Assignment 3

Keshav Padiyar Manuru

03/01/2021

Assignment 3. Neural networks

Implementation of Neural networks using back propagation algorithm. The NN has single input and output units with 10 hidden units. Sigmoid $h(a) = \frac{1}{1+e^{-a}}$ and Tanh $h(a) = \tanh(a)$ functions are used as activation functions.

```
Nnet <- function(a_function){

  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


  # plot(trva)
  # plot(tr)
  # plot(va)

  ##  Initializing Variables

  w_j <- runif(10, -1, 1) # Input weight
  b_j <- runif(10, -1, 1) # Input Bias
  w_k <- runif(10, -1, 1) # Output weight
  b_k <- runif(1, -1, 1)  # Output bias
  l_rate <- 1/nrow(tr)^2  # learning rate
  n_ite = 25000           # Number of Iterations
  error <- rep(0, n_ite)  # Error in Training
  error_va <- rep(0, n_ite) # Error in Validation

  for(i in 1:n_ite) {
  # error computation: Your code here

  ## Error Computation for Test

  a_j <- (as.matrix(tr$Var))%*% (w_j) +
    matrix(rep(b_j,nrow(tr)),ncol=length(b_j),byrow=T) # Activations: aj

  ## Condition to switch the activation functions
  if (a_function == "sigmoid"){
```

```
      z_j <- 1/(1+exp(-a_j)) # Equation of Sigmoid function
  }else{
      z_j <- tanh(a_j) # Equation of Tanh function
  }
a_k <- (z_j %*% w_k) +
  matrix(rep(b_k,nrow(tr)),ncol=length(b_k),byrow=T) # Output Activations: ak
error[i] <- sum((tr$Sin - as.numeric(a_k))^2) # Least Square Error in test

## Error Computation for validation (Same steps as test, used data set va)
a_j <- (as.matrix(va$Var))%*% (w_j) + matrix(rep(b_j,nrow(va)),ncol=length(b_j),byrow=T)
if (a_function == "sigmoid"){
      z_j <- 1/(1+exp(-a_j))
  }else{
      z_j <- tanh(a_j)
  }
a_k <- (z_j %*% w_k) + matrix(rep(b_k,nrow(va)),ncol=length(b_k),byrow=T)
error_va[i] <- sum((va$Sin - as.numeric(a_k))^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
      # Renaming Variables for ease of understanding
      w_i <- w_j # Input weight
      w_o <- w_k # Output weight
      b_i <- b_j  # Input Bias
      b_o <- b_k # Output Bias
      x <- tr[n,1] # Training data
      a_j <-(w_i * x)+ b_i #Input Activations
```

