Special Task 5

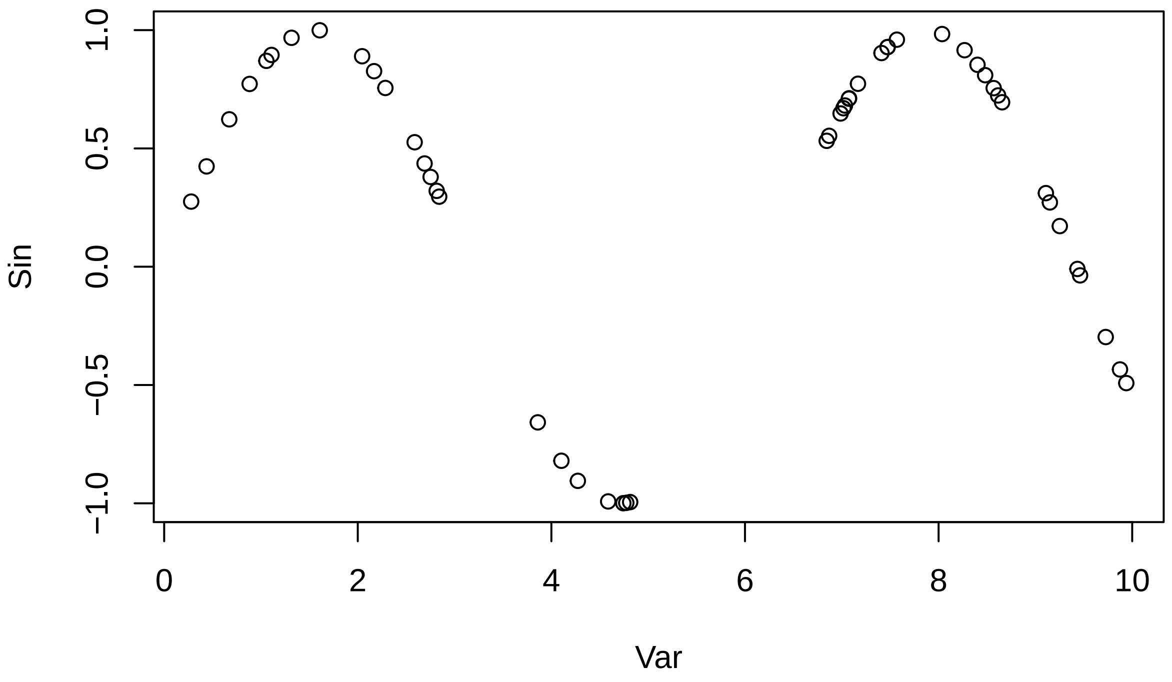
Mim Kemal Tekin

12/18/2018

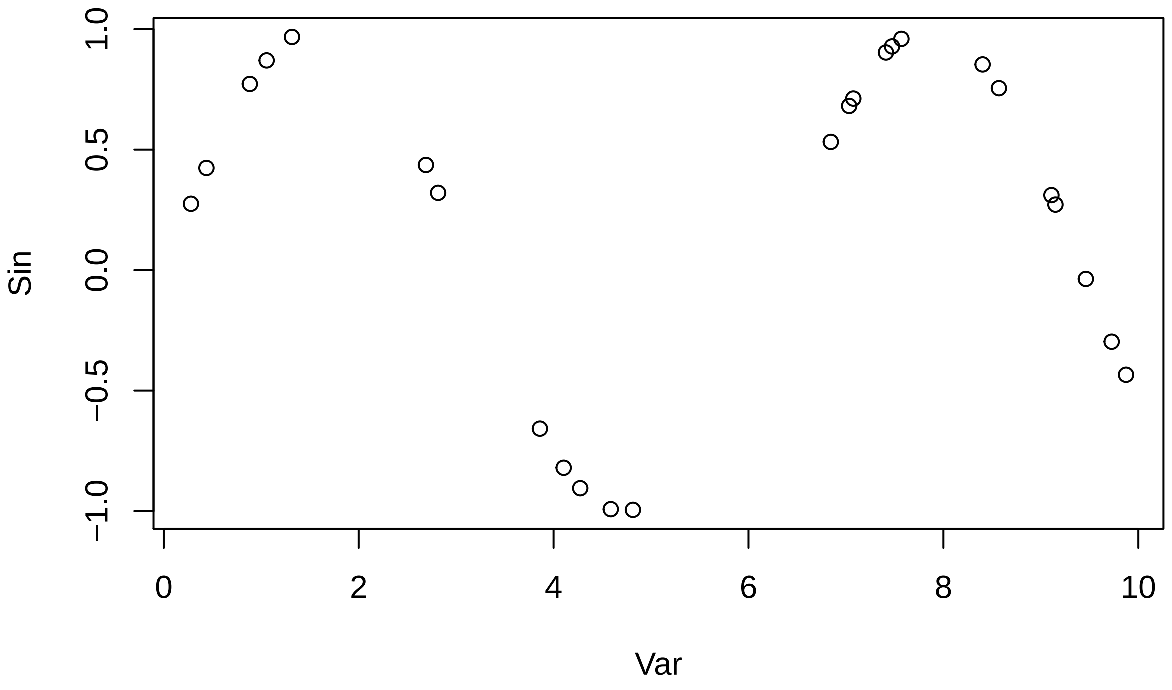
Special Task 4

In this task we implement a back propagation algorithm and train a Neural Network Regression. We generate a data from uniform distribution and we take sinus of all observations. In this tash the target of neural network is predicting this sinus values correctly. We implement some weightes and bias values randomly from uniform distribution again. And in forward propagation it calculates 10 hidden units in the hidden layer and after that value of output layer by using weights and bias values. After this iteration with backpropagation, we calculate new optimized weights for between input and hidden layer; hidden and output layer. We can see some output plots below regarding to dataset, model error and predictions.

