BNN구현

Hyung Jun Lim

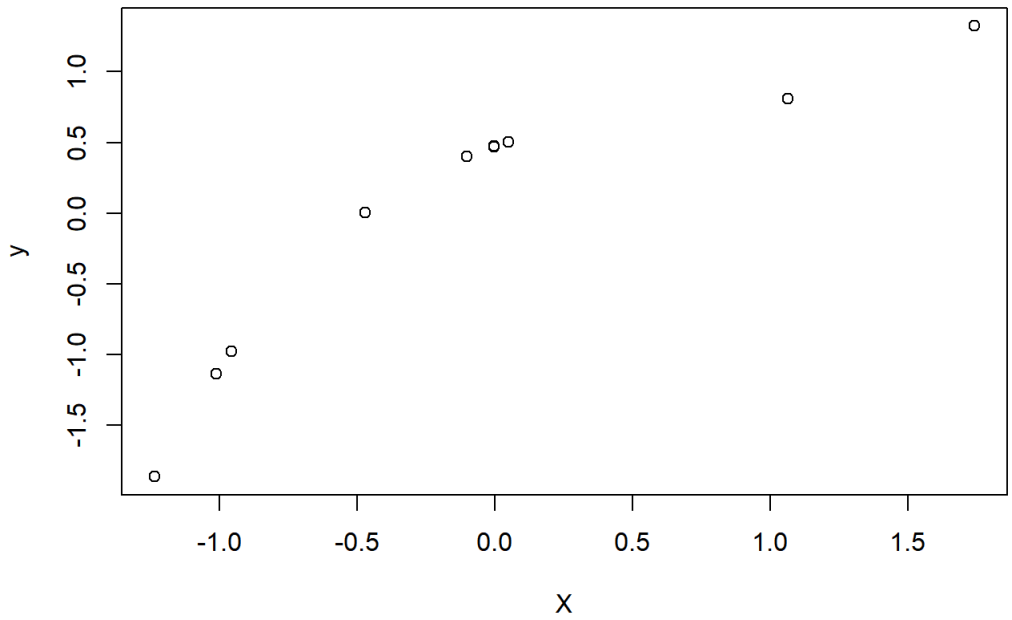
Bayesian Neural Network

Basic settings and Structure

```
## data
# input variable: X ~ N(1,0)
# sample size: 10
# target variable: Y = 3X^3 - X^2 +2X + e

# prior for parameters
# weight: w ~ N(0, (sigma_w)^2)
# bias: b ~ N(0, (sigma_b)^2)
# N(0,1) for now except for w,b for output layer

## activation function: ReLU

## structure

# hidden layers: 2
# units per layers: 8
```

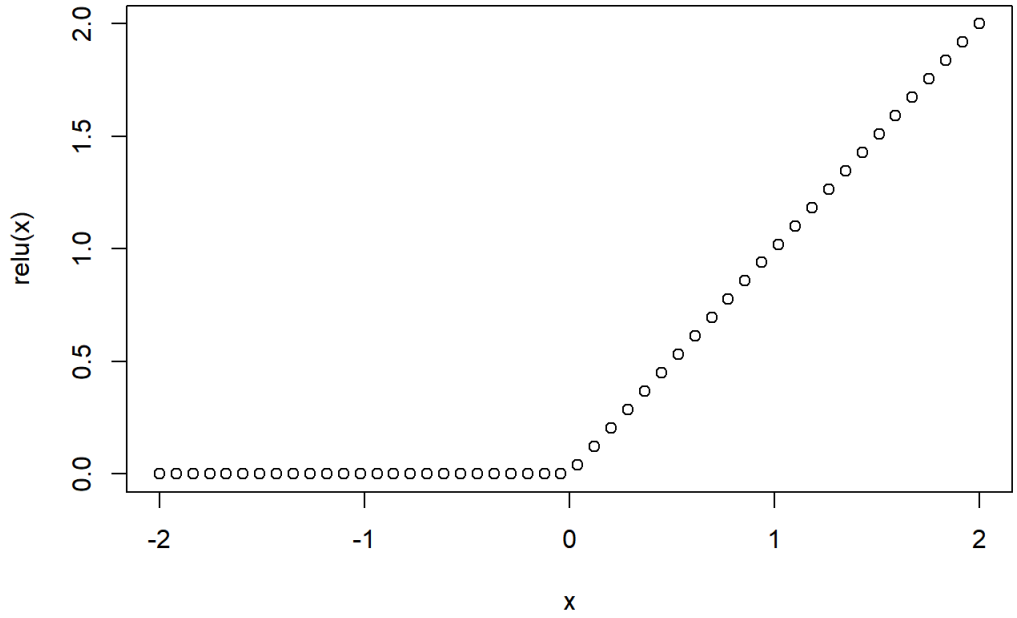
data generation

```
X <- matrix(rnorm(10),nrow=10)
y <-   X^3 - 2*X^2 + 2*X + 1
y <- scale(y)
plot(X,y)
```



