STAT 775: Machine Learning
HW 06

Terence Henriod

March 21, 2015

Abstract

In this assignment, feed-forwards neural networks using simple stochastic backpropagation were explored.

1 Exercise 01

1.1 Problem Statement

Implement a single neural network to perform digit image classification for all ten classes of digits using the zip.data data set from the ESL website. To reduce the model complexity, perform PCA to limit the input features to 20. Use a single hidden layer with varying numbers of units. Use 10 output units. Use sigmoidal units.

1.2 Results

A neural net with 20 inputs, 20 hidden layer units, and 10 outputs was constructed and trained. Using a learning rate of 2 and exposing the network to each training sample observation 500 times produces the following results, summarized in a confusion matrix:

Actual/Prediction	0	1	2	3	4	5	6	7	8	9	% Correct
0	341	1	2	2	3	3	6	0	1	0	94.9
1	0	254	0	0	3	2	4	0	0	1	96.2
2	5	0	174	4	7	1	2	2	3	0	87.9
3	3	0	3	139	0	18	0	1	2	0	83.7
4	1	2	7	0	181	2	1	1	1	4	90.5
5	3	1	1	11	3	138	0	0	2	1	86.3
6	6	0	2	0	1	3	155	0	3	0	91.2
7	0	0	0	1	5	0	0	138	1	2	93.9
0	2	2	3	9	2	6	2	2	131	7	78.9
0	0	2	1	1	8	0	0	4	3	158	89.3
Overall											90.13

For reference, the neural net did not outperform the naive bayes classification (with 16 principle components) that was done in a previous assignment, although adding more rounds of training or varying the number of nodes in the hidden layer might remedy this. Also, the training time was extraordinarily long, so other back-propagation techniques should be used to improve the training time.