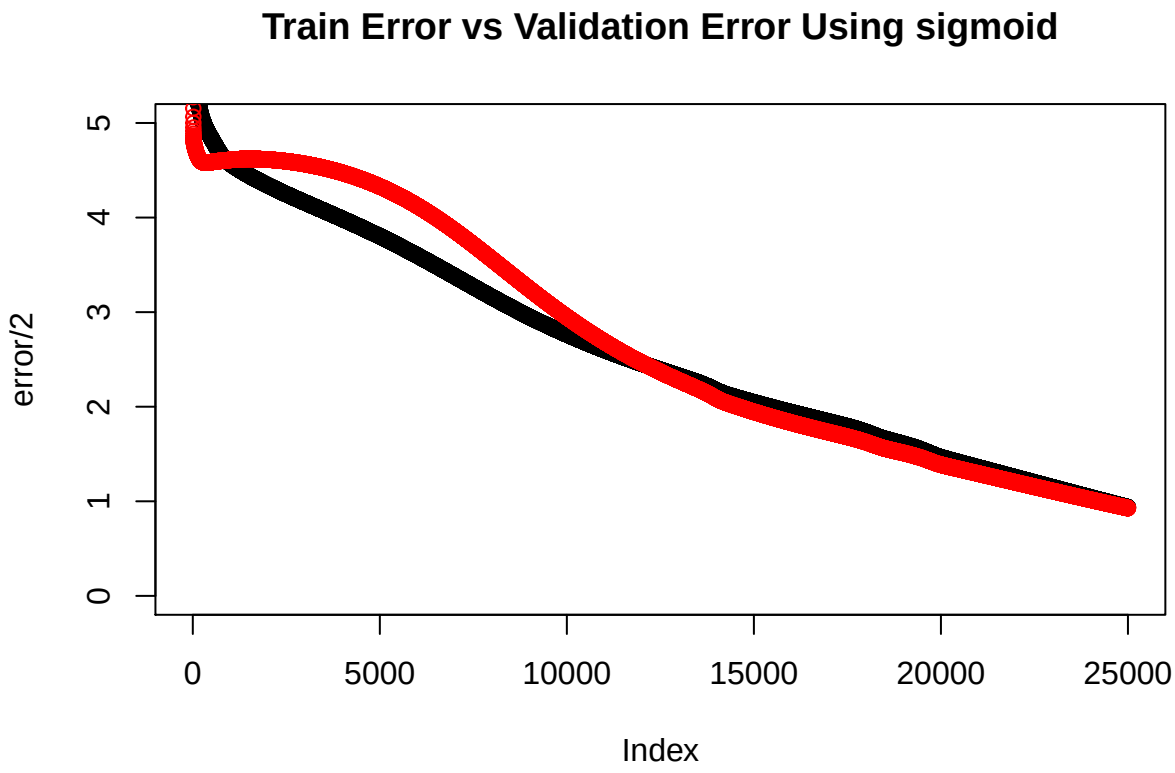
```
    ## Condition to switch the activation functions
    if (a_function == "sigmoid"){

          z_j <- 1/(1+exp(-a_j))

      }else{

          z_j <- tanh(a_j)

      }

      a_k <- (z_j %*% w_o) + b_o # Output activations

  ## End of  Forward Propagation

  # backward propagation: Your code here

      y_k <- as.vector(a_k) # For regression output activation is target

      delta_k <- (as.numeric(y_k-tr[n,2])) # partial derivative wrt output activation

        if (a_function == "sigmoid"){

             # Incase of sigmoid function - derivative
             delta_j <-  (z_j * (1-z_j)) *  (w_o * delta_k) # Partial derivative wrt input avtivation

        }else{

            # Incase of sigmoid function - derivative
            delta_j <-  (1-z_j^2) *  (w_o * delta_k)

        }


      E_wi <-delta_j * x # gradient of input weights

      E_wo <- z_j * delta_k # gradient of output weights

      w_ni <- w_j - (l_rate * E_wi) # new input weights using stochastic gradient descent method

      w_no <- w_k - (l_rate * E_wo) # new output weights

      w_j <- w_ni # replace old weights with new weights

      w_k <- w_no

      b_j <- b_i - (l_rate * delta_j) # new input bias using stochastic gradient descent

      b_k <- b_o - (l_rate * delta_k) # new output bias


}
```

