Stochastic Gradient Descent Mini-Batch Tutorial

Brad Rafferty

6/7/2020

First, define the various constants of the problem. The **σ** and **batch size** variables are set as vectors so as to loop through the six cases (two different **σ** values, three different **batch size** values).

```
N   <- 100      # Number of data points
n   <- 10       # Number of parameters, {a1,...,a_n}
eps <- 0.01     # Learning rate
sigs <- c(0.01, 1) # Standard deviation of the normal distribution for delta
sig_ind <- 1    # Standard deviation for an independent standard normal
distribution
n_epochs <- 10  # Number of epochs
batch_sizes <- c(1, 5, 10) # Batch size, can be 1, 5, or 10
verbose <- F    # If True, will print out epoch and batch updates
```

Define the structural model, which is a simple linear model parameterized by weights $\{a_1, \dots, a_n\}$.

```
# Structural model
mdl <- function(x, a){
  F_hat <- x %*% a # sum(a_j * x_j) for j = 1 to n
  return(F_hat)
}
```

The loss function is a least squares operator, specifically a mean squared error.

```
# Loss function
calc_loss <- function(w, X, y){
  N <- dim(X)[[1]]
  n <- dim(X)[[2]]
  F_hat <- mdl(X, w)        # Dot product
  se    <- (y - F_hat)^2    # Squared errors
  loss  <- 1/N * sum( se )  # Loss = mean squared error
  return(loss)
}
```

The gradient of the mean squared error loss function is coded below.

```
calc_grad <- function(X, y, F_hat, batch_size){
        if(batch_size == 1){ # For batch_size == 1, transposing X_b requires
an extra t() to make it a column vector
        grad  <- 2 * 1/batch_size * t(t(X)) %*% (F_hat - y)  # gradient value
      } else {
        grad  <- 2 * 1/batch_size * t(X) %*% (F_hat - y)     # gradient value
```

```
      }
  return(grad)
}
```

The structural model, loss function, and the gradient definition are now utilized in the Stochastic Gradient Descent Mini-Batch algorithm below, sgd_mb.

```
sgd_mb <- function(X, y, eps, batch_size, n_epochs,
w=matrix(numeric(dim(X)[[2]]), ncol(X), 1) ){

  # Weights initialized as 0-vector of length (y) by default in argument
declaration
  loss_history_1 <- 0  # Initialize a history of losses over each calculation
  loss_history_2 <- 0  # History of losses over each epoch
  n <- length(y) # Number of observations
  n_batches <- floor(n / batch_size) # Number of batches based on size of
each batch and size of training data
  cnt <- 1 # Counter for counting loss calculations

  for(epoch in 1:n_epochs){

    if(verbose){
      cat(sprintf('Epoch: %d\n', epoch))
    }

    # Initialize loss
    loss  <- 0
    # Permute the order of the training data before indexing the batches
    idcs  <- sample(n, n, replace = FALSE, prob = NULL)
    X     <- X[idcs, ]
    y     <- y[idcs]

    for(b in seq(from = 1, to = n, by = batch_size)){

      if(verbose){
        cat(sprintf('\tBatch: %d-%d\n', b, b+batch_size-1))
      }

      # Index X and y by the indices that constitute the current batch
      X_b <- X[b:(b+batch_size-1), ]
      y_b <- y[b:(b+batch_size-1)]
      # Calculate the model for current weights, w, and batch of X, X_b
      F_hat <- mdl(X_b, w)
      # Calculate the gradient
      g     <- calc_grad(X = X_b, y = y_b, F_hat = F_hat, batch_size =
batch_size)
      # Update weights
      w     <- w - eps * g
      # Update the loss and track its history over every batch
```

```r
      loss  <- calc_loss(w = w, X = X, y = y)
      loss_history_1[[cnt]] <- loss
      cnt = cnt + 1

    } # batch for loop

    # Track loss at each epoch as well
    loss_history_2[epoch] <- loss

  } # epoch for loop

  return(list(weights=w, loss_all=loss_history_1, loss_epoch=loss_history_2))

} # sgd_mb
```

