```r
#### Custom Neuralnet - Iris Dataset
### Functions

## Returns the heaveside value
heaviside = function(z){
  return(ifelse(z>0, 1, 0))
}

### Setup
set.seed(12345)

# Ln function
nums = runif(50, 0.1, 10)
log = log(nums)
data.log = data.frame(x=nums, y=log)

# Heaviside function
nums = runif(50, -1, 1)
heav = heaviside(nums)
data.heav = data.frame(x=nums, y=heav)

# Quadratic function
nums = runif(50, -1, 1)
quad = nums^2
data.quad = data.frame(x=nums, y=quad)

# Absolute function
nums = runif(50, -5, 5)
abs = abs(nums)
data.abs = data.frame(x=nums, y=abs)

# Sine function
rad = runif(50, -pi, pi)
sin = sin(rad)
data.sin = data.frame(x=rad, y=sin)

# Data info
data = data.sin # feel free to change to other data set from above
n.data = dim(data)[1]
x.all = data$x
y.all = data$y

# Split into training and validation set
ids = sample(1:n.data, n.data/2)
train = data[ids,] # training
valid = data[-ids,] # validation
x.train = t(train$x)
x.valid = t(valid$x)
y.train = train$y
y.valid = valid$y

# init weights
w.j = runif(10, -1, 1)
b.j = runif(10, -1, 1)
```

```
w.k = runif(10, -1, 1)
b.k = runif(1, -1, 1)

n.train = dim(train)[1]
n.valid = dim(valid)[1]

learning.rate = 1/n.train #^2 # 1/25^2 which is in [1/n, 1/n^2]
n.iterations = 10000 # number of training iterations
error.train = numeric()
error.valid = numeric()
#error = numeric(n.iterations)

### Functions

### Implementation

for(n in 1:n.iterations){
  # error train
  z.j.train = tanh(w.j%*%x.train+b.j)
  y.k.train = w.k%*%z.j.train+b.k
  error.train[n] = 1/2*sum((y.k.train-y.train)^2)

  # error train, too slow
  #for(m in 1:n.train) {
  #  z.j <- tanh(w.j * train[m,]$rad + b.j)
  #  y.k <- sum(w.k * z.j) + b.k
  #  error[n] <- error[n] + (y.k - train[m,]$sin)^2
  #}
  # error[n] = error[n]/2

  # error validation
  z.j.valid = tanh(w.j%*%x.valid+b.j)
  y.k.valid = w.k%*%z.j.valid+b.k
  error.valid[n] = 1/2*sum((y.k.valid-y.valid)^2)

  cat("n: ", n, "| error.train: ", error.train[n], "| error.valid: ",
error.valid[n], "\n")
  flush.console()

  for(i in 1:n.train){
    # forward propagation
    z.j = tanh(w.j*x.train[i]+b.j)
    y.k = sum(w.k*z.j)+b.k

    # backward propagation
    d.k = y.k-y.train[i]
    d.j = (1-z.j^2)*w.k*d.k

    # update weigths and biases
    b.k = b.k - learning.rate*d.k
    w.k = w.k - learning.rate*d.k*z.j
    b.j = b.j - learning.rate*d.j
    w.j = w.j - learning.rate*d.j*x.train[i]
  }
}
```

```
plot(error.valid[-1], type="l", col=3)
lines(error.train[-1], type="l", col=1)
#lines(error[-1], type="l", col=5)

plot(x.all, y.all, col=rgb(0,0,1,0.5), pch=19,
     xlab="X", ylab="Y")
points(x.valid, y.k.valid, col=rgb(1,0,0,0.5), pch=19)
points(x.train, y.k.train, col=rgb(0,1,0,0.5), pch=19)
legend("bottomright", legend=c("Actual", "Validation", "Train"),
col=c(4, 2, 3), pch=19)
```