Activation ReLU function

```r
# activation function (r has tanh as basic function)
x <- seq(-2,2, len=50)

relu <- function(x){
  x[which(x<0)] <- 0
  return(x)
}

plot(x, relu(x))
```



Random Paramter Sampler

```r
# setting
d_input = ncol(X)
n_layer = 2
n_unit = 16
d_output = 1
sigma_w = 1
sigma_b = 1
os_w = os_b = 1/sqrt(n_unit)


# initial weights
sample_w <- function(d_input, n_layer, n_unit, d_output, sigma_w, sigma_b){

  input_w <- array( rnorm(d_input*n_unit, mean=0, sd=sigma_w), dim=c(d_input, n_unit) )
  input_b <- array( rnorm(n_unit, mean=0, sd=sigma_b), dim=c(1,n_unit) )

  hidden_w <- array(rnorm(n_unit*n_unit*(n_layer-1), mean=0, sd=sigma_w),
                    dim=c((n_layer-1),n_unit, n_unit))
  hidden_b <- array(rnorm(n_unit*(n_layer-1), mean=0, sd=sigma_b),dim=c(1,n_layer-1))

  ow <- array(rnorm(n_unit*d_output, mean=0, sd=1/sqrt(n_unit)),dim=c(d_output,n_unit))
  ob <- array(rnorm(d_output, mean=0, sd=1/sqrt(n_unit)),dim= c(1,d_output))

  obj <- list(input_w=input_w, input_b=input_b, hidden_w=hidden_w, hidden_b=hidden_b,
              ow=ow, ob=ob)
  return(obj)
}


weights <- sample_w(d_input, n_layer, n_unit, d_output, sigma_w, sigma_b)
weights
```

```
## $input_w
##            [,1]      [,2]      [,3]     [,4]       [,5]      [,6]       [,7]
## [1,] 0.05958022 0.7657836 0.7226491 1.642411 -0.4484099 0.2622868 -0.2557812
##           [,8]       [,9]     [,10]      [,11]     [,12]     [,13]      [,14]
## [1,] 0.5209755 -0.8595165 -1.226537 -0.8863333 -1.583878 -2.739134 -0.4722763
##          [,15]      [,16]
## [1,] 0.6187551 -0.2755418
##
## $input_b
##            [,1]      [,2]      [,3]     [,4]     [,5]     [,6]     [,7]    [,8]
## [1,] -0.6389772 0.3088154 0.5210317 1.006213 1.349308 1.075146 1.878999 1.40054
##          [,9]     [,10]     [,11]    [,12]     [,13]     [,14]     [,15]
## [1,] 1.979524 0.1669162 0.2643739 1.304646 -1.255653 0.3351382 -1.299363
##           [,16]
## [1,] -0.9637945
##
## $hidden_w
## , , 1
##
##           [,1]      [,2]       [,3]       [,4]      [,5]      [,6]      [,7]
## [1,] -1.991823 -1.215328 -0.7187233 -0.4698416 0.6097993 0.8069686 0.3194716
##           [,8]      [,9]    [,10]     [,11]      [,12]     [,13]      [,14]
## [1,] -1.265923 0.3577013 0.378273 0.2955755 -0.1884963 -1.846145 -0.1093989
##         [,15]      [,16]
## [1,] 1.210091 -0.6131106
##
## , , 2
##
##          [,1]        [,2]       [,3]       [,4]       [,5]       [,6]      [,7]
## [1,] -1.79358 -0.02170076 -0.6254143 -0.2584786 0.03025013 0.09319655 0.3753532
##           [,8]       [,9]    [,10]      [,11]      [,12]     [,13]     [,14]
## [1,] -0.374428 -0.5987642 2.490541 -0.4477932 -0.8635836 -1.693304 0.9815999
##         [,15]      [,16]
## [1,] 1.980247 -0.8929623
##
## , , 3
##
##           [,1]       [,2]     [,3]        [,4]      [,5]      [,6]     [,7]
## [1,] 0.8182561 -0.0348485 -1.32216 -0.06178561 0.3083231 -1.066326 1.246246
##           [,8]     [,9]      [,10]      [,11]     [,12]    [,13]     [,14]
## [1,] 0.3416163 1.941797 -0.2077384 0.09312567 0.5161567 -1.42314 0.2560498
##          [,15]      [,16]
## [1,] 0.8296358 -0.9368019
##
## , , 4
##
##            [,1]       [,2]     [,3]      [,4]     [,5]     [,6]    [,7]
## [1,] -0.1288398 -0.5990524 1.718581 -1.357764 1.779782 1.319336 1.42387
##           [,8]      [,9]     [,10]     [,11]      [,12]      [,13]     [,14]
## [1,] 0.1611557 0.3632189 0.5544048 0.5282333 -0.2655624 -0.3165352 0.2487132
##           [,15]      [,16]
## [1,] -0.07803785 0.3312612
##
## , , 5
##
##          [,1]       [,2]      [,3]     [,4]    [,5]       [,6]     [,7]
## [1,] 1.241015 -0.2588921 0.3585835 1.045131 1.59517 -0.4715856 1.111028
##           [,8]      [,9]    [,10]      [,11]     [,12]      [,13]      [,14]
## [1,] 0.4063337 0.3358675 -1.13796 -0.6974787 -1.140388 -0.6320384 -0.7163312
##            [,15]      [,16]
## [1,] -0.08999065 -0.4161758
##
## , , 6
##
##            [,1]      [,2]       [,3]     [,4]      [,5]       [,6]      [,7]
## [1,] -0.1914521 0.8233796 -0.1835463 1.375654 0.1698006 -0.3601429 0.8694227
##           [,8]     [,9]    [,10]     [,11]      [,12]      [,13]      [,14]
## [1,] 0.3833803 1.297424 -1.06252 -1.565978 -0.3435153 -0.6579749 -0.4101127
##         [,15]     [,16]
## [1,] 2.029424 -1.440267
##
## , , 7
```

```
##
##           [,1]      [,2]      [,3]       [,4]      [,5]      [,6]     [,7]
## [1,] -1.325019 0.9471399 0.7606724 -0.4394013 0.5871517 -1.426259 1.825303
##           [,8]       [,9]    [,10]      [,11]      [,12]     [,13]      [,14]
## [1,] 0.08704906 -0.5306833 0.856025 -0.3159783 -0.2679861 -1.590482 -0.3009372
##          [,15]    [,16]
## [1,] -0.2556447 -1.44924
##
## , , 8
##
##            [,1]       [,2]      [,3]      [,4]       [,5]      [,6]      [,7]
## [1,] -0.7706265 -0.0663165 0.6281879 -1.209909 -0.2843163 0.6013943 0.2234027
##           [,8]      [,9]     [,10]      [,11]      [,12]      [,13]     [,14]
## [1,] 0.6294775 -1.624992 0.7475279 -0.5158577 -0.2801305 -0.2854639 -2.258607
##          [,15]      [,16]
## [1,] 0.05463623 0.02303611
##
## , , 9
##
##          [,1]     [,2]       [,3]      [,4]      [,5]     [,6]      [,7]
## [1,] 1.727347 1.689116 -0.1212177 0.7027363 0.1990217 -1.29051 0.3563151
##            [,8]      [,9]      [,10]     [,11]       [,12]     [,13]     [,14]
## [1,] 0.02898882 0.7423142 -0.1054608 0.8139261 0.008014693 0.5496878 -1.567548
##         [,15]     [,16]
## [1,] -2.03964 -1.227598
##
## , , 10
##
##            [,1]      [,2]        [,3]       [,4]    [,5]      [,6]      [,7]
## [1,] -0.4230693 0.1194209 0.004808185 -0.7033391 2.49051 0.4523985 0.2322826
##            [,8]     [,9]     [,10]    [,11]      [,12]      [,13]     [,14]
## [1,] 0.09452944 -1.40705 0.3878655 1.009463 -0.0950847 -0.5365738 -2.216052