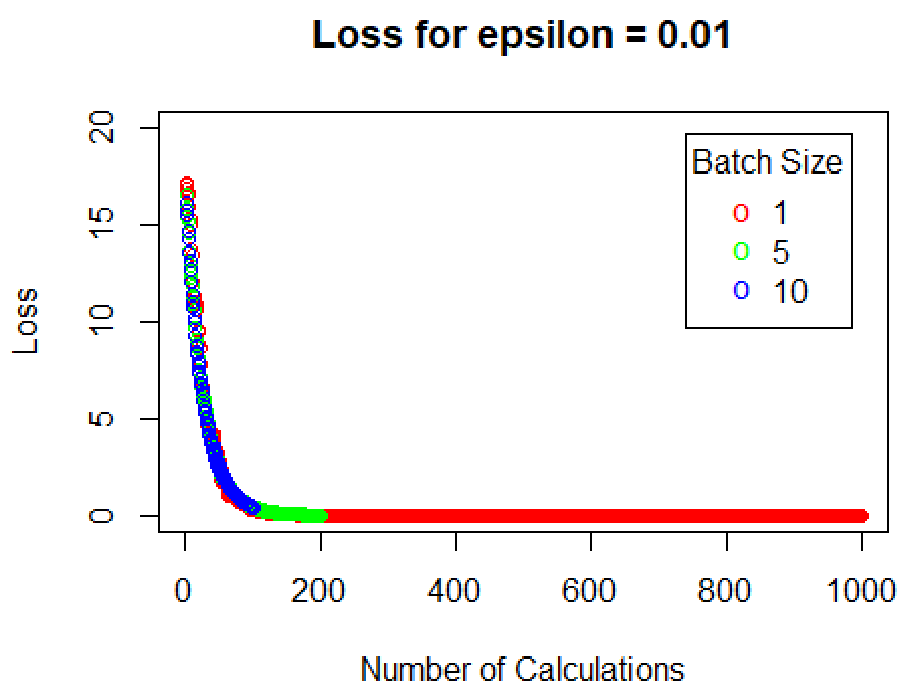
Now, we loop through the six parameter cases listed before where the **σ**, and **batch size** values are varied.

```r
cnt1 = 1
sgd_out = list()

# Loop through each batch size and sigma case
for(sig in sigs){

  # Form the response vector per its definition in the problem statement
  set.seed(20200603) # Set seeds before every rnorm() to make problem
reproducible
  X   <- matrix(rnorm(N*n, mean=0, sd=sig_ind), N, n ) # random normal matrix
with N rows, n columns
  set.seed(20200604)
  a   <- matrix(rnorm(n, 0, sd=sig_ind))
  set.seed(20200605)
  del <- matrix(rnorm(N, 0, sd=sig)) # Standard normal noise
  y   <- X %*% a + del # Response vector

  for(batch_size in batch_sizes){

    sgd_out[[cnt1]] <- sgd_mb(X, y, eps, batch_size = batch_size, n_epochs)
    cnt1 = cnt1 + 1

  }

}
```

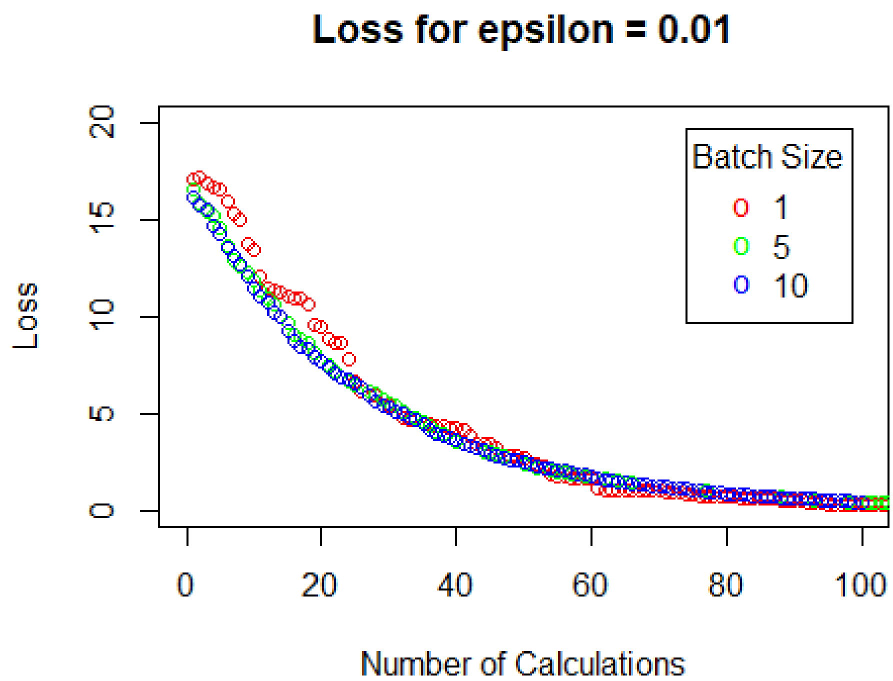
Plot loss over each epoch

Each **batch size** case corresponding to a constant **σ** is overlaid on the same plot. Two x-axis limits for the **Number of Calculations** is shown to zoom in on the early changes to the loss.

