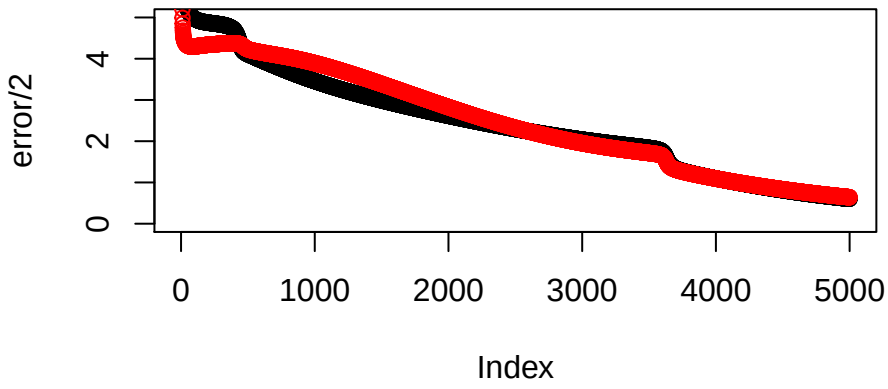
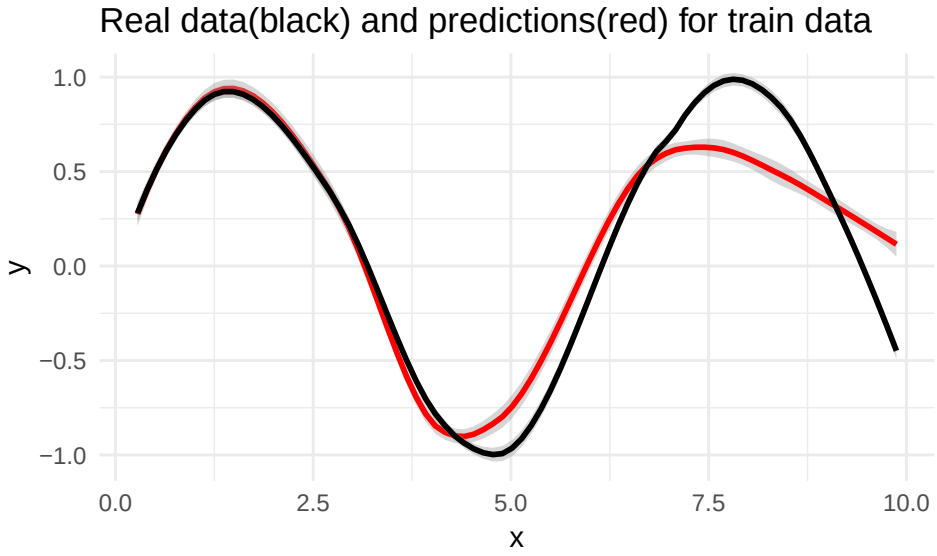
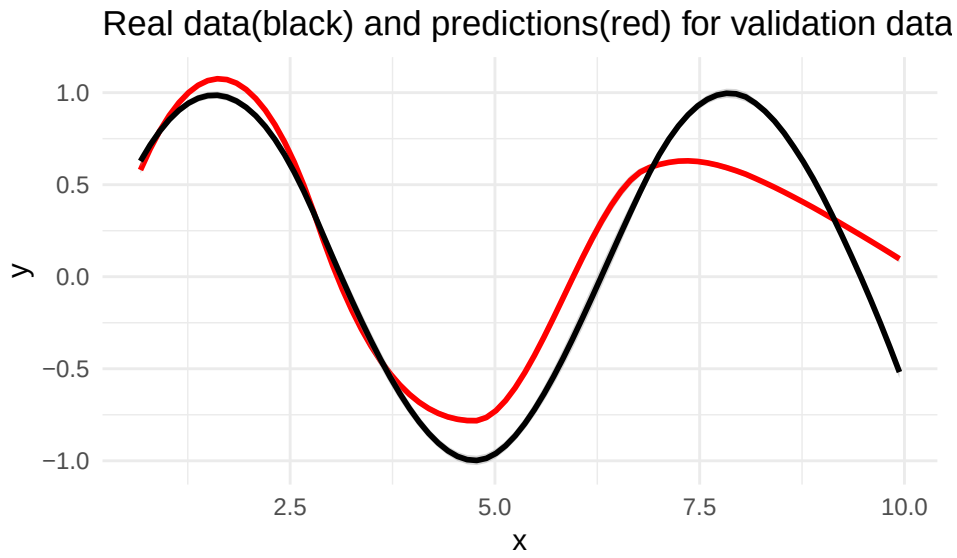
Special_Task

Andreas Stasinakis

December 18, 2018







Analysis

In this task we perform backpropagation algorithm for fitting the parameters of a NN for regression. Our target is to minimize the error and we can achieve that by changing the weights for the inputs. In the first place we choose the weights randomly, we train the neural network and we calculate the hidden units for each observation and in the end the output layer (forward propagation). After this step the backpropagation procedure starts. We have to calculate δ for the output(in this case we have only one output, so one value for δ). Moreover we calculate one the $\delta's$ for each hidden unit and in the end we have to update the weights and the bias using the partial derivatives. In this case we stop in 5000 iterations but in general We continue until our error is small enough.

From the graphs we can say that the neural network perfoms efficient enough to the real data. The final error was close to 0.6 for 5000 iterations and it could be less if we run it for more iterations. But we have also mention one imortant thing here. Most of the times, the validation error is higher than the training error. In this case after around 3500 iterations the validation error overlaps with the training error, which means that it is better not to run the algorith for more iterations. If we continue, our model maybe overfitting to the validation data. Finaly, for the last to plots of the predicted values, both seem to do quite accurate predictions.