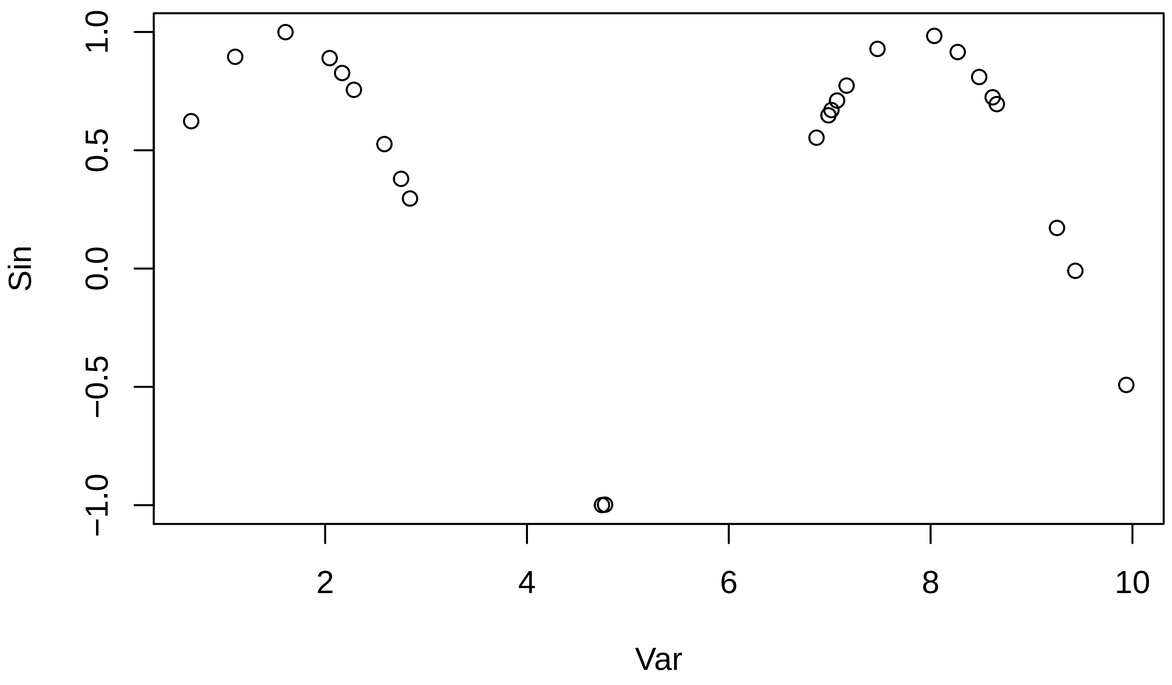
```
## [1] "Original Data"
```



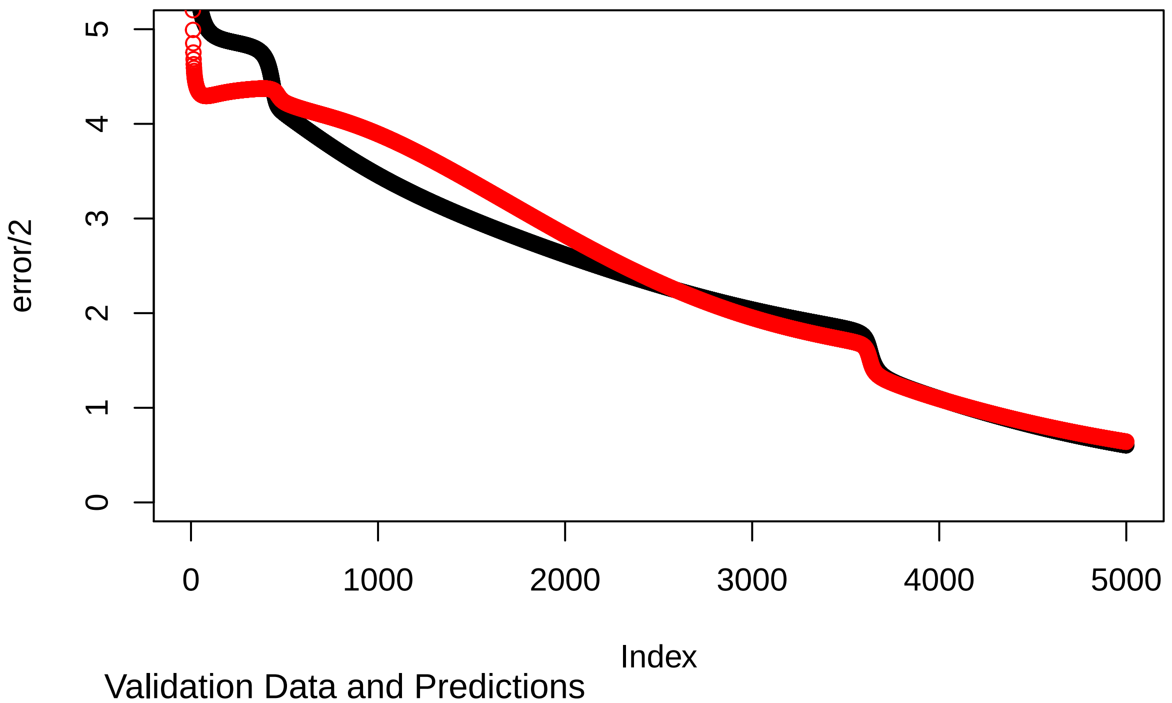
```
## [1] "Train Data"
```