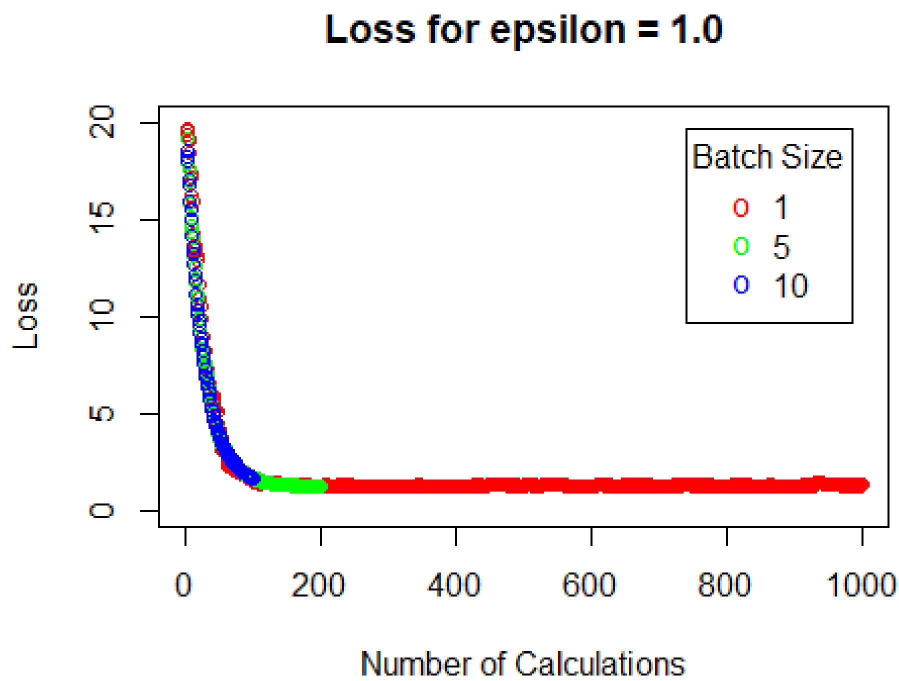
```r
cols <- c("red", "green", "blue")

plot(1:length(sgd_out[[1]]$loss_all), sgd_out[[1]]$loss_all, xlab = 'Number
of Calculations', ylab = 'Loss', main = 'Loss for epsilon = 0.01', ylim =
c(0, 20), col = cols[[1]])
points(1:length(sgd_out[[2]]$loss_all), sgd_out[[2]]$loss_all, col =
cols[[2]])
points(1:length(sgd_out[[3]]$loss_all), sgd_out[[3]]$loss_all, col =
cols[[3]])
legend("topright", inset=.05, title="Batch Size",
       c("1","5","10"), pch = c("o","o","o"), col = c(cols[[1]], cols[[2]],
cols[[3]]))
```