##          [,15]    [,16]
## [1,] -1.045131 1.980797
##
## , , 11
##
##           [,1]        [,2]      [,3]       [,4]       [,5]     [,6]      [,7]
## [1,] -1.009651 -0.02926523 -0.293153 -0.3550195 -0.6260448 1.269536 0.7570458
##           [,8]      [,9]     [,10]     [,11]     [,12]      [,13]      [,14]
## [1,] -0.1858441 0.8873071 0.9802396 0.5675179 0.6495975 -0.8750439 0.05418207
##         [,15]      [,16]
## [1,] 1.201498 -0.3895664
##
## , , 12
##
##            [,1]      [,2]       [,3]     [,4]       [,5]      [,6]       [,7]
## [1,] -0.7851444 -1.599377 0.02900657 1.858943 -0.1320304 0.9350804 -0.5241344
##           [,8]     [,9]     [,10]     [,11]      [,12]       [,13]      [,14]
## [1,] 0.3073773 1.956547 -1.384488 -1.209909 -0.1300241 -0.05516077 -0.9887949
##          [,15]    [,16]
## [1,] -1.117382 1.208809
##
## , , 13
##
##           [,1]      [,2]      [,3]       [,4]      [,5]     [,6]      [,7]
## [1,] 0.4940795 -0.716323 -1.047559 -0.7002454 0.7388504 1.464081 0.1860664
##              [,8]       [,9]     [,10]     [,11]    [,12]    [,13]    [,14]
## [1,] -0.003239158 -0.3308384 -0.358965 0.7772569 0.360437 1.617207 1.487723
##          [,15]    [,16]
## [1,] -2.005207 1.152786
##
## , , 14
##
##          [,1]        [,2]     [,3]     [,4]        [,5]      [,6]      [,7]
## [1,] 2.332687 -0.08677437 0.935283 1.132353 -0.01208684 -1.567273 -1.560678
##           [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
## [1,] -0.1679231 -1.379993 0.6251803 0.6501111 0.9907971 -1.551781 0.2043057
##          [,15]      [,16]
## [1,] -0.4327399 -0.8242876
##
## , , 15
##
```

```
##            [,1]     [,2]     [,3]        [,4]       [,5]       [,6]       [,7]
## [1,] -0.2507042 1.992768 1.22081 -0.07117649 -1.704336 -0.3651644 -2.648526
##           [,8]       [,9]     [,10]      [,11]     [,12]      [,13]      [,14]
## [1,] 1.085261 -1.565557 -1.539929 -0.1663657 -1.252851 -0.8965843 -0.0759795
##           [,15]      [,16]
## [1,] -0.2801366 -0.3791601
##
## , , 16
##
##           [,1]       [,2]      [,3]      [,4]      [,5]      [,6]       [,7]
## [1,] 1.415036 0.03609491 -1.350023 0.4199404 -1.136302 0.7334832 -0.3998354
##            [,8]      [,9]      [,10]     [,11]      [,12]     [,13]     [,14]
## [1,] -0.3800737 0.1036329 -0.6496329 0.7743215 -0.4868583 0.2817516 0.8826472
##           [,15]     [,16]
## [1,] -1.785265 -1.038461
##
##
## $hidden_b
##            [,1]
## [1,] 0.03266811
##
## $ow
##            [,1]       [,2]      [,3]      [,4]        [,5]       [,6]       [,7]
## [1,] 0.2836611 -0.2368946 0.380985 0.1356692 -0.01804222 -0.1192834 0.02319851
##            [,8]       [,9]      [,10]      [,11]     [,12]      [,13]     [,14]
## [1,] 0.2003848 0.06744362 0.3711086 -0.1189906 0.1296332 -0.321888 0.3287608
##            [,15]      [,16]
## [1,] -0.06700798 0.1538824
##
## $ob
##           [,1]
## [1,] 0.3149434
```