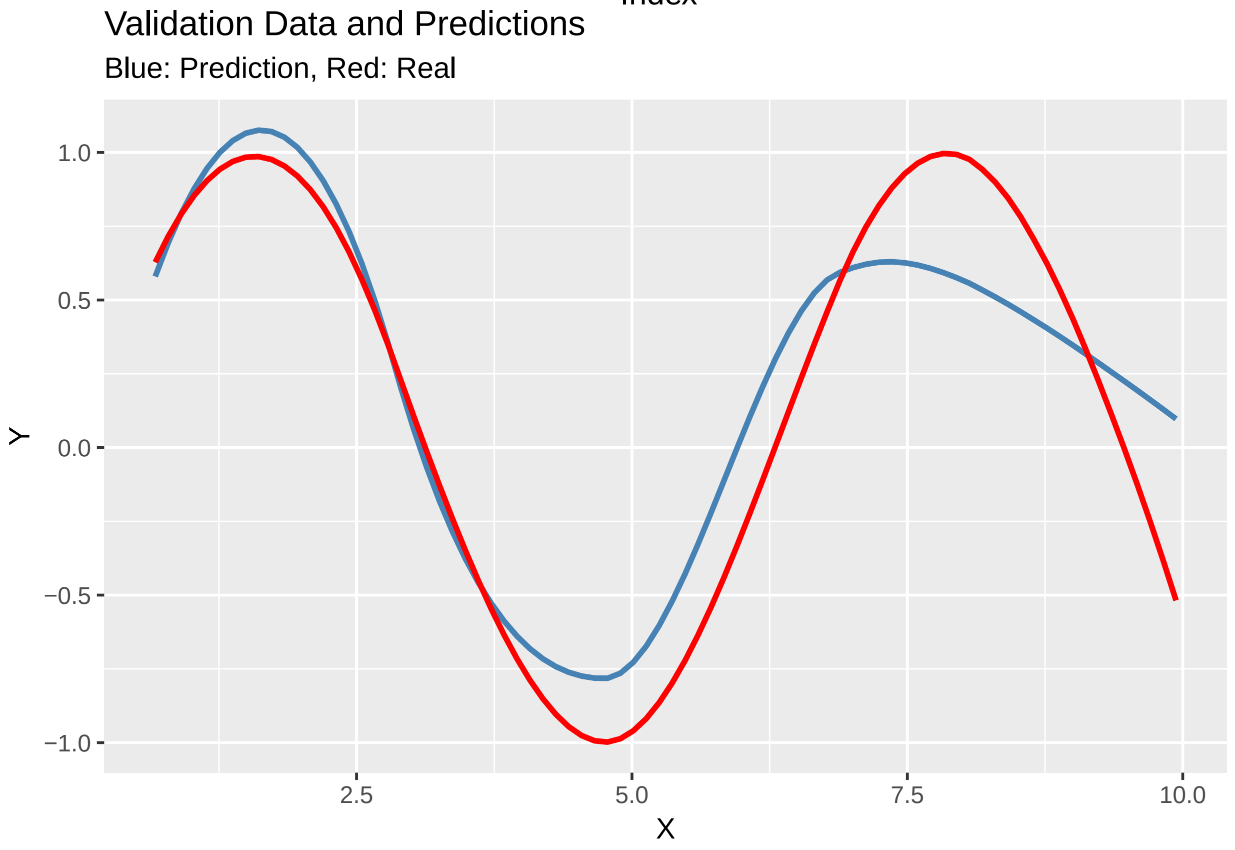


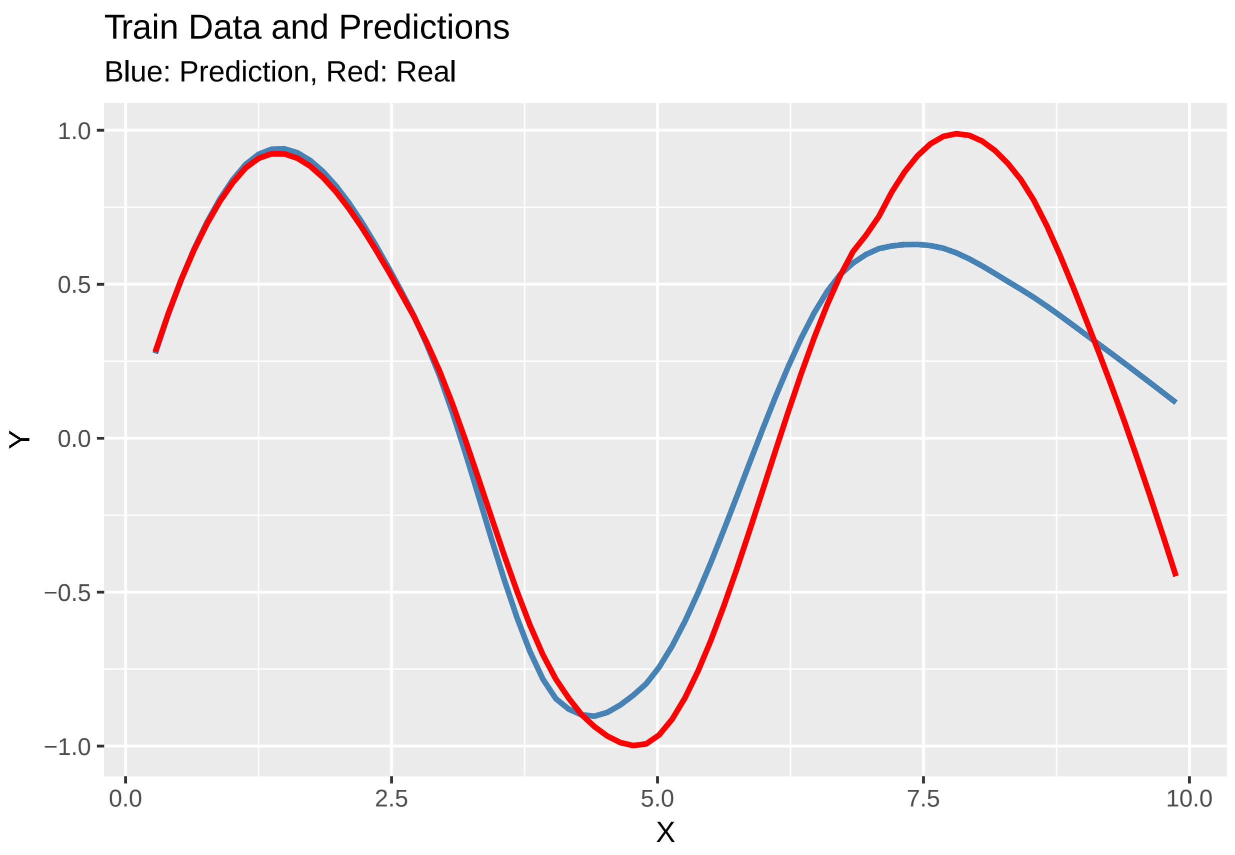
```
## [1] "Validation Data"
```



```
## [1] "Errors"
```


error/2
0
1
2
3
4
5
0
1000
2000
3000
4000
5000
Index


Validation Data and Predictions
Blue: Prediction, Red: Real
Y
1.0
0.5
0.0
−0.5
−1.0
2.5
5.0
7.5
10.0
X



