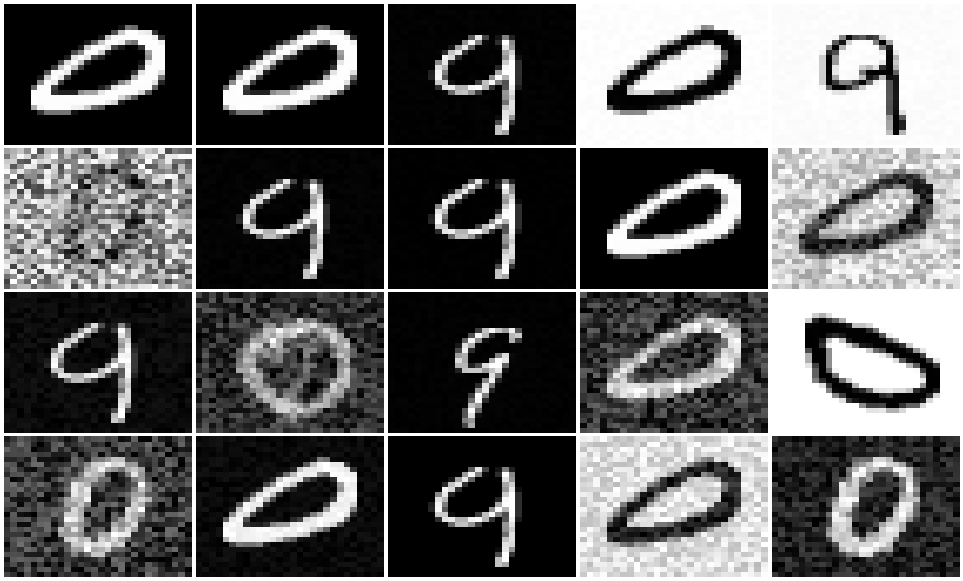
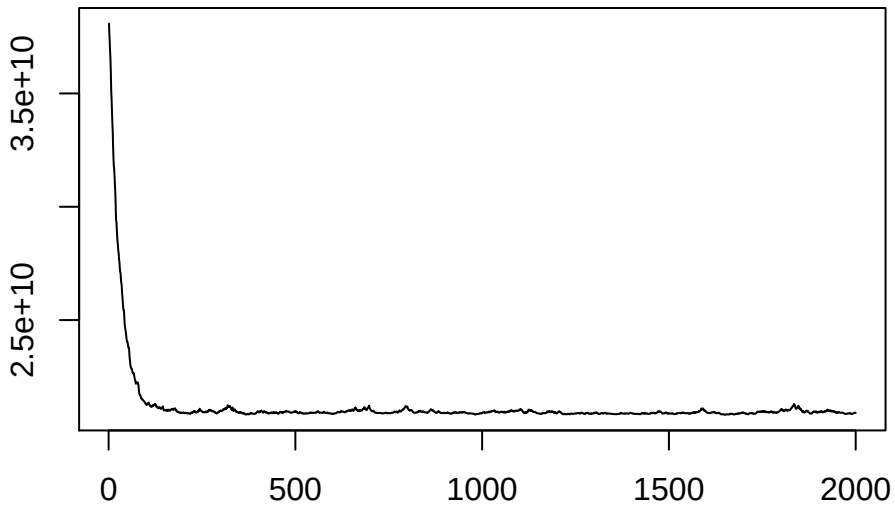
STAT 9100 HOMEWORK 5