Computing the output of the network with [BNN]

```
BNN <- function(X, weights, activ){

  # set result
  result <- c()

  # calculation
  input_to_hidden <-  X %*% weights$input_w + weights$input_b
  input_to_hidden <- activ(input_to_hidden)

  hidden_signal <- input_to_hidden

  if(dim(weights$hidden_w)[1]==0){
    hidden_signal <- hidden_signal
  }else{
    for(j in 1:dim(weights$hidden_w)[1]){
      hidden_signal <- activ(hidden_signal %*% t(weights$hidden_w[j,,]) + weights$hidden_b[j,])
    }
  }

  output_signal <- hidden_signal %*% t(weights$ow) + weights$ob
  result <- output_signal

  return(result)
}

BNN(X[1], weights,relu) # test
```

```
##           [,1]
## [1,] 2.359529
```

Poseterior density

```
# poseterior density
post_den <- function(y, X, weights){
  likelihood <- 1
  for(i in 1:nrow(X)){
    likelihood <- likelihood * dnorm(y[i], mean=BNN(X[i],weights,relu),sd=0.1)
  }
  return(likelihood)
}
```

Rejection Sampling acceptance Probabilty

```
# rejection prob function

reject_pr <- function(y, X, weights){

  result <- 1/1.02

  for(i in 1:nrow(X)){
    if(dnorm(y[i], mean=BNN(X[i],weights,relu),sd=0.1) == 0){
      result <- 0
    }else{
      result <- result * dnorm(y[i], mean=BNN(X[i],weights,relu),sd=0.1)/ dnorm(y[i], mean=BNN(X[i],weights,
relu), sd=0.102)  # conver the target density with Gaussian density with heavier tail
    }
  }
  return(result)
}

# test
reject_pr(y,X,weights)
```

```
## [1] 0
```

1. BNN by Rejection Sampling

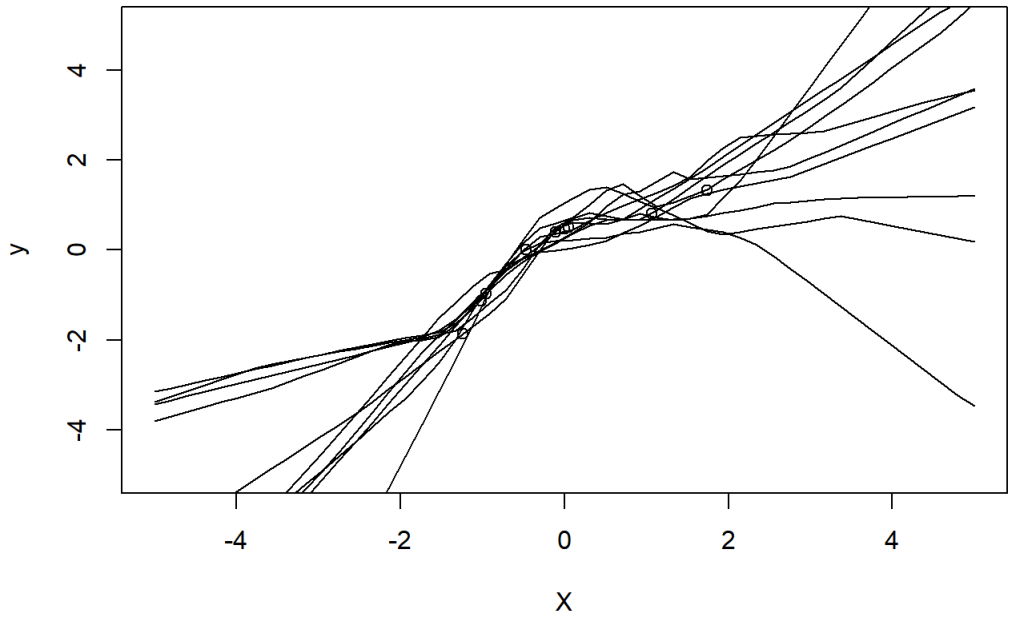
```
# sample posterior with rejection sampling

sample_post <- function(X,y){

  post <- list()
  count <- 0
  iter <- 0
  while(count < 10){

    weights <- sample_w(d_input, n_layer, n_unit, d_output, sigma_w, sigma_b)
    u <- runif(1)

    prob <- reject_pr(y,X,weights)
    if(u <= prob){
      n <- count+1
      post[[n]] <- weights
      count <- count+1
      print(count)
    }else{
      count <- count
    }
    iter = iter + 1
  }
  print(iter)
  return(post)
}

system.time(post <- sample_post(X,y))
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 7299
```