Appendix

```r
knitr::opts_chunk$set(echo = F, message = F, warning = F,
                      error = F)

library(ggplot2)
  set.seed(1234567890)
  Var <- runif(50, 0, 10)
  trva <- data.frame(Var, Sin=sin(Var))
  tr <- trva[1:25,] # Training
  va <- trva[26:50,] # Validation

  print("Original Data")
  plot(trva)
  print("Train Data")
  plot(tr)
  print("Validation Data")
  plot(va)

  w_j <- runif(10, -1, 1)
  b_j <- runif(10, -1, 1)
  w_k <- runif(10, -1, 1)
  b_k <- runif(1, -1, 1)
  l_rate <- 1/nrow(tr)^2
  n_ite = 5000
```

```r
error <- rep(0, n_ite)
error_va <- rep(0, n_ite)

for(i in 1:n_ite) {
  ### error computation
  # error for training data
  for(n in 1:nrow(tr)){
    # same process like forward propagation with train data in order to
    # find ith iteration train errors
    # calculate activation value
    a_j = w_j * tr[n, ]$Var + b_j
    # hidden unit and activation function
    z_j = tanh(a_j)
    # calculate output layer
    y_k = sum(w_k * z_j) + b_k
    # calculate error
    error[i] = error[i] + (y_k - tr[n, ]$Sin)^2
  }
  # error for validation data
  for(n in 1:nrow(va)){
    # same process like forward propagation with validation data in order to
    # find ith iteration validation errors
    # calculate activation value
    a_j = w_j * va[n, ]$Var + b_j
    # hidden unit and activation function
    z_j = tanh(a_j)
    # calculate output layer
    y_k = sum(w_k * z_j) + b_k
    # calculate error
    error_va[i] = error_va[i] + (y_k - va[n, ]$Sin)^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 #########
    ### we will calculate predictions from our inputs by using weights and
    ### bias values
    # calculate activation value
    # we multiply our current weights and our current input (n)
    # and add the bias value
    a_j = w_j * tr[n, ]$Var + b_j
    # hidden unit and activation function
    # apply our activation function which is tanh in this case
    z_j = tanh(a_j)
    # calculate output layer
    # z_j was the values of hidden layer which contains 10 units
    # we multiply these results with the output weights and getting sum of
    # them and add output bias to this sum
    y_k = sum(w_k * z_j) + b_k
    ######### backward propagation #########
    ### which means we will check our outputs with current weights, and
    ### we will change these weights and biases
    # diffirence at output layer
    # delta_k is differences between the predicted value with
    # forward propagation and the true values
    d_k = y_k - tr[n, ]$Sin
    # difference between hidden layer
    # delta_j is the correction weight for hidden layer depends on the
    # difference between d_k
    d_j = (1 - z_j^2) * w_k*d_k

    # we are going back after this case
    # we set new weights between hidden layer and output layer
    # by substracting the weighted hidden unit values from the w_k weightes
    # l_rate is learning rate which smooth the learning effect
    w_k = w_k - l_rate*d_k*z_j
    # calculate new bias for hidden units to output layer with d_k
    b_k = b_k - l_rate*d_k
    # calculate weights beteen input layer and hidden layer
    w_j = w_j - l_rate*d_j*tr[n, ]$Var
    # calculate new bias for inputs with dj
    b_j = b_j - l_rate*d_j

  }
}

print("Errors")
plot(error/2, ylim=c(0, 5))
points(error_va/2, col = "red")

# predictions

pred_train = c()
for(n in 1:nrow(tr)){
  # same process like forward propagation with train data in order to
  # find ith iteration train errors
  # calculate activation value
  a_j = w_j * tr[n, ]$Var + b_j
  # hidden unit and activation function
  z_j = tanh(a_j)
  # calculate output layer
  pred_train[n] = sum(w_k * z_j) + b_k
}


pred_val = c()
for(n in 1:nrow(va)){
    # same process like forward propagation with validation data in order to
    # find ith iteration validation errors
    # calculate activation value
    a_j = w_j * va[n, ]$Var + b_j
    # hidden unit and activation function
    z_j = tanh(a_j)
    # calculate output layer
    pred_val[n] = sum(w_k * z_j) + b_k
}
```

```r
# create dataframes to plots
df_val_plot = data.frame(real = va$Var,
                         pred = pred_val,
                         var = va$Sin)

df_train_plot = data.frame(real = tr$Var,
                           pred = pred_train,
                           var = tr$Sin)

ggplot(df_val_plot) +
  stat_smooth(aes(x = real, y = pred), span = 0.4, se=F,
              col = "steelblue") +
  stat_smooth(aes(x = real, y = var), span = 0.4, se=F,
              col = "red") +
  labs(title = "Validation Data and Predictions",
       subtitle = "Blue: Prediction, Red: Real",
       x = "X", y = "Y")

ggplot(df_train_plot) +
  stat_smooth(aes(x = real, y = pred), span = 0.4, se = F,
              col = "steelblue") +
  stat_smooth(aes(x = real, y = var), span = 0.4, se = F,
              col = "red") +
  labs(title = "Train Data and Predictions",
       subtitle = "Blue: Prediction, Red: Real",
       x = "X", y = "Y")
```