

Question 1

Use own language to explain the following concepts:

Gradient Descent:

Popular machine learning optimization method (learning rule) which is used for updating a model's weights during the training process. Using Gradient Descent, weight updates are calculated using the negative of the squared error function's gradient. This error is calculated by passing training examples through network of linear units (no activation functions, therefore continuous) and calculating error as difference between expected outcome and predicted outcome. The negative derivative of the model's squared error will point downhill, and Gradient Descent will use the learning rate parameter to determine how far to step in the downhill direction. Gradient descent can update weights after calculating the training error for all training instances, or alternatives can update weights after a random subset of instances.

Neural Network learning rate:

A machine learning hyperparameter that determines the size of the step to take when updating weights. In neural networks, the learning rate is multiplied by the negative gradient of the network's squared error, and the result is used to update the weights of the network. If the learning rate is too small, learning will take longer, but if it is too high then it is possible that the network will oscillate or even diverge.

Multi-Layer Feed Forward Neural Network:

At least one hidden layer is required for a multi-layer network, a layer which sits between the input and output layers. With the exception of the last layer, each layer l_k must be connected to the next layer in the network l_{k+1} to satisfy requirement of feed forward network. In addition, feed forward networks can not contain loops or connections to previous layers. Therefore, multi-layer feed forward network is a neural network with at least one hidden layer and where every layer is connected to the following layer, with no loops to same layer or previous layers. Multi-layered networks solve a major problem encountered with single layer networks – they are able learn non-linearly separable data.

Hidden Nodes in Neural Network:

Hidden nodes are nodes of the network's hidden layer. Hidden nodes receive their input from an input layer or another hidden layer, and their output is fed into either another hidden layer or an output layer.

Output Nodes in Neural Network:

The final layer in the neural network. The number of output nodes will determine the size of the network's output vector, and is usually determined to be the total number of possible classes expected.

Backpropagation Rule:

The rule which is used to update weights in a multilayer network, it determines how to distribute weight adjustments throughout the network based on the error of the network and gradient descent. There are two steps – forward pass, and backward pass. The forward pass computes the output of all units in the network and calculates the network's error. Then the backward pass, starting with output layer and moving back through the network, recursively computes the local gradient of each neuron and uses this to update the network's weights.

Question 2

What is the gradient at point (2, 4) for equation $y = x^2$

The derivative of $y = x^2$ with respect to x is $2x$. At (2, 4), the gradient = $(2)(2) = 4$.

Following gradient descent, find the next movement towards the global minimum:

With a learning rate of 0.1, the next step will be the negative of the gradient multiplied by the learning rate = - 0.4

Question 3

Derive back propagation update rule for neural network with one hidden layer and one output layer, including BP rule for hidden and output nodes. Derivation included in below image:

ANN HW 4 (2)

③ Derive backpropagation weight update rules for neural net with one hidden layer.

We are interested in each neuron's local gradient. Recall that a given neuron's output $v_j = \sum_{i=0}^m w_{ij}x_{ij}$.

By chain rule, $\frac{\partial E(w)}{\partial w_{ij}} = \frac{\partial E(w)}{\partial v_j} \frac{\partial v_j}{\partial w_{ij}}$; Local Gradient of neuron j defined as $\delta_j = -\frac{\partial E(w)}{\partial v_j}$ ⑤

From $\frac{\partial v_j}{\partial w_{ij}} = \frac{\partial \sum w_{ij}x_{ij}}{\partial w_{ij}} = x_{ij}$; $\Delta w_{ij} = -n \frac{\partial E(w)}{\partial w_{ij}} = -n \frac{\partial E(w)}{\partial v_j} \frac{\partial v_j}{\partial w_{ij}}$

Therefore, we must derive δ_j for both hidden layer nodes and output layer nodes.

A) Output Neuron - By Chain Rule

- Error of jth neuron, $e_j = d_j - o_j$

$\frac{\partial e_j}{\partial o_j} = -1$ $\frac{\partial E(w)}{\partial e_j} = e_j$

$\delta_j = -\frac{\partial E(w)}{\partial v_j} = -\frac{\partial E(w)}{\partial e_j} \frac{\partial e_j}{\partial o_j} \frac{\partial o_j}{\partial v_j} = -e_j \psi'(v_j)$

$e_j = d_j - o_j$; $o_j = \psi(v_j) = \frac{1}{1 + e^{-v_j}}$

$\Delta w_{ij} = n \delta_j x_{ij} = n (d_j - o_j) \cdot a \cdot o_j (1 - o_j) \cdot x_{ij}$