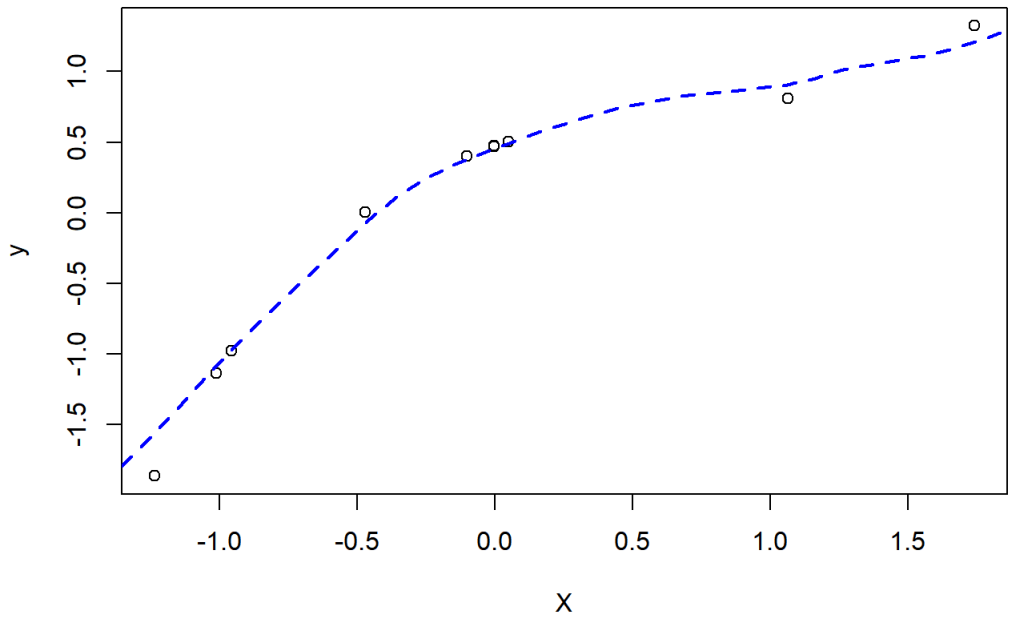
```
##    user  system elapsed
##   19.12    0.15   29.78
```

Plotting

```
# plot posterior
draw_post <- function(post){
  data1 <- seq(-5,5,len=50)
  for(j in 1:length(post)){
    output1 <- c()
    for(i in 1:length(data1)){
      output1[i] <- BNN(data1[i],post[[j]],relu)
    }
    lo <- loess(output1~data1)
    lines(lo, type = 'l', lty=1)

  }
}

# average line
draw_post_aver <- function(post){
  data1 <- seq(-5,5,len=100)

  output1 <-c()
  for(i in 1:length(data1)){
    output1[i] <- 0
    for(j in 1:length(post)){
      output1[i] <- output1[i] + BNN(data1[i],post[[j]],relu)
    }
    output1[i] <- output1[i]/length(post)
  }
  lo <- loess(output1~data1)
  lines(lo, type = 'l', lty=2, col='blue', lwd=2)
}

plot(X,y, xlim=c(-5,5),ylim=c(-5,5))

draw_post(post)
```




```
plot(X,y)
draw_post_aver(post)
```



2. BNN with Metropolis-Hastings Algorithm

Flatten and Unflattening the paramter list

```
flatten <- function(list){
  # dimension for recovery
  dim <- list
  for(k in 1:length(list)){
    dim[[k]] <- dim(list[[k]])
  }
  # flattened saved in vec
  vec <- c()
  for(i in 1:length(list)){
    vec <- c(vec, as.vector(list[[i]]))
  }
  # return list
  obj <- list(vec=vec, dim=dim)
  return(obj)
}


un_flatten <- function(vec,dim,weights){
  obj <- list()
  for(i in 1:length(dim)){
    obj[[i]] <- array(vec[1:prod(dim[[i]])], dim= dim[[i]])
    vec <- vec[ (prod(dim[[i]])+1) : length(vec) ]
  }
  obj <- setNames(obj, names(weights))
  return(obj)
}
```