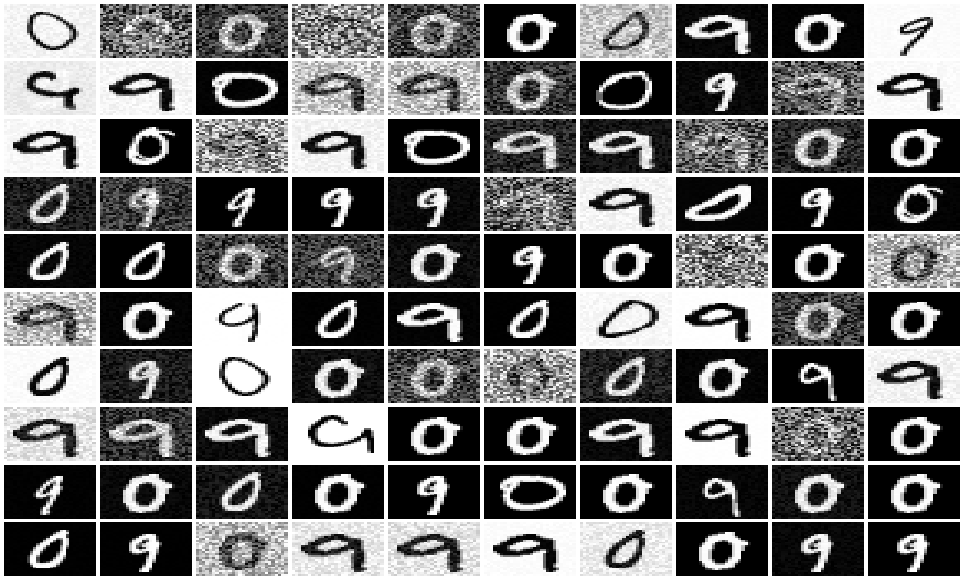
Peng Shao

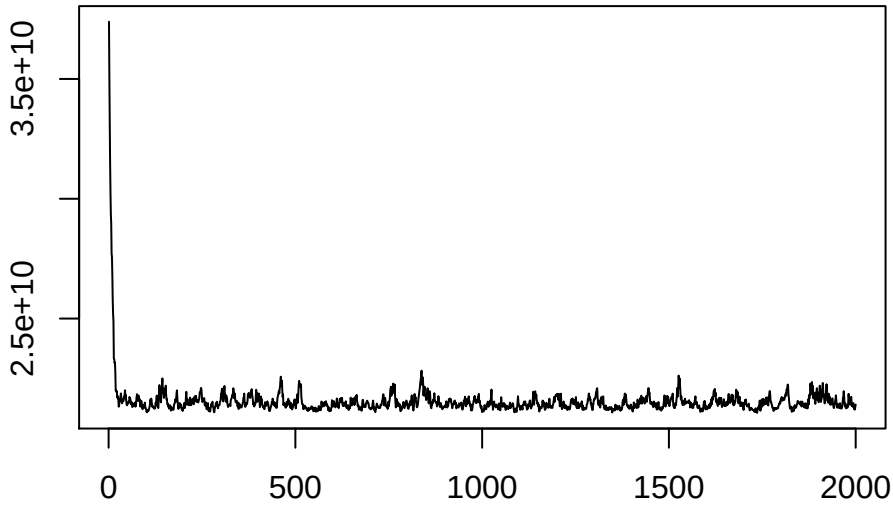
March 24, 2016

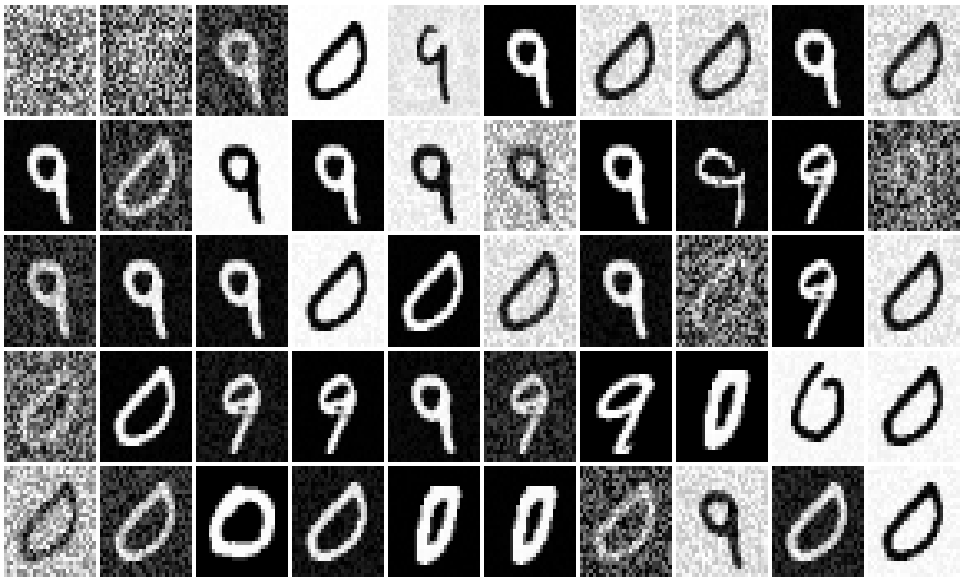
(All graphs are listed as the order mentioned in the text.) ## a. I tried 20, 50, 100 hidden units respectively, the final performances does not show too much difference, but the less hidden units there are, the more stable the iteration becomes. All the results show that part of hidden units can recognize "0", part of them can recognize "9" and the rest can hardly know anything.