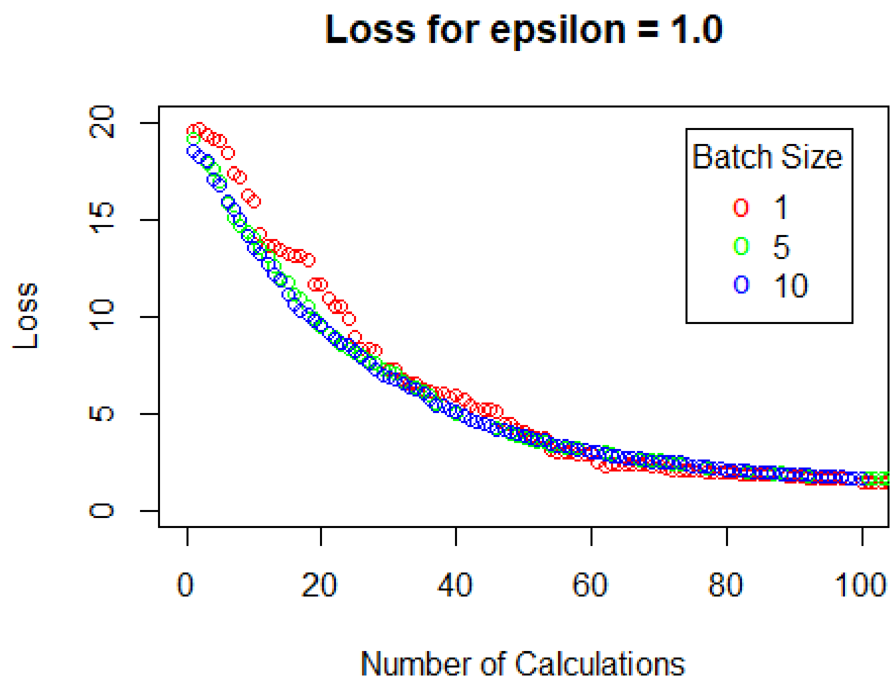

```r
plot(1:length(sgd_out[[1]]$loss_all), sgd_out[[1]]$loss_all, xlab = 'Number
of Calculations', ylab = 'Loss', main = 'Loss for epsilon = 0.01', xlim =
c(0, 100), ylim = c(0, 20), col = cols[[1]])
points(1:length(sgd_out[[2]]$loss_all), sgd_out[[2]]$loss_all, col =
cols[[2]])
points(1:length(sgd_out[[3]]$loss_all), sgd_out[[3]]$loss_all, col =
cols[[3]])
legend("topright", inset=.05, title="Batch Size",
       c("1","5","10"), pch = c("o","o","o"), col = c(cols[[1]], cols[[2]],
cols[[3]]))
```




```
plot(1:length(sgd_out[[4]]$loss_all), sgd_out[[4]]$loss_all, xlab = 'Number
of Calculations', ylab = 'Loss', main = 'Loss for epsilon = 1.0', ylim = c(0,
20), col = cols[[1]])
points(1:length(sgd_out[[5]]$loss_all), sgd_out[[5]]$loss_all, col =
cols[[2]])
points(1:length(sgd_out[[6]]$loss_all), sgd_out[[6]]$loss_all, col =
cols[[3]])
legend("topright", inset=.05, title="Batch Size",
       c("1","5","10"), pch = c("o","o","o"), col = c(cols[[1]], cols[[2]],
cols[[3]]))
```

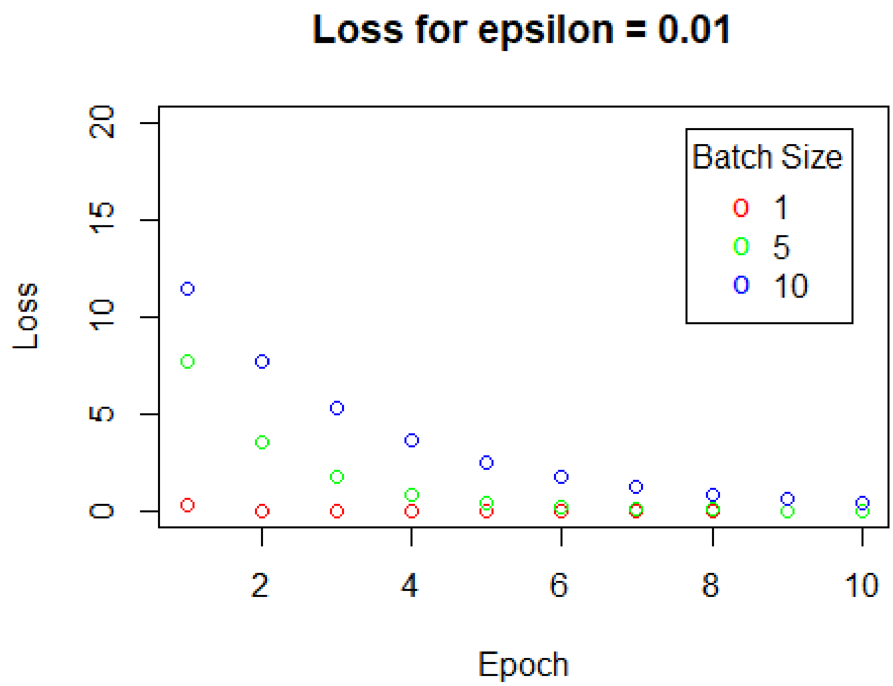



```
plot(1:length(sgd_out[[4]]$loss_all), sgd_out[[4]]$loss_all, xlab = 'Number
of Calculations', ylab = 'Loss', main = 'Loss for epsilon = 1.0', xlim = c(0,
100), ylim = c(0, 20), col = cols[[1]])
points(1:length(sgd_out[[5]]$loss_all), sgd_out[[5]]$loss_all, col =
cols[[2]])
points(1:length(sgd_out[[6]]$loss_all), sgd_out[[6]]$loss_all, col =
cols[[3]])
legend("topright", inset=.05, title="Batch Size",
       c("1","5","10"), pch = c("o","o","o"), col = c(cols[[1]], cols[[2]],
cols[[3]]))
```

