][28:1,]), col  = gray(0:1))
image(t(image[1:28, 281:560][28:1,]), col  = gray(0:1))
```