vectorize the standard deviation of prior dists of paramters

```
# get sd vector

get_sd <- function(weights){
  sd <- rep(sigma_w, length(flatten(weights)$vec))
  sd[(length(sd)- (n_unit+1)*d_output +1) :length(sd)] <- 1/sqrt(n_unit)
  return(sd)
}
```

posterior density function

```
# post density
post_density <- function(y, X, weights){

  # likelihood
  likelihood <- 1
  for(i in 1:nrow(X)){
    likelihood <- likelihood * dnorm(y[i], mean=BNN(X[i],weights,relu),sd=0.1)
  }

  # prior
  flat_w <- flatten(weights)$vec
  N <- length(flat_w)
  sd_vec <- get_sd(weights)

  prior <- 1
  for(i in 1:N){
    prior <- prior * dnorm(flat_w[i], mean=0, sd=sd_vec[i])
  }

  post <- likelihood*prior

  return(post)
}

post_density(y,X,weights)
```

```
## [1] 0
```

acceptance probability with MH alogrithm

```
acceptance_pr2 <- function(y, X, w_old, w_new){

  result <- 1

  for(i in 1:nrow(X)){
    if(dnorm(y[i], mean=BNN(X[i],w_new,relu),sd=0.1) == 0 | is.na(dnorm(y[i], mean=BNN(X[i],w_new,relu),sd=0
.1)) ){
      result <- 0
    }else{
      result <-  result* dnorm(y[i], mean=BNN(X[i],w_new,relu),sd=0.1)/ dnorm(y[i], mean=BNN(X[i],w_old,relu
), sd=0.1)
    }
  }

  flat_new <- flatten(w_new)$vec
  flat_old <- flatten(w_old)$vec
  N <- length(flat_new)

  sd_vec <- get_sd(w_new)

  prior <- 1
  for(i in 1:N){
    prior <- prior * dnorm(flat_new[i], mean=0, sd=sd_vec[i])/dnorm(flat_old[i], mean=0, sd=sd_vec[i])
  }

  result <- result*prior

  result <- min(1, result)
  if(is.na(result)){
    result = 0
  }
  return(result)
}


# test
w_new <- sample_w(d_input, n_layer, n_unit, d_output, sigma_w, sigma_b)
acceptance_pr2(y,X,weights, w_new)
```

```
## [1] 1
```

Sampling BNN with MH algorithm

```
MH_post2 <- function(X,y,weights){

  post <-list()
  post[[1]] <- weights

  count <- 1
  iter <- 0
  w_old <- post[[1]]

  while(count < 15){

    w_new <- sample_w(d_input, n_layer, n_unit, d_output, sigma_w, sigma_b)
    u <- runif(1)

    prob <- acceptance_pr2(y,X,w_old,w_new)
    if(u <= prob){
      n <- count+1
      post[[n]] <- w_new
      count <- count+1
      print(count)
      w_old <- w_new
    }else{
      count <- count
    }
    iter = iter + 1
  }
  print(iter)
  return(post)
}

system.time(post <- MH_post2(X,y,weights))
```