B) Hidden Neurons $\delta_j = -\sum_{k \in C} \frac{\partial E(w)}{\partial v_k} \frac{\partial v_k}{\partial v_j}$ - v_k is output neuron
- C is set of output neurons

Will use chain rule to solve $\frac{\partial v_k}{\partial v_j} = \frac{\partial v_k}{\partial o_j} \frac{\partial o_j}{\partial v_j}$; $v_k = \sum_{j=0}^m w_{jk} o_j$

$\frac{\partial v_k}{\partial o_j} = w_{jk}$ and the output node v_k is derivative of sigmoid! $= a \cdot o_j \cdot (1 - o_j) = a \cdot o_j (1 - o_j)$

Combining components: $\delta_j = \psi'(v_j) \sum_{k \in C} \delta_k w_{jk} = a \cdot o_j \cdot (1 - o_j) \sum_{k \in C} \delta_k w_{jk}$

$\Delta w_{ij} = n \delta_j x_{ij} = n \cdot a \cdot x_{ij} \cdot o_j \cdot (1 - o_j) \sum_{k \in C} \delta_k w_{jk}$

Output Node: $\delta_k = a \cdot o_k \cdot (1 - o_k) \cdot (d_k - o_k)$

Hidden Node: $\delta_h = a \cdot o_h \cdot (1 - o_h) \cdot \sum_{k \in C} \delta_k w_{hk}$, where C is set of output nodes

Output Node: $\Delta w_{ij} = n (d_j - o_j) \cdot a \cdot o_j (1 - o_j) x_{ij}$

Hidden Node: $\Delta w_{ij} = n \cdot a \cdot x_{ij} \cdot o_j (1 - o_j) \sum_{k \in C} \delta_k w_{jk}$

Question 4

What is mean squared error of the network with respect to the 3 instances? Update the weights using instance I₁. The following image includes work and solution:

ANN HW4

i) Given Network

Input	Label
I ₁ <1, 1, 0.5>	1
I ₂ <1, 0, 1>	0
I ₃ <1, 0.5, 0.5>	1

ii) Forward Pass - calculate output of each node:

I ₁ : $V_a = (1 \times 1) + (1 \times 1) + (1 \times 0.5) = 2.5$ $O_a = \Phi(V_a) = 0.9933$	$V_b = 2.5$ same as a $O_b = 0.9933$	$V_c = (1 \times 1) + (1 \times 0.9933) + (1 \times 0.9933) = 2.987$ $O_c = \Phi(V_c) = 0.9975 = \text{actual output}$
I ₂ : $V_a = (1 \times 1) + (1 \times 0) + (1 \times 1) = 2.0$ $O_a = \Phi(V_a) = 0.9820$	$V_b = 2.0$ same as a $O_b = 0.9820$	$V_c = (1 \times 1) + (1 \times 0.9820) + (1 \times 0.9820) = 2.964$ $O_c = 0.9973 = \text{actual output}$
I ₃ : $V_a = (1 \times 1) + (1 \times 0.5) + (1 \times 0.5) = 2.0$ $O_a = \Phi(V_a) = 0.9820$	$V_b = 2.0$ same as a $O_b = 0.9820$	$V_c = (1 \times 1) + (1 \times 0.9820) + (1 \times 0.9820) = 2.964$ $O_c = 0.9973 = \text{actual output}$

iii) Calculate Mean Squared Error: $E(w) = \frac{1}{2N} \sum_n \sum_j (d_j(n) - o_j(n))^2$

$$E(w) = \frac{1}{2(3)} [(1 - 0.9975)^2 + (0 - 0.9973)^2 + (1 - 0.9973)^2] = \frac{0.9946}{6} = 0.166$$

Mean Sq. Error = 0.166

iv) Calculate updated weights after feeding just I₁ to network! $I_1 = <1, 1, 0.5>, 1$