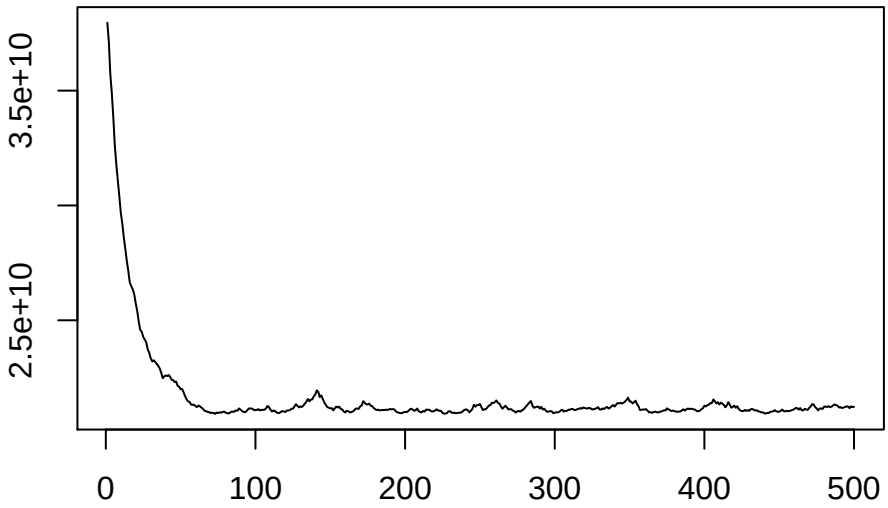






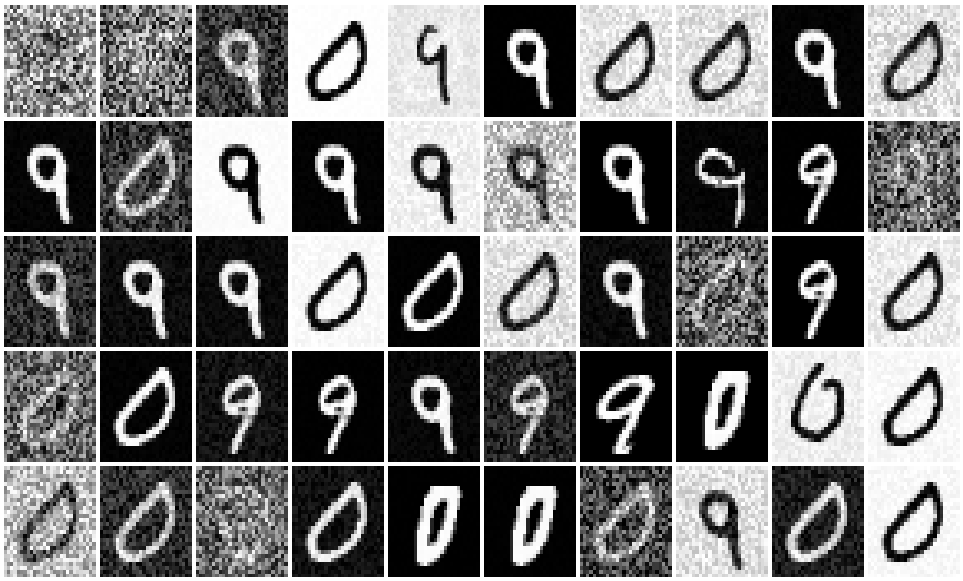

3.5e+10
2.5e+10
0
500
1000
1500
2000

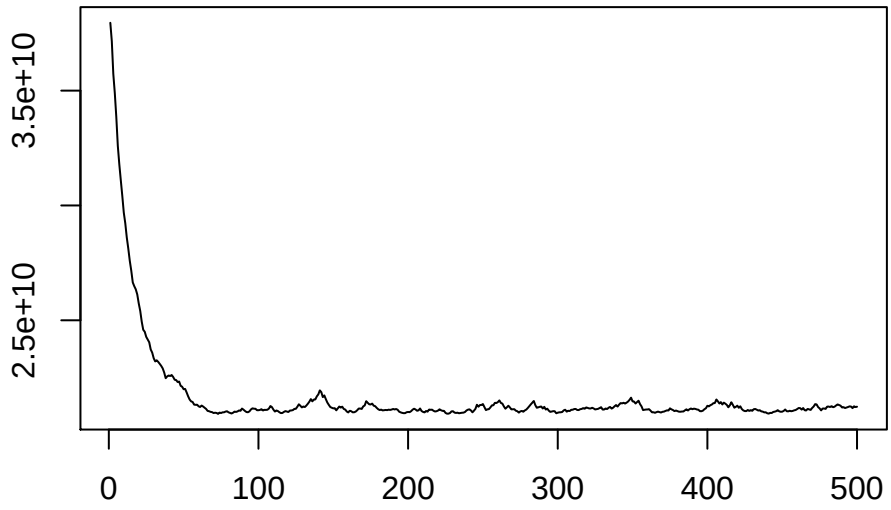


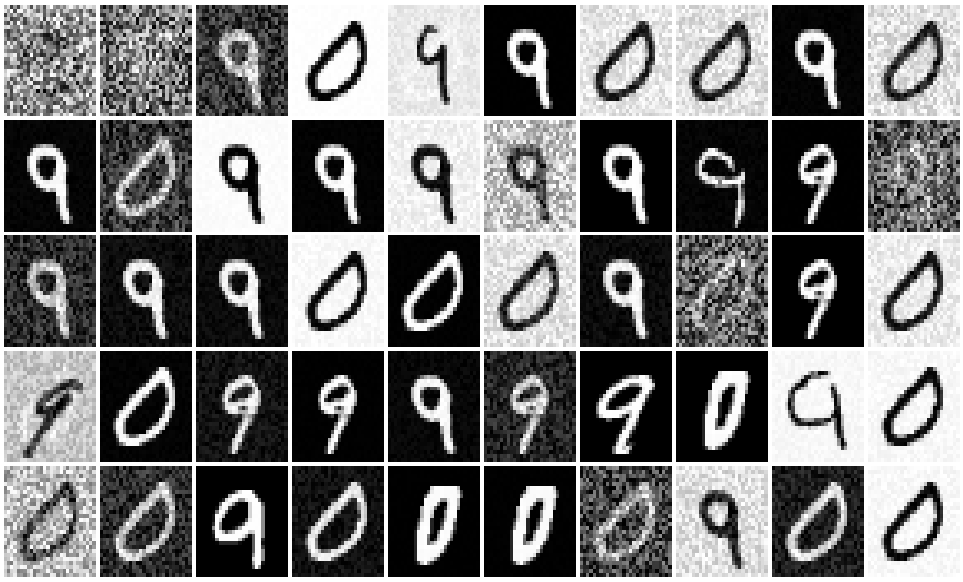


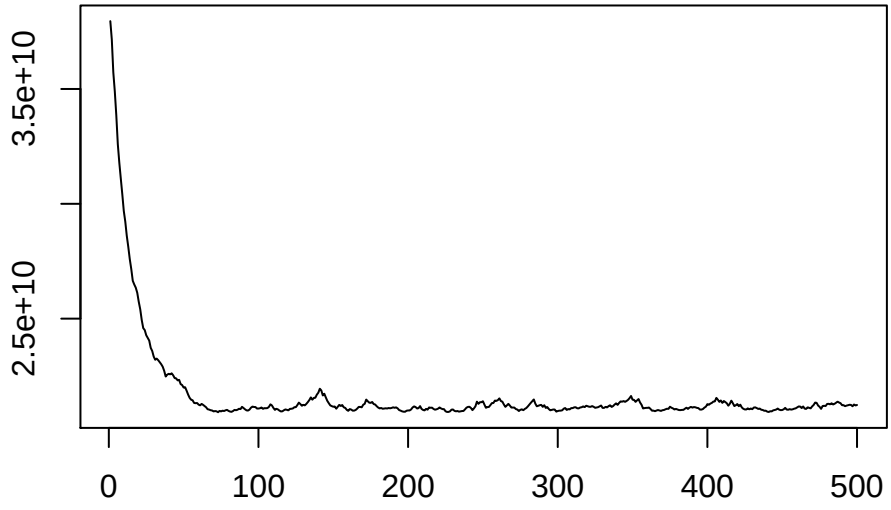
b.