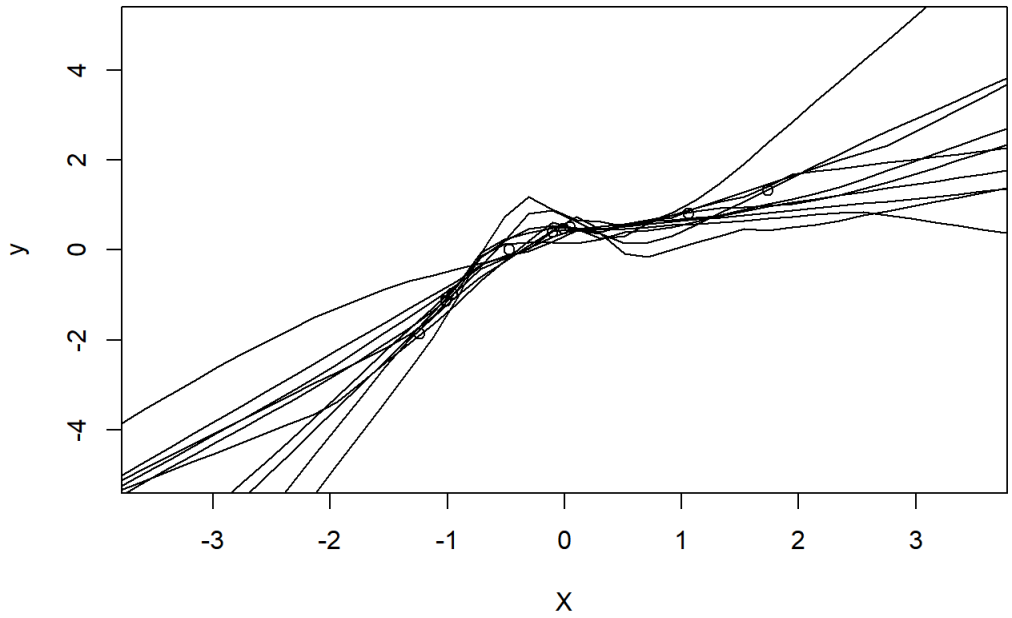
```
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 26051
```

```
##    user  system elapsed
##  147.14    1.64  344.87
```

- Much faster than the Rejection sampling despite more samples

Plotting

```
plot(X,y, xlim=c(-3.5,3.5),ylim=c(-5,5))
draw_post(post[6:15])
```



3. BNN with hybrid MCMC

gradient function for differentiating the vector

```
# gradient
grad <- function(weights,func){ # input: vector
  obj <- c()
  h <- 10^(-7)
  f <- func(weights)
  temp <- flatten(weights)
  w <- temp$vec
  dim <- temp$dim

  for(i in 1:length(w)){
    wh <- w
    wh[i] <- wh[i] +h
    wh <- un_flatten(wh, dim, weights)
    df <- func(wh) - f
    obj[i] <- df/h
  }
  return(obj)
}
```

Sampling BNN with hybrid MCMC

```
library(mvtnorm)
```

```
## Warning: package 'mvtnorm' was built under R version 3.6.3
```

```r
# hamiltonian MCMC
hybrid_post <- function(X,y,iter,step_size, sample_size){

  post <-list()
  weights <- sample_w(d_input, n_layer, n_unit, d_output, sigma_w, sigma_b)

  while(post_density(y,X,weights)==0){
    weights <- sample_w(d_input, n_layer, n_unit, d_output, sigma_w, sigma_b)
  }
  post[[1]] <- weights

  # Kinetic and Potential Energy
  V <- function(weights) -log(post_density(y,X,weights))
  #K <- function(m_vec)  -log(dmvnorm(m_vec, rep(0,N), diag(N)))

  # gradient functions
  dVdx <- function(x) grad(x, V)
  dKdp <- function(p) p

  # info
  count <- 1
  iter <- 0
  w_old <- post[[1]]
  sd_vec <- get_sd(weights)

  while(count < sample_size){

    flat_w <- flatten(w_old)
    vec <- flat_w$vec
    dim <- flat_w$dim
    N <- length(vec)
    new_vec <- c()

    # momentum sampling
    m_vec <- rmvnorm(1, rep(0,N) , diag(N))

    #leapfrog
    for(j in 1:iter){
      m_vec <- m_vec + step_size/2 * dVdx(un_flatten(vec,dim,w_old))
      vec <- vec - step_size * m_vec
      m_vec <- m_vec + step_size/2 * dVdx(un_flatten(vec,dim,w_old))
    }

    w_new <- un_flatten(vec, dim, weights)
    u <- runif(1)

    prob <- acceptance_pr2(y,X,w_old,w_new)
    if(u <= prob){
      n <- count+1
      post[[n]] <- w_new
      count <- count+1
      print(count)
      w_old <- w_new
    }else{
      count <- count
    }
    iter = iter + 1
  }
  print(iter)
  return(post)
}

system.time(post <- hybrid_post(X,y,1,0.01,15))
```

```
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 30
```

```
##    user  system elapsed
##  467.42    2.23  615.95
```

Plotting

```
plot(X,y, xlim=c(-5,5), ylim=c(-5,5))
draw_post(post[1:15])
draw_post_aver(post[1:15])
```