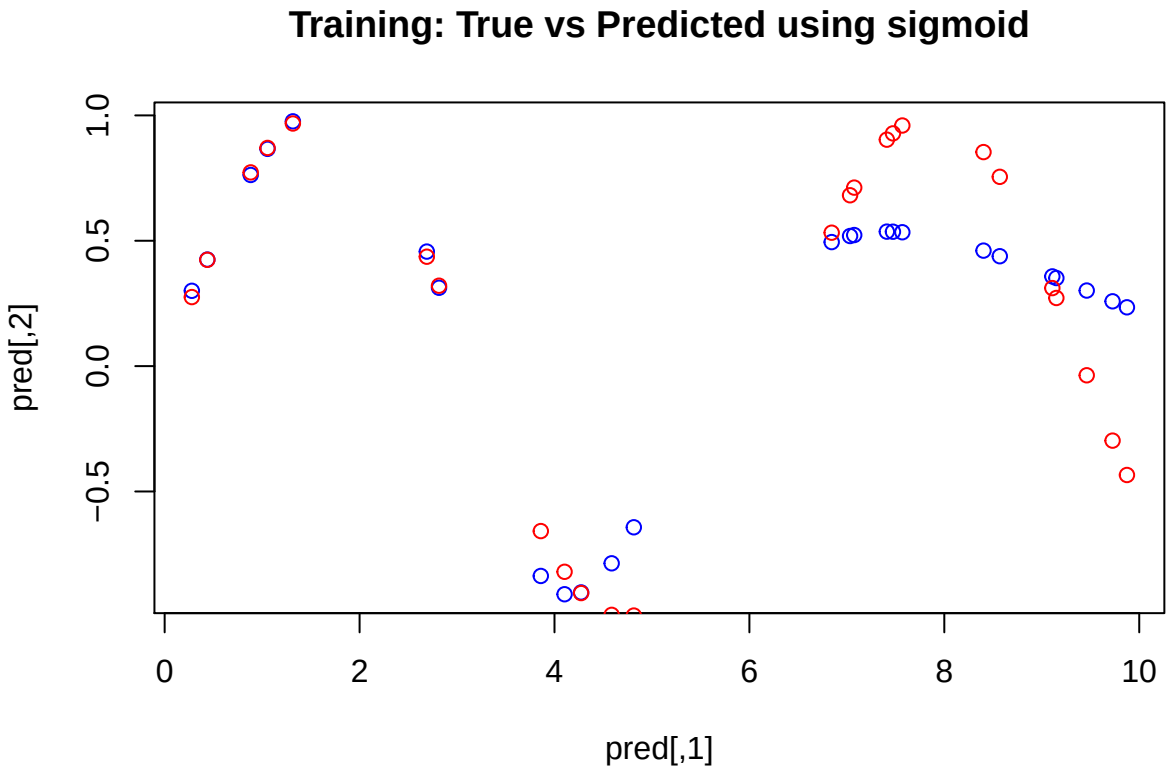
```
}

  # print final weights and errors
cat("Converged Weights and Biases Using ",a_function,"\n")
cat("Input Weight: ", w_j,"\n")
cat("Input Bias: ",b_j, "\n")
cat("Output Weight: ",w_k,"\n")
cat("Output Bias: ",b_k,"\n")

plot(error/2, ylim=c(0, 5), main = paste0("Train Error vs Validation Error Using ",a_function))
points(error_va/2, col = "red")


# plot prediction on training data

## predicting the target values using converged weights and biases for training data

a_j <- (as.matrix(tr$Var))%*% (w_j) + matrix(rep(b_j,nrow(tr)),ncol=length(b_j),byrow=T)

if (a_function == "sigmoid"){

      z_j <- 1/(1+exp(-a_j))

  }else{

      z_j <- tanh(a_j)

  }

y_k <- (z_j %*% w_k) + matrix(rep(b_k,nrow(tr)),ncol=length(b_k),byrow=T)

pred <- cbind(tr$Var,y_k)

plot(pred,col="blue",main=paste0("Training: True vs Predicted using ", a_function))

points(tr, col = "red")

# plot prediction on validation data

## predicting the target values using converged weights and biases for validation data
a_j <- (as.matrix(va$Var))%*% (w_j) + matrix(rep(b_j,nrow(va)),ncol=length(b_j),byrow=T)

if (a_function == "sigmoid"){

      z_j <- 1/(1+exp(-a_j))

  }else{

      z_j <- tanh(a_j)

  }

y_k <- (z_j %*% w_k) + matrix(rep(b_k,nrow(va)),ncol=length(b_k),byrow=T)
```