1.3 Code

The following R code was used to perform the LDA and classification:

```
#
# Initial Setup
#
setwd("C:/Users/Terence/Documents/GitHub/STAT775/HW06")

#
# Data Cleaning
#
DATA.PATH <- "../DataSets/zip.data/"
ZIP.TRAIN.FILE.NAME <- paste0(DATA.PATH, "zip.train")
ZIP.TEST.FILE.NAME <- paste0(DATA.PATH, "zip.test")

get.multi.dimensional.label <- function(label) {
#
# Expands a single digit label to a {0, 1} vector-label
#
# Args:
#   label: a single numeric value [0-9]
```

```
  multi.dimensional.label <- matrix(0, nrow = 1, ncol = 10)
  multi.dimensional.label[1, as.numeric(label) + 1] <- 1
  return(multi.dimensional.label)
}

squash.multi.dimensional.label <- function(multi.dim.label) {
#
# Collapses a digit's {0, 1} vector-label to a single-value label
#
# Args:
#   mulit.dim.label: a 10 x 1 {0, 1} vector indicating the true class

  label <- -1
  highest.probability <- max(multi.dim.label)
  for (i in 1:nrow(multi.dim.label)) {
    if (multi.dim.label[[i]] == highest.probability) {
      label <- i - 1
    }
  }
  return(label)
}

read.data.tuples <- function(file.path.name) {
  data.fram.e <- read.table(file.path.name)

  data <- data.matrix(data.fram.e[, -1])

  targets <- matrix(nrow = nrow(data.fram.e), ncol = 10)
  for (i in 1:nrow(data.fram.e)) {
    targets[i, ] <- get.multi.dimensional.label(data.fram.e$V1[[i]])
  }

  data.tuple <- list(
    observations = data,
    labels = data.matrix(data.fram.e[, 1]),
    targets = targets
  )
  return(data.tuple)
}


#
# PCA
#

get.pca.summary <- function(data, num.components = 20) {
#
# Args:
#   data: an n x m matrix of n observations of m dimensions
#   num.components: the number of principle components to keep

  num.component.s <- min(ncol(data), num.components)
  n.obs <- nrow(data)
  full.dimensionality <- ncol(data)
```

```
  mu <- colMeans(data)

  # get centered data
  x <- data
  for (i in 1:n.obs) {
    x[i, ] <- data[i, ] - mu
  }

  # covariance matrix
  sigma <- t(x) %*% x
  sigma <- sigma * (1.0 / n.obs)

  eigen.decomposition <- eigen(sigma, F)  # TODO: is cov symmetric? Not sure...
  eigen.vectors <- eigen.decomposition$vectors[, 1:num.component.s]

  pca.summary <- list(
    rotation = eigen.vectors,
    mu = matrix(mu, nrow = 1, ncol = full.dimensionality)
  )

  return(pca.summary)
}

predict <- function(pca.summary, data) {
#
# Args:
#   data: an n x d matrix of observations; rows are observations
#   pca.summary: a tuple of the rotation matrix (eigenvectors as columns) and
#                the mean (row vector of column means) computed in the pca
#                computations

  x <- data
  for (i in 1:nrow(data)) {
    x[i, ] <- data[i, ] - pca.summary$mu
  }

  return (t(t(pca.summary$rotation) %*% t(x)))
}

#
# Neural Net
#

sigmoid <- function(x) {
#
# Args:
#   x: a numeric or vector; the function is applied element-wise

  return(1.0 / (1.0 + exp(-x)))
}

sigmoid.derivative <- function(x) {
  #
  # Args:
```

```
  #   x: a numeric or vector

  return(sigmoid(x) * (1.0 - sigmoid(x)))
}

construct.neural.net <- function(
  topology = c(2, 2, 1),
  activation = sigmoid,
  activation.derivative = sigmoid.derivative,
  debug = F) {
#
# Args:
#   topology: a list or vector of the dimensions of each layer
#   activation: a function to be used for the activation of each unit
#   activation.derivative: a function that is the derivative of activation

  layer.weights <- list()
  derivative.matrices <- list()
  outputs <- list()

  previous.layer.dim <- 1
  next.layer.dim <- 1
  for (i in 1:(length(topology) - 1)) {
    previous.layer.dim <- topology[[i]] + 1  # +1 for bias
    next.layer.dim <- topology[[i + 1]]
    num.elements <- (previous.layer.dim) * next.layer.dim

    layer.weights[[i]] <- matrix(
      if(debug) {rep(1, num.elements)}
      else {runif(n = num.elements, min = -0.001, max = 0.001)},
      nrow = previous.layer.dim,
      ncol = next.layer.dim
    )

    outputs[[i]] <- matrix(0, nrow = next.layer.dim, ncol = 1)
    derivative.matrices[[i]] <- diag(0, next.layer.dim)
  }

  return(list(
    input.dim = topology[[1]],
    output.dim = next.layer.dim,  # should be dim of last layer
    n.layers = length(layer.weights),
