Build a neural network

Mats Hansson

16 april 2016

How to bulid a Neural Network

In this paper we a bulding a neural network from scratch

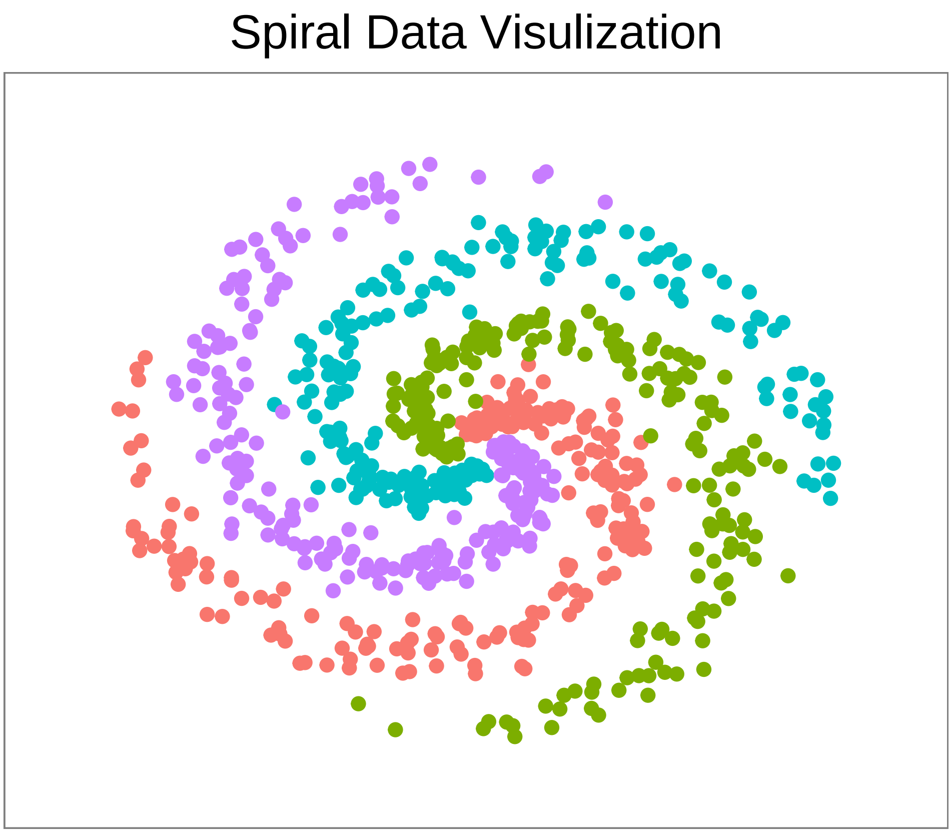
Create the data

```
library(ggplot2)
library(caret)
```

```
## Loading required package: lattice
```

```
N <- 200 # number of points per class
D <- 2 # dimensionality
K <- 4 # number of classes
X <- data.frame() # data matrix (each row = single example)
y <- data.frame() # class labels
set.seed(308)
for (j in (1:K)){
  r <- seq(0.05,1,length.out = N) # radius
  t <- seq((j-1)*4.7,j*4.7, length.out = N) + rnorm(N, sd = 0.3) # theta
  Xtemp <- data.frame(x =r*sin(t), y = r*cos(t))
  ytemp <- data.frame(matrix(j, N, 1))
  X <- rbind(X, Xtemp)
  y <- rbind(y, ytemp)}

data <- cbind(X,y)
colnames(data) <- c(colnames(X), 'label')
```

```
######################
x_min <- min(X[,1])-0.2
x_max <- max(X[,1])+0.2
y_min <- min(X[,2])-0.2
y_max <- max(X[,2])+0.2 # lets visualize the data:
ggplot(data) + geom_point(aes(x=x, y=y, color = as.character(label)),
                          size = 2) + theme_bw(base_size = 15) +
  xlim(x_min, x_max) + ylim(y_min, y_max) +
  ggtitle('Spiral Data Visulization') +
  coord_fixed(ratio = 0.8) +
  theme(axis.ticks=element_blank(),
         panel.grid.major = element_blank(),
         panel.grid.minor = element_blank(),
         axis.text=element_blank(),
         axis.title=element_blank(),
         legend.position = 'none')
```



Neural network conntrucrion

```
X <- as.matrix(X)
Y <- matrix(0, N*K, K)
for (i in 1:(N*K)){
 Y[i, y[i,]] <- 1}


nnet <- function(X, Y, step_size = 0.5, reg = 0.001, h = 10, niteration){  # get dim of input
  N <- nrow(X) # number of examples
  K <- ncol(Y) # number of classes
  D <- ncol(X) # dimensionality   # initialize parameters randomly
  W <- 0.01 * matrix(rnorm(D*h), nrow = D)
  b <- matrix(0, nrow = 1, ncol = h)
  W2 <- 0.01 * matrix(rnorm(h*K), nrow = h)
  b2 <- matrix(0, nrow = 1, ncol = K)    # gradient descent loop to update weight and bias
  for (i in 0:niteration)
  {    # hidden layer, ReLU activation
   hidden_layer <- pmax(0, X%*% W + matrix(rep(b,N), nrow = N, byrow = T))
   hidden_layer <- matrix(hidden_layer, nrow = N)     # class score
   scores <- hidden_layer%*%W2 + matrix(rep(b2,N), nrow = N, byrow = T)     # compute and normalize
   exp_scores <- exp(scores)
   probs <- exp_scores / rowSums(exp_scores)     # compute the loss: sofmax and regularization
```

```
      corect_logprobs <- -log(probs)
      data_loss <- sum(corect_logprobs*Y)/N
      reg_loss <- 0.5*reg*sum(W*W) + 0.5*reg*sum(W2*W2)
      loss <- data_loss + reg_loss    # check progress
      if (i%%1000 == 0 | i == niteration){
        print(paste("iteration", i,': loss', loss))}     # compute the gradient on scores
      dscores <- probs-Y
      dscores <- dscores/N     # backpropate the gradient to the parameters
      dW2 <- t(hidden_layer)%*%dscores
      db2 <- colSums(dscores)    # next backprop into hidden layer
      dhidden <- dscores%*%t(W2)    # backprop the ReLU non-linearity
      dhidden[hidden_layer <= 0] <- 0    # finally into W,b
      dW <- t(X)%*%dhidden
      db <- colSums(dhidden)     # add regularization gradient contribution
      dW2 <- dW2 + reg *W2
      dW <- dW + reg *W     # update parameter
      W <- W-step_size*dW
      b <- b-step_size*db
      W2 <- W2-step_size*dW2
      b2 <- b2-step_size*db2
    }

    return(list(W, b, W2, b2))}
```

