```
#### Custom Neuralnet - Iris Dataset

### Setup
set.seed(12345)

## Iris dataset
data.iris = as.data.frame(iris)
idx = data.iris$Species %in% c("virginica", "versicolor")
y.iris = ifelse(data.iris[idx,]$Species=="virginica", 1, 0)
X.iris = data.iris[idx,c(1,3)] # Select only Lengths to be able to
plot
n.iris = dim(X.iris)[1]
X.iris = as.matrix(X.iris / max(X.iris)) # Rescale for faster
convergence

# Data info
n.all = n.iris
X.all = X.iris
y.all = y.iris

# Split into training and validation set
ids = sample(1:n.all, n.all/2)
train = X.all[ids,] # training
valid = X.all[-ids,] # validation
X.train = as.matrix(X.all[ids,])
X.valid = as.matrix(X.all[-ids,])
y.train = y.all[ids]
y.valid = y.all[-ids]
n.train = dim(X.train)[1]
n.valid = dim(X.valid)[1]

# init weights
w.j = as.matrix(cbind(runif(2, -0.5, 0.5), runif(2, -0.5, 0.5)))
b.j = as.matrix(runif(2, -0.5, 0.5), ncol=1, nrow=2)
w.k = runif(2, -0.5, 0.5)
b.k = runif(1, -0.5, 0.5)

learning.rate = 6 #^2 # 1/25^2 which is in [1/n, 1/n^2]
n.iterations = 1000 # number of training iterations
error.train = numeric()
error.valid = numeric()

### Functions
sigmoid = function(z){
  return(1 / (1+exp(-z)))
}

### Implementation

for(n in 1:n.iterations){
  # error train
  z.j.train = sweep(X.train %*% t(w.j), 2, b.j, "+")
  a.k.train = z.j.train %*% w.k+b.k
  y.k.train = sigmoid(a.k.train) > 0.5
```

```
  error.train[n] = mean((y.k.train-y.train)^2)

  # error validation
  z.j.valid = sweep(X.valid %*% t(w.j), 2, b.j, "+")
  a.k.valid = z.j.valid %*% w.k+b.k
  y.k.valid = sigmoid(a.k.valid)
  error.valid[n] = mean((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)
    a.k = t(w.k) %*% z.j+b.k
    y.k = sigmoid(a.k)[1]

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

    # 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)
plot(error.train[-1], type="l", col=1)
#lines(error[-1], type="l", col=5)
yfit.train = ifelse(y.k.train>0.5, 1, 0)
plot(X.train[,1], X.train[,2], col=rgb(y.train,1-y.train,0,0.5),
pch=19,
     xlab="X", ylab="Y")
points(X.train[,1], X.train[,2], col=rgb(yfit.train,1-
yfit.train,0,0.5), pch=3)
```