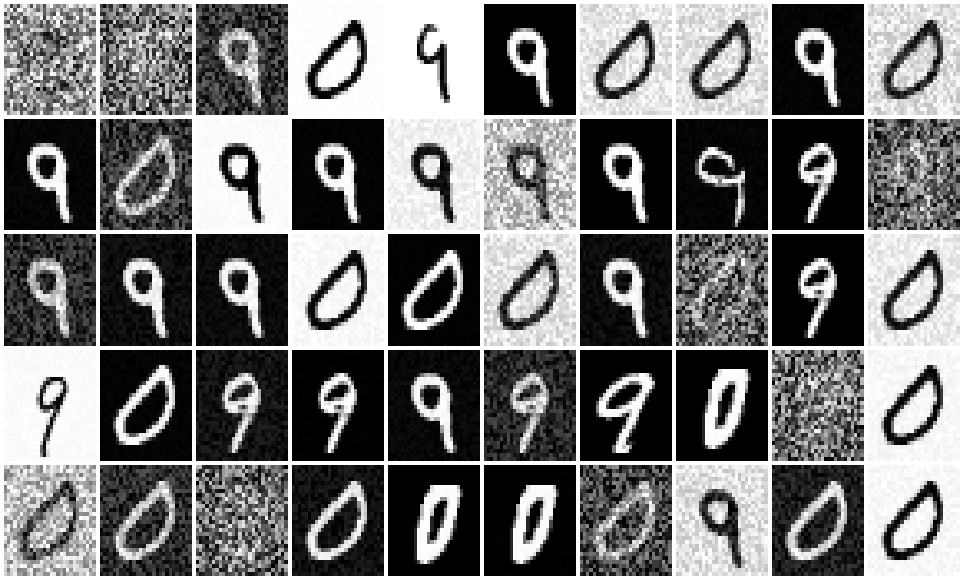
The λs I choose are: 0.001, 0.01, 0.1, 1, 10. The first three – 0.001, 0.01, 0.1 – do not show to much effect on the result. When $\lambda = 1$, there are more hidden units can recognize "0". When $\lambda = 10$, almost all hidden units try to recognize "0".