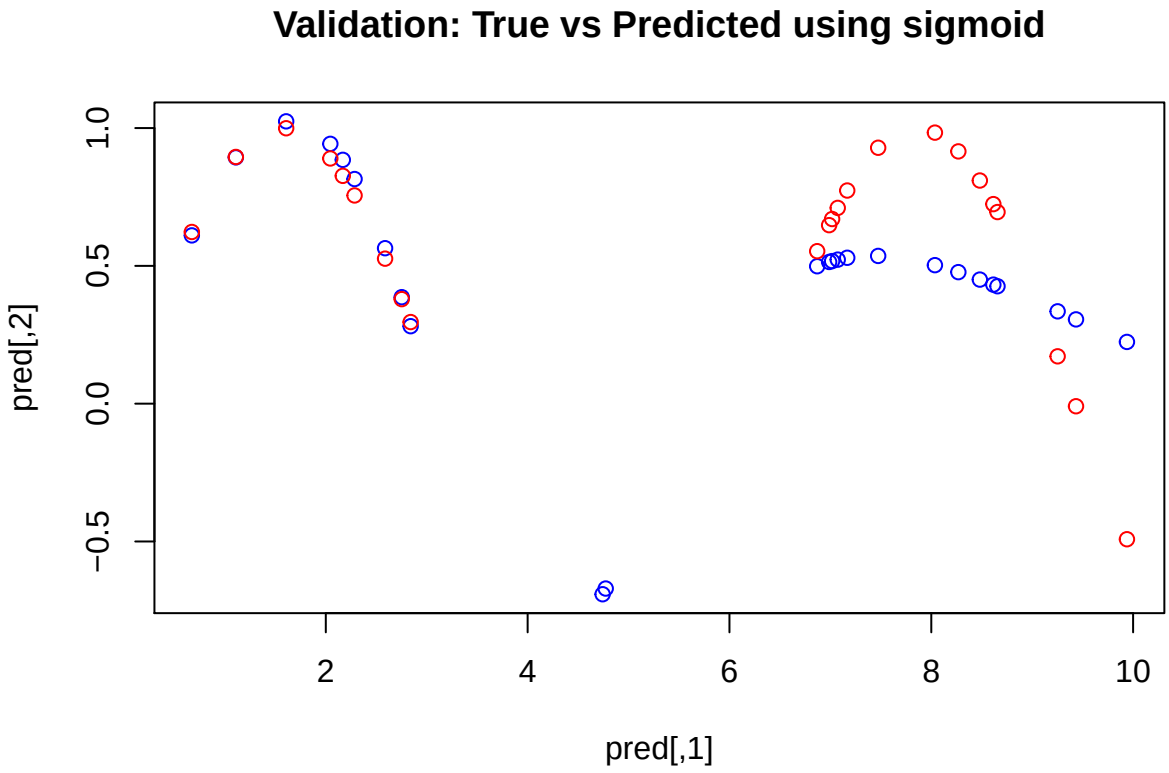
```
  pred <- cbind(va$Var,y_k)

  plot(pred,col="blue",main=paste0("Validation: True vs Predicted using ", a_function))

  points(va, col = "red")


}
```

```
## NN using Sigmoid activation function
Nnet("sigmoid")
```

```
## Converged Weights and Biases Using  sigmoid
## Input Weight:  0.629252 1.814546 -0.1810731 1.163563 0.1878194 -1.183294 1.799135 0.886882 0.597502 -
## Input Bias:  0.6264243 -5.865782 0.3065285 -1.560954 -0.4227426 5.739419 -0.9978777 -2.636683 0.7483
## Output Weight:  0.3252505 -4.34645 1.309957 -1.301127 -1.51287 -3.678236 2.464523 2.444056 0.2505412
## Output Bias:  0.8377446
```