Appendix

```r
knitr::opts_chunk$set(echo = FALSE, fig.width = 5, fig.height = 3,
                      fig.align = "center",
                      warning = F, error = F, message = F)
library(ggplot2)
set.seed(1234567890)

#create random dataset
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))

#split it in training and validation data
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation

# vector for activation untis
alpha = c()

#vector for hidden units
z = c()

# create random weights and bias
w_j <- runif(10, -1, 1)
b_j <- runif(10, -1, 1)
w_k <- runif(10, -1, 1)
b_k <- runif(1, -1, 1)

# learning rate
l_rate <- 1/nrow(tr)^2
n_ite = 5000 # number of iterations
error <- rep(0, n_ite) #error for training data
error_va <- rep(0, n_ite) # error for validation data


#loop for all iterations
for(i in 1:n_ite) {
  # we want to reduce the error
  # for that reason we are upadating the weights
  # first we calculater the error
  # error for training data
    for(n in 1:nrow(tr)){
      # same process like forward propagation
      # we use the train data to calculate the error
      # calculate activation unit
      alpha = w_j * tr[n,1] + b_j

      # Use the activation function tanh to calculate all the hidden units
      # from the hidden layer
      z = tanh(alpha)

      # calculate output layer
      y_k = sum(w_k * z) + b_k
```

```
    # calculate first error for training data
    error[i] = error[i] + (y_k - tr[n,2])^2
}

# error for validation data

for(n in 1:nrow(va)){

  # same process as before

    # calculate activation unit

    alpha = w_j * va[n, 1] + b_j

    # hidden units for the hidden layer using the activation function

    z = tanh(alpha)

    # calculate output layer
    y_k = sum(w_k * z) + b_k

    # calculate error for validation data
    error_va[i] = error_va[i] + (y_k - va[n, 2])^2
  }

#cat("i: ", i, ", error: ", error[i]/2, ", error_va: ", error_va[i]/2, "\n")
#flush.console()

for(n in 1:nrow(tr)) {

  # forward propagation: Your code here
  # calculate the activation units
  alpha = tr[n,1]*w_j + b_j

  #calculate the Z's
  z = tanh(alpha)

  # produce the output
  y_k = sum(z*w_k) + b_k

  #Compute delta for the output units(one value).
  delta_k = y_k - tr[n,2]

  # backward propagation:
  # we have to update  our weights
  # calculate delta_j for the hidden units
  # vector with one value for each hidden unit
  delta_j = (1- z^2)*w_k*delta_k

  #For each new weight we have the old one ,
  #- the learning rate * the partial derivative of error function
  #with respect to the weight
  # one vector of with lenght as many hidden units we have (10)
```

```
    #With this way we update the weights for the hidden units
     w_k = w_k - l_rate*(delta_k*z)

    # Now we update the weights for the input
    # Same formula as before ,instead the hidden units,
    #we take the training value
     w_j = w_j - l_rate*(delta_j*tr[n,1])

    #Except for the weights we also have to update the bais.
    # we use the same formula as in the weights with one different.
    # bias for the hidden unit
     b_k = b_k - l_rate*delta_k

     # bias for the input
     b_j = b_j - l_rate*delta_j

  }
}

# plot the error (both training and validation ) for all the errors
 plot(error/2, ylim=c(0, 5))
 points(error_va/2, col = "red")

# predictions for training data
predictions_train = c()

for(n in 1:nrow(tr)){

  #The same procedure as before
  alpha = w_j * tr[n,1] + b_j
  z = tanh(alpha)
  y_k = sum(w_k * z) + b_k
  predictions_train = c(predictions_train,y_k)
}


#predict for the validation data
predictions_valid = c()
for(n in 1:nrow(va)){

  #The same procedure as before
  alpha = w_j * va[n,1] + b_j
  z = tanh(alpha)
  y_k = sum(w_k * z) + b_k
  predictions_valid = c(predictions_valid,y_k)
}

#Create plot for the real values
# train and validation data in one plot
# create a common data frame
df = cbind(tr,va)
colnames(df) = c("train_var","train_sin","valid_var","valid_sin")
```

```
#plot the real values
# ggplot(df)+
#   geom_smooth(aes(x = df[,1], y = df[,2]),color = "red") +
#   geom_smooth(aes(x = df[,3], y = df[,4]), color = "black")

# plot the predictions for train data
#create a data frame

df_train = data.frame(real = tr$Var,
                      predictions = predictions_train,real_va = tr$Sin)

ggplot(df_train)+
  stat_smooth(mapping = aes(x = df_train$real,y = df_train$predictions ),
              color = "red",span  = 0.4)+
  stat_smooth(mapping = aes(x = df_train$real,y = df_train$real_va ),
              color = "black", span = 0.40 ) +
  labs(title = "Real data(black) and predictions(red) for train data",
       x = "x" , y = "y") +
  theme_minimal()


# plot the predictions for validation data
#create a data frame

df_valid = data.frame(real = va$Var,
                      predictions = predictions_valid,real_va = va$Sin)

ggplot(df_valid)+
  stat_smooth(mapping = aes(x = df_valid$real,y = df_valid$predictions ),
              color = "red",span  = 0.4)+
  stat_smooth(mapping = aes(x = df_valid$real,y = df_valid$real_va ),
              color = "black",span = 0.40 ) +
  labs(title = "Real data(black) and predictions(red) for validation data",
       x = "x" , y = "y") +
  theme_minimal()
```