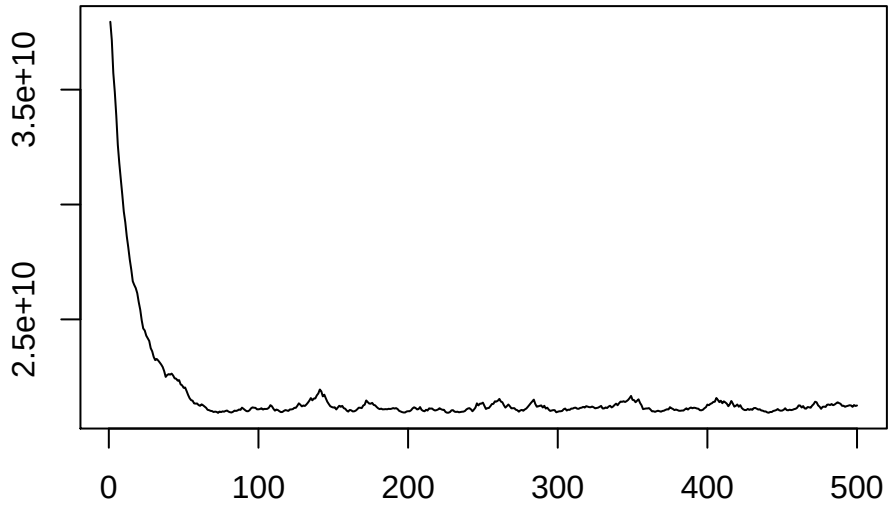


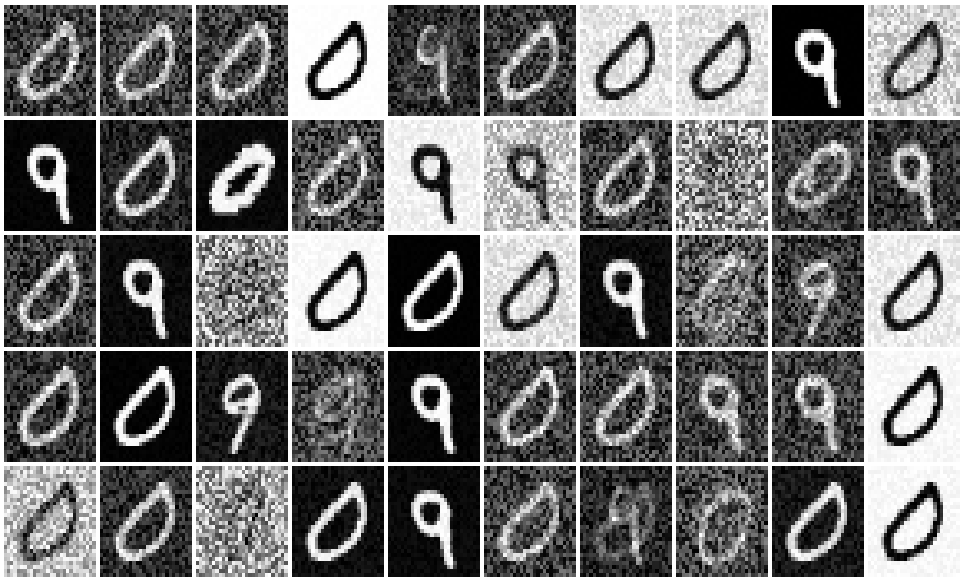

3.5e+10
2.5e+10
0
100
200
300
400
500

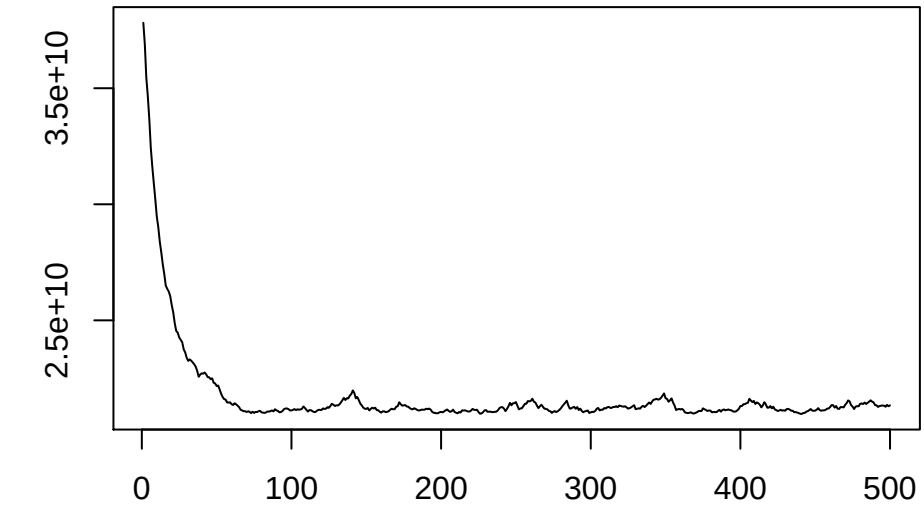


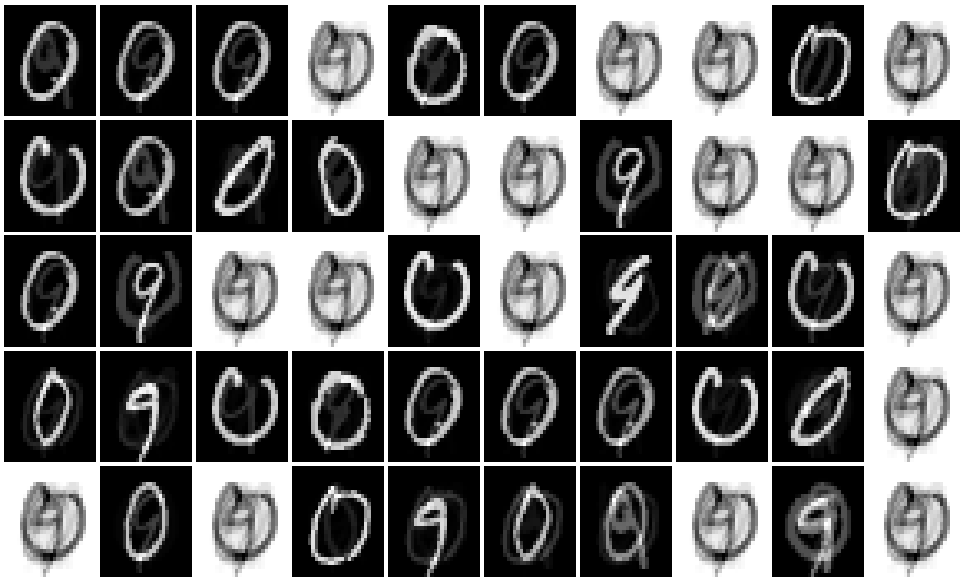

3.5e+10
2.5e+10
0
100
200
300
400
500

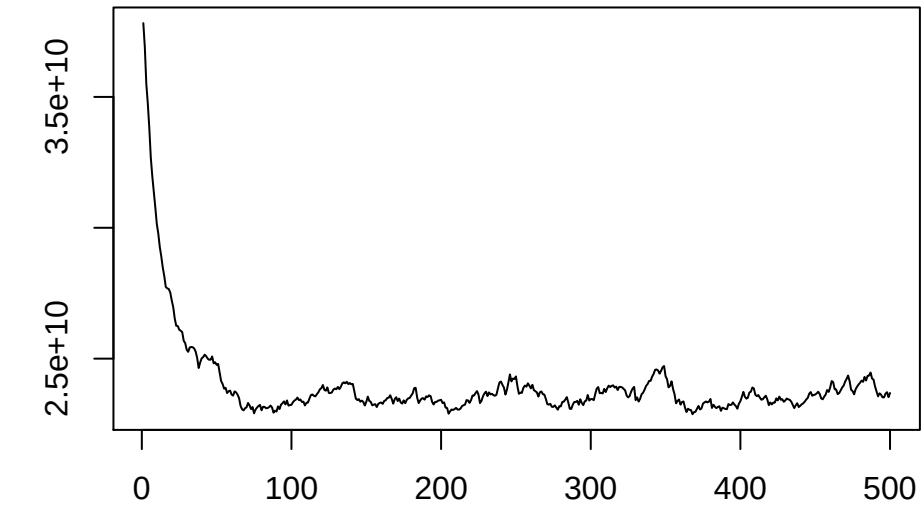