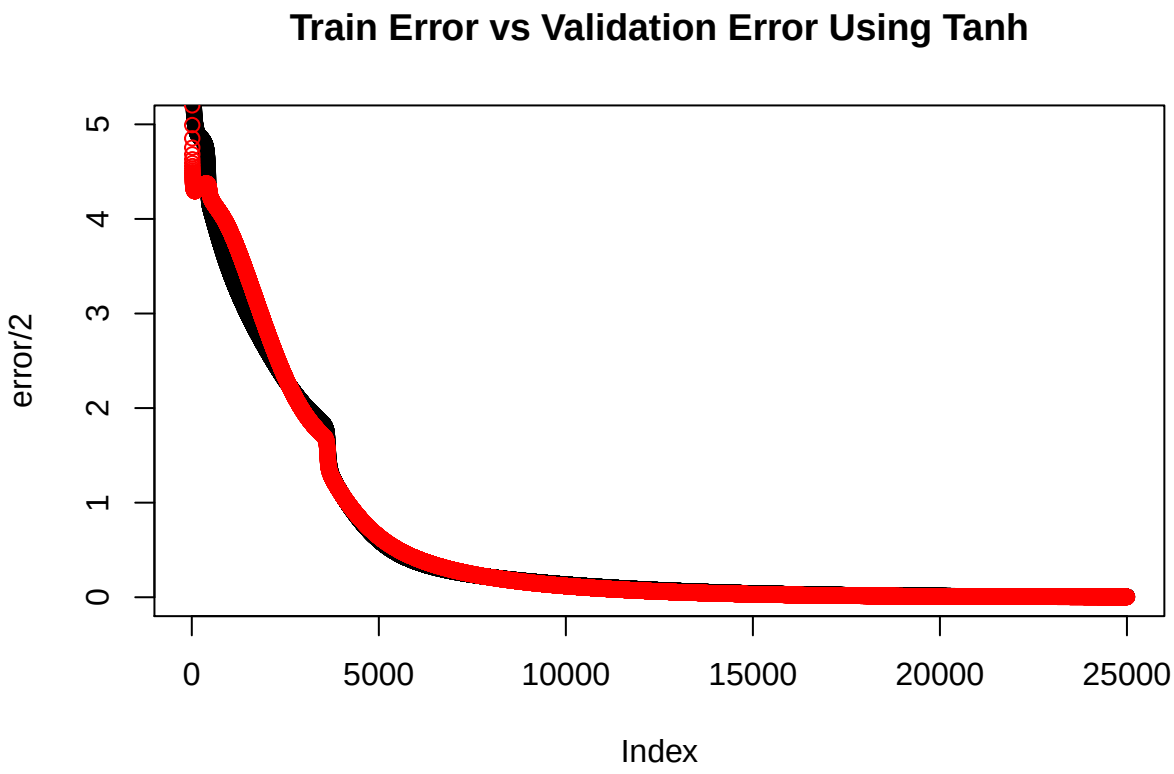


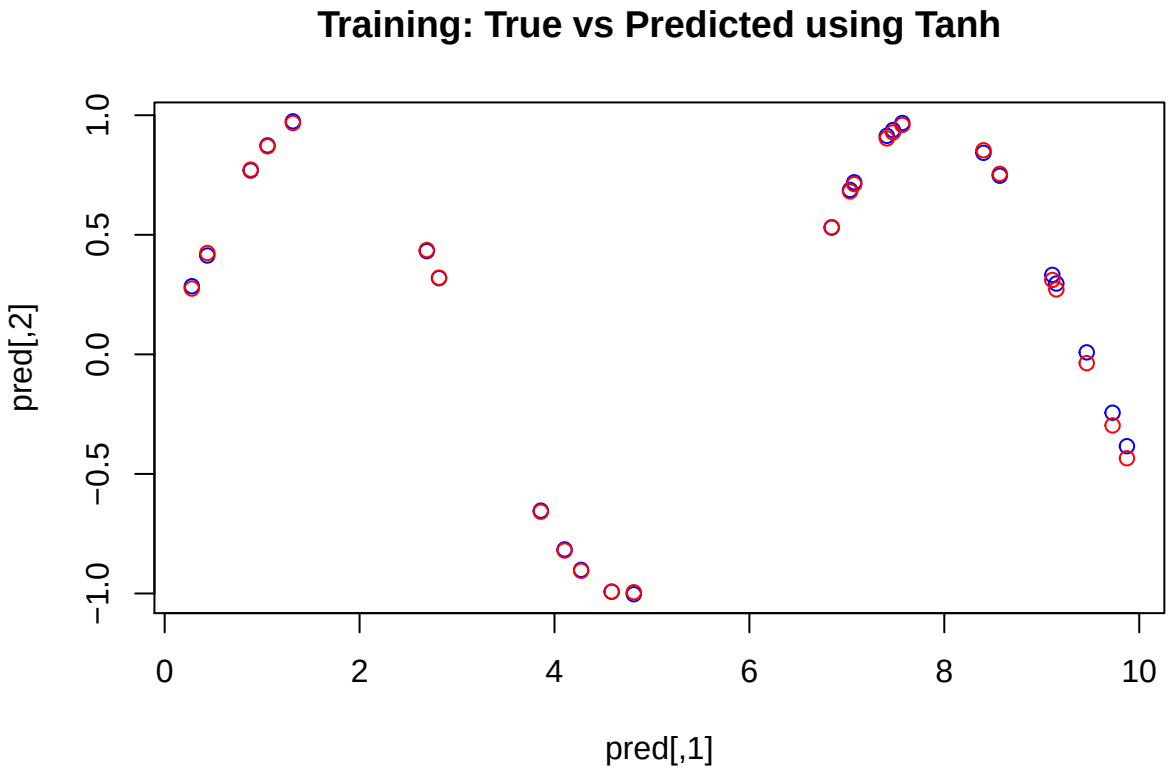

Training: True vs Predicted using sigmoid
pred[,2]
1.0
0.5
0.0
−0.5
0
2
4
6
8
10
pred[,1]