Prediction function and model training

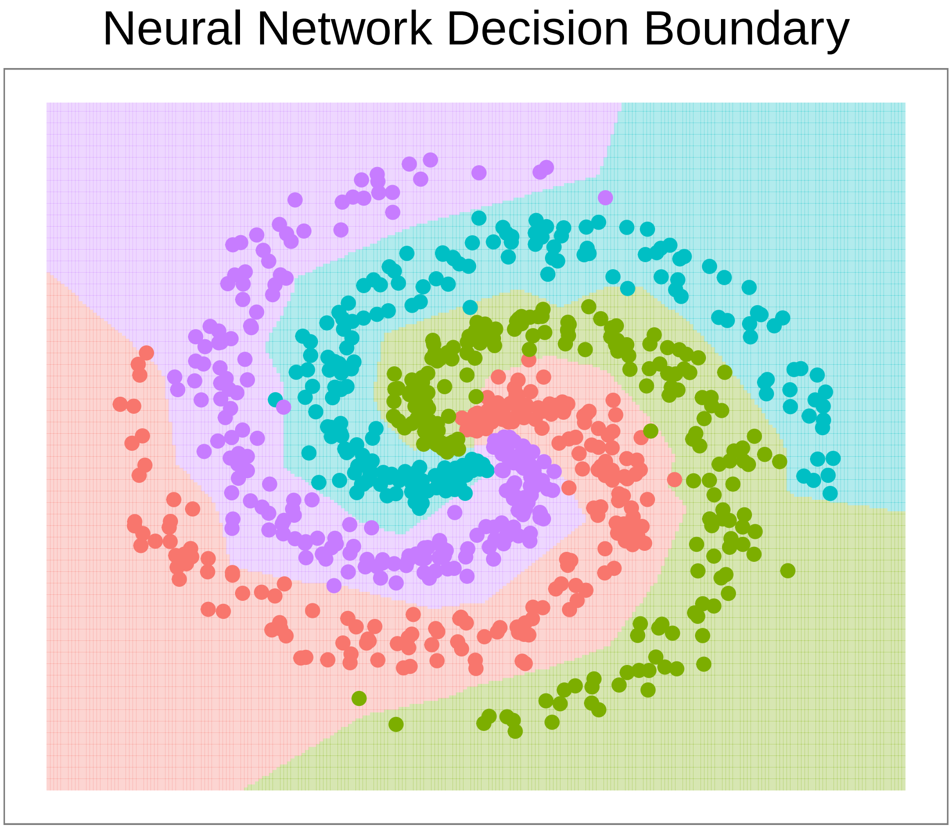
```
nnetPred <- function(X, para = list())
  {
  W <- para[[1]]
  b <- para[[2]]
  W2 <- para[[3]]
  b2 <- para[[4]]
  N <- nrow(X)
  hidden_layer <- pmax(0, X%*% W + matrix(rep(b,N), nrow = N, byrow = T))
  hidden_layer <- matrix(hidden_layer, nrow = N)
  scores <- hidden_layer%*%W2 + matrix(rep(b2,N), nrow = N, byrow = T)
  predicted_class <- apply(scores, 1, which.max)
  return(predicted_class)
  }

nnet.model <- nnet(X, Y, step_size = 0.4,reg = 0.0002, h=50, niteration = 6000)
```

```
## [1] "iteration 0 : loss 1.38628868932674"
## [1] "iteration 1000 : loss 0.967921639616882"
## [1] "iteration 2000 : loss 0.448881467342854"
## [1] "iteration 3000 : loss 0.293036646147359"
## [1] "iteration 4000 : loss 0.244380009480792"
## [1] "iteration 5000 : loss 0.225211501612035"
## [1] "iteration 6000 : loss 0.218468573259166"
```

```
predicted_class <- nnetPred(X, nnet.model)
print(paste('training accuracy:',mean(predicted_class == (y))))
```

```
## [1] "training accuracy: 0.96375"
```

```
hs <- 0.01
grid <-as.matrix(expand.grid(seq(x_min, x_max, by = hs),
                             seq(y_min, y_max, by =hs)))
Z <- nnetPred(grid, nnet.model)
ggplot()+  geom_tile(aes(x = grid[,1],
                         y = grid[,2],
                         fill=as.character(Z)),
                     alpha = 0.3, show.legend = F)+ geom_point(data = data,
                                                                 aes(x=x, y=y,
                                                                     color = as.character(label)),
                                                                 size = 2) + theme_bw(base_size = 15) +
```


Neural Network Decision Boundary