3.5e+10
2.5e+10
0
100
200
300
400
500



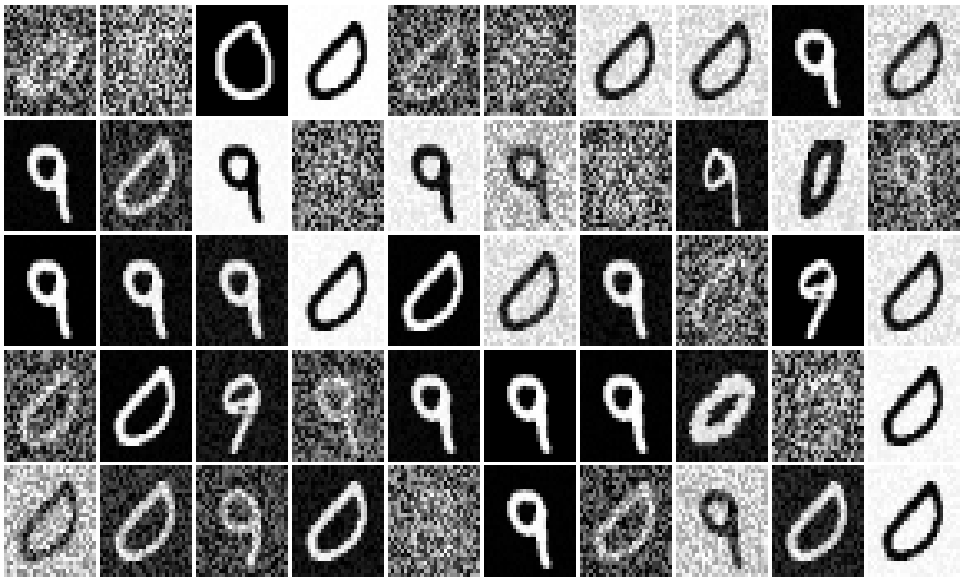

3.5e+10
2.5e+10
0
100
200
300
400
500

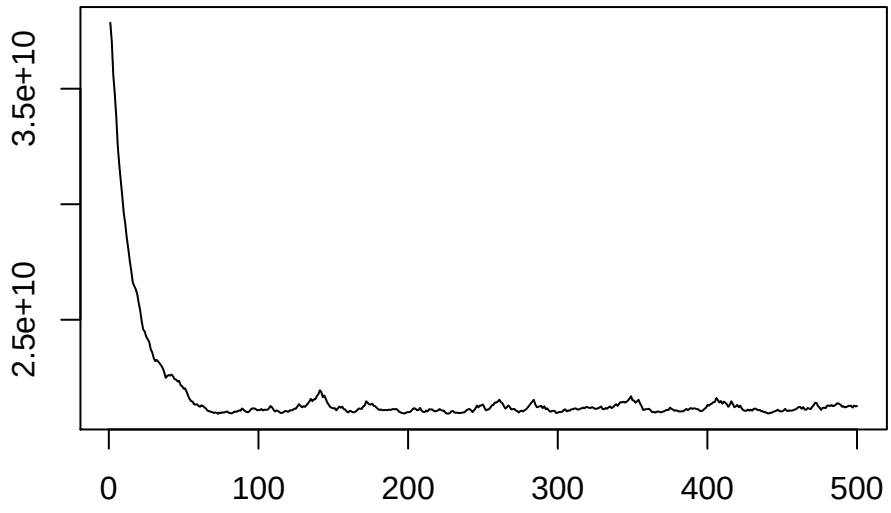


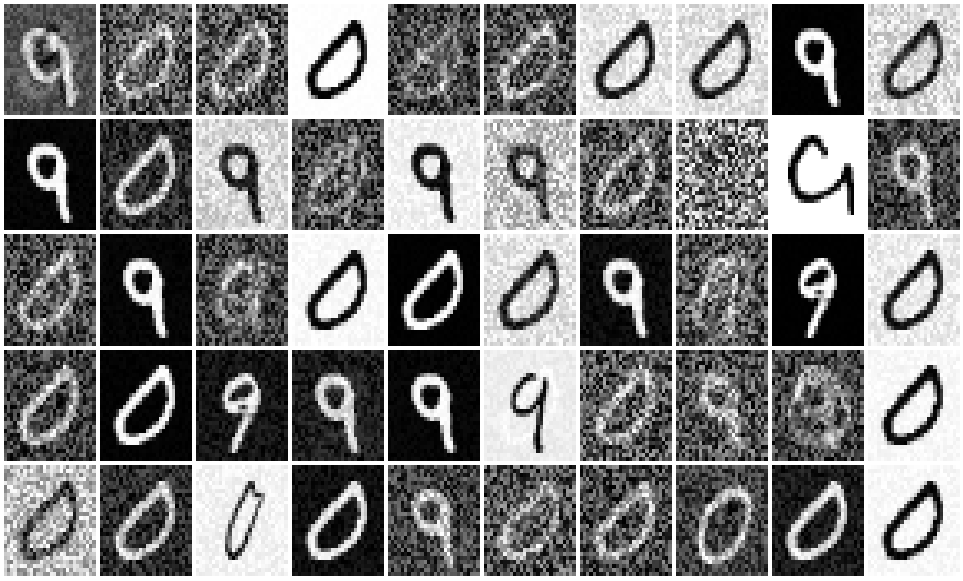

3.5e+10
2.5e+10
0
100
200
300
400
500

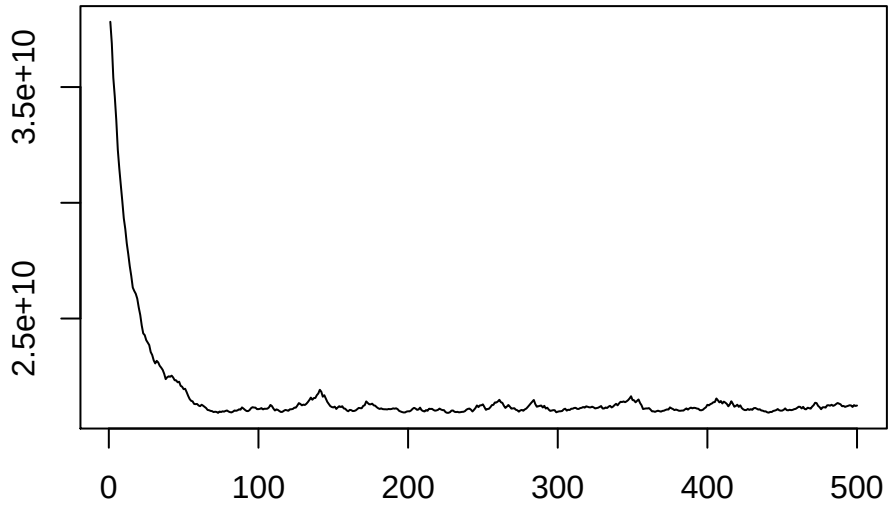
c.