    activation = activation,
    activation.deriv = activation.derivative,
    input = matrix(0, nrow = 1, ncol = topology[[1]]),
    output = matrix(0, nrow = tail(topology, 1)[[1]], 1),
    weights = layer.weights,
    outputs = outputs,
    derivatives = derivative.matrices
  ))
}

apply.inputs <- function(net, x, for.training = T) {
#
```

```
# Args:
#   x: a 1 x n vector of inputs; n should be the same as for net
#   net: a structure with all of the appropriate data for a neural network,
#        as created by construct.neural.net()
#   for.training[T]: currently unused

  net$input <- matrix(x, nrow = 1)
  previous.output <- cbind(net$input, 1)
  for (i in 1:net$n.layers) {
    weighted.sums <- previous.output %*% net$weights[[i]]

    net$outputs[[i]] <- net$activation(weighted.sums)

    net$derivatives[[i]] <- diag(
      as.list(net$activation.deriv(weighted.sums)),
      length(net$outputs[[i]])
    )

    previous.output <- cbind(net$outputs[[i]], 1)
  }

  net$output <- t(tail(net$outputs, 1)[[1]])

  return(net)
}

backprop.weight.update <- function(net, target, learning.rate = 0.1) {
#
# Args:
#   net: a neural net object that has had inputs applied and derivatives stored
#   target: a column vector; the target output that should have been observed
#   learning.rate: the learning rate of the network
#                  TODO: refactor learning.rate to be less hacky, allow for
#                        advanced techniques

  last.index <- net$n.layers

  deltas <- list()
  error <- matrix(net$output, ncol = 1) - matrix(target, ncol = 1)
  deltas[[last.index + 1]] <- error
  W <- diag(1, nrow = nrow(error))
  D <- net$derivatives[[last.index]]
  for (i in last.index:1) {
    deltas[[i]] <- D %*% W %*% deltas[[i + 1]]

    if (i > 1) {
      D <- net$derivatives[[i - 1]]
      W <- net$weights[[i]]
      W <- W[1:(nrow(W) - 1), ]
    }
  }

  weight.updates <- list()
  o.hat <- cbind(net$input, 1)
```

```
  for (i in 1:last.index) {
    weight.updates[[i]] <- -learning.rate * t(deltas[[i]] %*% o.hat)

    if (i < last.index) {
      o.hat <- cbind(net$outputs[[i]], 1)
    }
  }

  for (i in 1:length(weight.updates)) {
    net$weights[[i]] <- net$weights[[i]] + weight.updates[[i]]
  }

  return(net)
}

#
# Main
#
NUM.PRINCIPLE.COMPONENTS <- 20
NUM.DIGITS <- 10
NUM.EPOCHS <- 500
CONFUSION.LABELS <-
  c(' 0 ', ' 1 ', ' 2 ', ' 3 ', ' 4 ', ' 5 ', ' 6 ', ' 7 ', ' 8 ', ' 9 ')

# training ###################################
train <- read.data.tuples(ZIP.TRAIN.FILE.NAME)

pca.summary <- get.pca.summary(
  data = train$observations,
  num.components = NUM.PRINCIPLE.COMPONENTS
)

train$observations <- predict(pca.summary, train$observations)

k <- 20
digit.net <- construct.neural.net(
  topology = c(NUM.PRINCIPLE.COMPONENTS, k, NUM.DIGITS),
  activation = sigmoid,
  activation.deriv = sigmoid.derivative
)

for (t in 1:NUM.EPOCHS) {
  for (i in 1:nrow(train$targets)) {
    digit.net <- apply.inputs(
      net = digit.net,
      x = matrix(train$observations[i, ], nrow = 1)
    )
    digit.net <- backprop.weight.update(
      net = digit.net,
      target = matrix(train$targets[i, ], ncol = 1),
      learning.rate = 2
    )
  }
}
```

```
# testing ###################################
test <- read.data.tuples(ZIP.TEST.FILE.NAME)
test$observations <- predict(pca.summary, test$observations)

num.correct <- 0
confusion.matrix <- matrix(0, nrow = NUM.DIGITS, ncol = NUM.DIGITS + 1)
row.names(confusion.matrix) <- CONFUSION.LABELS
colnames(confusion.matrix) <- c(CONFUSION.LABELS, '% correct')
for (i in 1:nrow(test$observations)) {
  prediction <- squash.multi.dimensional.label(
    apply.inputs(
      net = digit.net,
      x = matrix(test$observations[i, ], nrow = 1),
      for.training = F
    )$output
  )

  if (prediction == test$labels[[i]]) {
    num.correct <- num.correct + 1
  }

  confusion.matrix[test$labels[[i]] + 1, prediction + 1] <-
    confusion.matrix[test$labels[[i]] + 1, prediction + 1] + 1
}

class.totals <- rowSums(confusion.matrix)
for (i in 1:nrow(confusion.matrix)) {
  confusion.matrix[i, 11] <- 100 * confusion.matrix[i, i] / class.totals[[i]]
}

print(100 * num.correct / nrow(test$labels))
print(confusion.matrix)
```