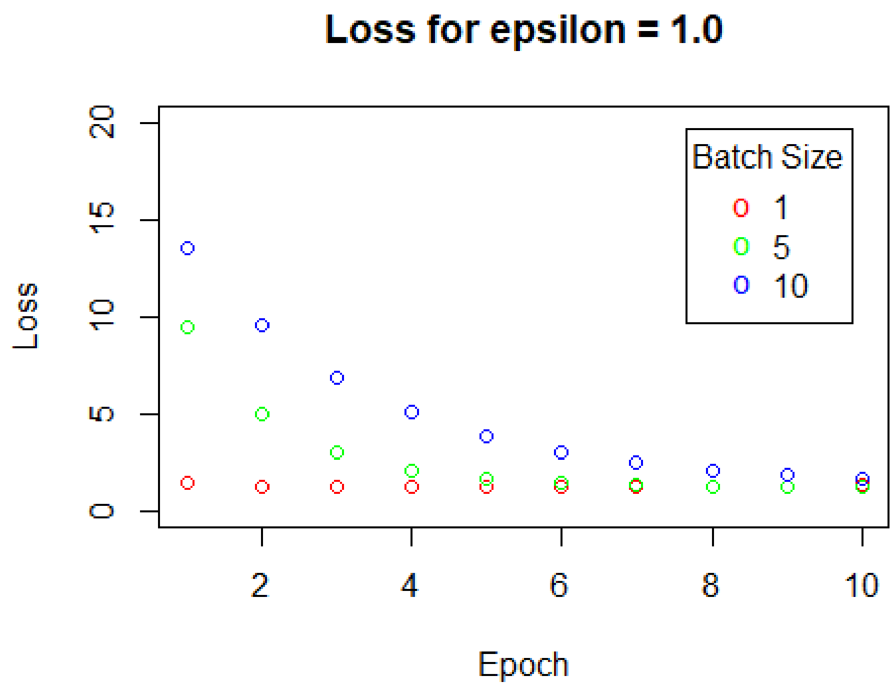


Plot loss over each epoch

```
cols <- c("red", "green", "blue")

plot(1:length(sgd_out[[1]]$loss_epoch), sgd_out[[1]]$loss_epoch, xlab =
'Epoch', ylab = 'Loss', main = 'Loss for epsilon = 0.01', ylim = c(0, 20),
col = cols[[1]])
points(1:length(sgd_out[[2]]$loss_epoch), sgd_out[[2]]$loss_epoch, col =
cols[[2]])
points(1:length(sgd_out[[3]]$loss_epoch), sgd_out[[3]]$loss_epoch, col =
cols[[3]])
legend("topright", inset=.05, title="Batch Size",
       c("1","5","10"), pch = c("o","o","o"), col = c(cols[[1]], cols[[2]],
cols[[3]]))
```




```r
plot(1:length(sgd_out[[4]]$loss_epoch), sgd_out[[4]]$loss_epoch, xlab =
'Epoch', ylab = 'Loss', main = 'Loss for epsilon = 1.0', ylim = c(0, 20), col
= cols[[1]])
points(1:length(sgd_out[[5]]$loss_epoch), sgd_out[[5]]$loss_epoch, col =
cols[[2]])
points(1:length(sgd_out[[6]]$loss_epoch), sgd_out[[6]]$loss_epoch, col =
cols[[3]])
legend("topright", inset=.05, title="Batch Size",
       c("1","5","10"), pch = c("o","o","o"), col = c(cols[[1]], cols[[2]],
cols[[3]]))
```



General Observations

- Small values of mini-batch size give a learning process that converges quickly (in terms of number of epochs), but at the cost of noise in the training process
  - However, when considering total number of gradient calculations (instead of epochs), the trade-off is different. For less than, say, 50 calculations, the higher batch sizes achieve lower losses and less variable trends in loss decrease
- Large values of mini-batch size give a learning process that converges slowly (in terms of number of epochs), but with accurate estimates of the error gradient
- Looking at total number of calculations instead of epochs, all batch sizes appear to converge at approximately the same rate