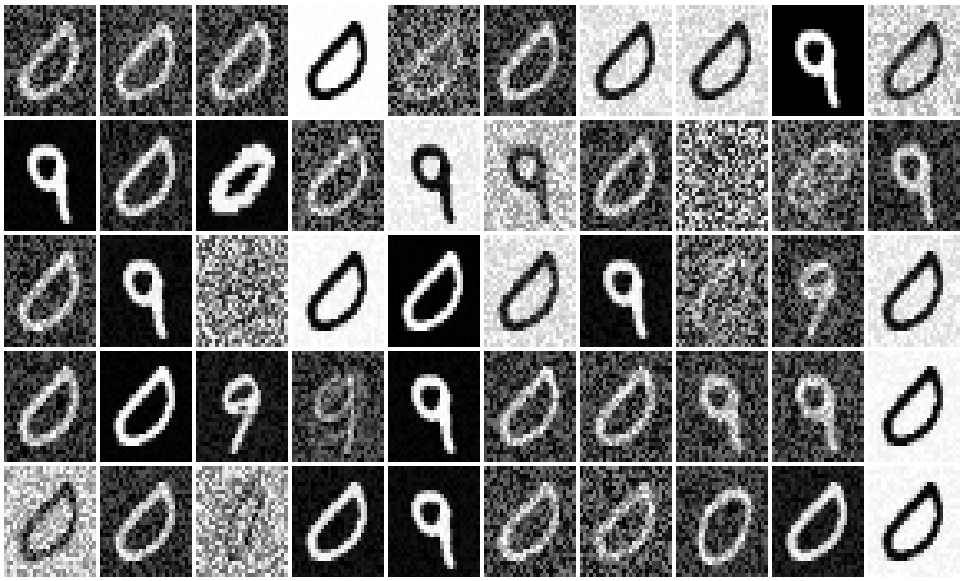
The β I choose are:0.01, 0.05, 0.1. It seems like when $\beta = 0.05$ the autoencoder performs best since there are least hidden units which "know nothing".

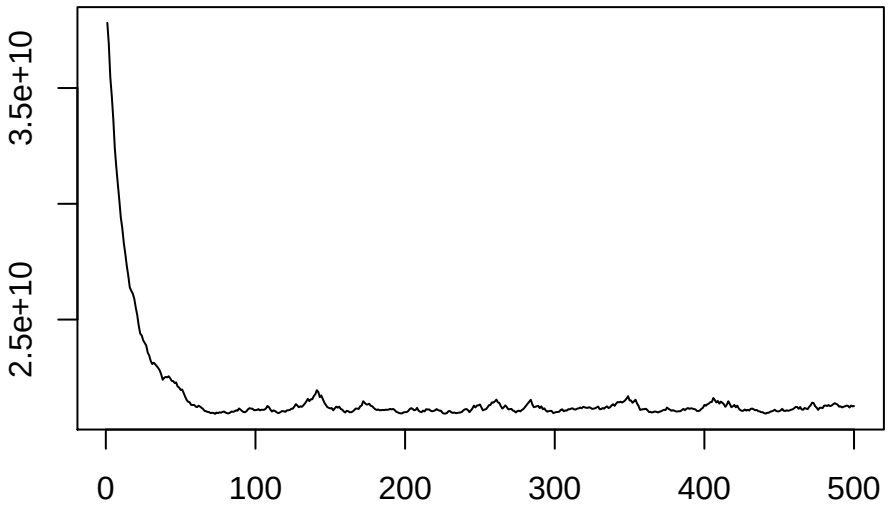







3.5e+10
2.5e+10
0
100
200
300
400
500





d.