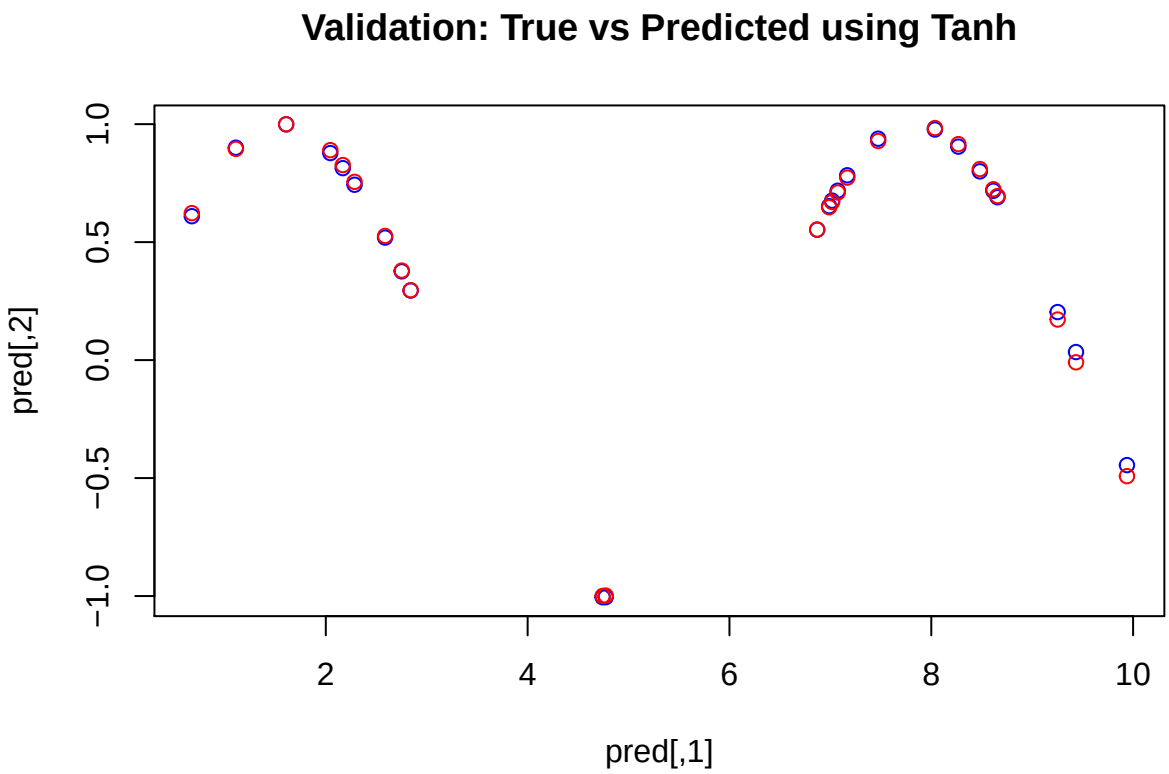



```
## NN using Tanh activation function
Nnet("Tanh")
```

```
## Converged Weights and Biases Using  Tanh
## Input Weight:  0.4541394 0.6569828 -0.5023898 0.3908682 0.7952875 -1.067047 1.102249 0.5989191 0.379
## Input Bias:  0.6370998 -2.154366 -0.4000872 -3.589061 0.6077995 0.8259108 -0.1052611 -3.889978 0.746
## Output Weight:  -0.2002968 -1.593744 0.3063093 -2.877745 -0.8697115 -0.696131 1.00758 2.575853 -0.30
## Output Bias:  0.3143672
```


Train Error vs Validation Error Using Tanh
error/2
0
1
2
3
4
5
0
5000
10000
15000
20000
25000
Index


Training: True vs Predicted using Tanh
pred[,2]
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10
pred[,1]



From the above graphs we could see that, neural network gives better result when we use $\tanh(a)$ as the activation function. This may be because, the tanh() function is having a stronger gradient as data is centered around zero hence derivatives are heigher. And also range of tanh() function is $[-1, 1]$ where as the range of $sigmoid()$ function is $[0, 1]$