1. Some of the hidden units can retain the color of the original picture, while the others will reverse the color.
2. All the results I got come from the modified code based on John's code. I also rewrite a new code for this homework, but the problem is that I cannot make it converge, and I am still working on it.

```
setwd("~/Documents/git/DL")
load("./hw5data.RData")
h=function(a){
        1/(1+exp(-a))
}

x=as.matrix(cbind(1,rbind(minst0_train, minst9_train)))
y=x

set.seed(1)
#split the sample into 50% training and 50% test by
#making a test dataset and removing those observations
#from the data.  Changing the proportion to large values to
#holdout most of the data usually has excellent results.
test=sample(1:nrow(x),size=round(nrow(x)*.50),replace=FALSE)
x.test=x[test,]
y.test=y[test,]
x=x[-test,]
y=y[-test,]

###Set tuning parameters###
lam=1
gam=0.001
J=50
m=1
rho=0.05
beta=0.1
############################
p=ncol(x)
```

```
K=ncol(y)

c=1;maxiter=500

betas=matrix(runif((J+1)*K,-.01,.01),nrow=K,ncol=(J+1))
alphas=matrix(runif(J*p,-.01,.01),ncol=p,nrow=J)

MSEs=rep(0,maxiter)
MSE.converge=.4 ###cross entropy convergence criteria
start=Sys.time()
repeat{
        samp.index=sample(1:nrow(x), size=m, replace = FALSE)
        y.s=y[samp.index,]
        x.s=x[samp.index,]

        z=cbind(1,h(x.s%*%t(alphas)))
        y.p=z%*%t(betas)  #each column is for a k

        KL=-rho/(z[, 2:(J+1)]+1e-6) + (1 - rho)/(1 - z[, 2:(J+1)] + 1e-9)
        y.diff=y.s-y.p

        ghat.beta=-t(t(z)%*%y.diff)/m + 2*lam*betas
        ghat.alpha=-(t((y.diff%*%betas[,2:(J+1)]+beta*KL)*z[,2:(J+1)]*(1-z[,2:(J+1)]))%*%x.s)/m + 2*lam

        betas=betas-gam*ghat.beta
        alphas=alphas-gam*ghat.alpha

        z=cbind(1,h(x%*%t(alphas)))
        y.p=z%*%t(betas)  #each column is for a k

        MSEs[c]=sum((y-y.p)^2)/m


        cat("Iteration: ",c,", MSE=",MSEs[c], "\n",sep="")

        if(MSEs[c]<MSE.converge| c>=maxiter){
                cat("Iteration: ",c,", MSE=",MSEs[c], "\n",sep="")
                break()
        }
        c=c+1
}
end=Sys.time()
end-start

# My code, but not work
setwd("~/Documents/git/DL")
load("./hw5data.RData")
sigm <- function(a){
        1/(1+exp(-a))
}

FORWARD_PROPAGATION <- function(x, W1, b1, W2, b2, m, n, K){
        z2 <- W1 %*% x + matrix(rep(b1, m), nr = n)
```

```
        h <- sigm(z2)
        z3 <- W2 %*% z2 + matrix(rep(b2, m), nr = K)
        y <- z3
        return(list(y = y, h = h))
}
BACKWARD_PROPAGATION <- function(fd, x, W2, beta, rho){
        delta3 <- -(x - fd$y)
        KL <- 0
        if (beta != 0){
                rho_hat <- apply(fd$h, 1, mean)
                KL <- -rho/(rho_hat + 1e-9) + (1 - rho)/(1 - rho_hat + 1e-9)
        }
        delta2 <- (t(W2) %*% delta3 + beta * KL) * fd$h * (1 - fd$h)
        return(list(delta2 = delta2, delta3 = delta3))
}
UPDATE_W1 <- function(delta, x, W1, alpha, lambda, m){
        return(W1 - alpha * (delta$delta2 %*% t(x)/m + lambda * W1))
}
UPDATE_b1 <- function(delta, b1, alpha){
        return(b1 - alpha * apply(delta$delta2, 1, mean))
}
UPDATE_W2 <- function(delta, fd, W2, alpha, lambda, m){
        return(W2 - alpha * (delta$delta3 %*% t(fd$h)/m + lambda * W2))
}
UPDATE_b2 <- function(delta, b2, alpha){
        return(b2 - alpha * apply(delta$delta3, 1, mean))
}
single_autoencoder <- function(input,
                               alpha = 0.01, # learning rate
                               beta = 0, # KL sparseness penalty
                               lambda = 0.01, # regularization penalty
                               maxiter = 10, # maximum iteration times
                               n = 10, # number of hidden layers
                               rho = 0.05, # sparsity parameter
                               tol = 1e-4, # tolerance
                               m = round(nrow(input)/10) # mini-batch size
){
        # Initialization
        N <- nrow(input)
        K <- ncol(input)
        W1 <- matrix(runif(n*K, -0.1, 0.1), nr = n)
        b1 <- matrix(runif(n, -0.1, 0.1))
        W2 <- matrix(runif(K*n, -0.1, 0.1), nr = K)
        b2 <- matrix(runif(K, -0.1, 0.1))
        input <- t(input)
        output <- input
        h <- matrix(0, nr = n, nc = m)
        ybar <- apply(output, 1, mean)
        Q_past <- sum((output - matrix(rep(ybar, N), ncol = N))^2)/N/2
        Q <- c()
        for (i in 1:maxiter){
                sample_ind <- sample(1:N, m)
                x <- input[, sample_ind]
```

```
                y <- output[, sample_ind]
                fp <- FORWARD_PROPAGATION(x, W1, b1, W2, b2, m, n, K)
                bp <- BACKWARD_PROPAGATION(fp, x, W2, beta, rho)
                W1 <- UPDATE_W1(bp, x, W1, alpha, lambda, m)
                b1 <- UPDATE_b1(bp, b1, alpha)
                W2 <- UPDATE_W2(bp, fp, W2, alpha, lambda, m)
                b2 <- UPDATE_b2(bp, b2, alpha)
                Q_cur <- sum((y - fp$y)^2)/m/2
                Q <- c(Q, Q_cur)
                cat("Iteration: ",i,", Q=",Q[i],"\n",sep="")
                # if (abs(Q_cur - Q_past) <= tol) break
        }
        return(list(W1 = W1, b1 = b1, W2 = W2, b2 = b2, err = Q, iter = i))
}
start.time <- Sys.time()
tt <- single_autoencoder(minst0_train[1:1000, ], m = 1, maxiter = 10)
dur <- Sys.time() - start.time
```