Already have node outputs from ii) $O_a = 0.9933, O_b = 0.9933, O_c = 0.9975$

$W_{ij} = W_{ij} + \Delta W_{ij}; \Delta W_{ij} = \eta \delta_j X_{ij};$

Node a	Node b	Node c
$\delta_j = \text{hidden nodes}$ $\delta_j = a \cdot o_j (1 - o_j) \sum_k \delta_k w_{jk}$ $\delta_a = (2)(0.9933)(0.0067)(1.25e-05 * 1) = 1.66e-07$ $\delta_b = (2)(0.9933)(0.0067)(1.25e-05 * 1) = 1.66e-07$	$\delta_j = \text{output node:}$ $\delta_j = a \cdot o_j (1 - o_j) (d_j - o_j)$ $\delta_c = (2)(0.9975)(0.0025)(0.0025) = 1.25e-05$	$\Delta W_{oa} = (0.5)(1.66e-07)(1) = 8.3e-08$ $W_{oa} = 1 + 8.3e-08$ $\Delta W_{ia} = (0.5)(1.66e-07)(1) = 8.3e-08$ $W_{ia} = 1 + 8.3e-08$ $\Delta W_{ab} = (0.5)(1.66e-07)(1) = 8.3e-08$ $W_{ab} = 1 + 8.3e-08$ $\Delta W_{ob} = (0.5)(1.66e-07)(1) = 8.3e-08$ $W_{ob} = 1 + 8.3e-08$ $\Delta W_{ac} = (0.5)(1.66e-07)(0.5) = 4.15e-08$ $W_{ac} = 1 + 4.15e-08$ $\Delta W_{bc} = (0.5)(1.66e-07)(0.5) = 4.15e-08$ $W_{bc} = 1 + 4.15e-08$
$\Delta W_{oc} = (0.5)(1.25e-05)(1) = 6.25e-06$ $W_{oc} = 1 + 6.25e-06$ $\Delta W_{ac} = (0.5)(1.25e-05)(0.9933) = 6.21e-06$ $W_{ac} = 1 + 6.21e-06$ $\Delta W_{bc} = (0.5)(1.25e-05)(0.9933) = 6.21e-06$ $W_{bc} = 1 + 6.21e-06$	$\Delta W_{oa} = (0.5)(1.25e-05)(1) = 6.25e-06$ $W_{oa} = 1 + 6.25e-06$ $\Delta W_{ia} = (0.5)(1.25e-05)(0.9933) = 6.21e-06$ $W_{ia} = 1 + 6.21e-06$ $\Delta W_{ab} = (0.5)(1.25e-05)(0.9933) = 6.21e-06$ $W_{ab} = 1 + 6.21e-06$	$\Delta W_{ob} = (0.5)(1.25e-05)(0.9933) = 6.21e-06$ $W_{ob} = 1 + 6.21e-06$

Question 5

Derive back propagation update rule given the new squared error function which penalizes weights with high magnitudes:

ANN HW4

5) The squared error function is adjusted to penalize weights with large magnitude. Derive backprop weight update rule for output nodes.

$$\text{Given } E(w) = \frac{1}{2} \sum_{n=1}^N \sum_{j \in C} [d_j(n) - o_j(n)]^2 + r \sum_{i,j} w_{ij}^2$$

$$\varphi(v_j) = \frac{1}{1 + e^{-av_j}} ; v_j = \sum w_{ij} x_{ij}$$

$$\boxed{\Delta w_{ij} = -n \frac{\partial E(w)}{\partial w_{ij}}} , \text{ need to derive } \frac{\partial E(w)}{\partial w_{ij}}$$

$$\frac{\partial E(w)}{\partial w_{ij}} = \frac{\partial \frac{1}{2} \sum \sum [d_j(n) - o_j(n)]^2 + r \sum_{i,j} w_{ij}^2}{\partial w_{ij}}$$

$$= \frac{\partial \frac{1}{2} \sum \sum [d_j(n) - o_j(n)]^2}{\partial w_{ij}} + \frac{\partial r \sum w_{ij}^2}{\partial w_{ij}}$$

$$\text{From problem 3, we know } \frac{\partial \frac{1}{2} \sum \sum [d_j(n) - o_j(n)]^2}{\partial w_{ij}} = -a \cdot o_j(1-o_j)(d_j - o_j)x_{ij} \quad (1)$$

$$\text{Still need } \frac{\partial r \sum w_{ij}^2}{\partial w_{ij}} = r 2w_{ij} \quad (2)$$

$$\text{Combining (1) and (2)} \Rightarrow \boxed{\frac{\partial E(w)}{\partial w_{ij}} = -a \cdot o_j(1-o_j)(d_j - o_j)x_{ij} + r 2w_{ij}} \quad (3)$$

$$\Delta w_{ij} = -n \frac{\partial E(w)}{\partial w_{ij}} = n \cdot a \cdot o_j \cdot (1-o_j) \cdot (d_j - o_j) \cdot x_{ij} - 2rTw_{ij}$$

$$\boxed{\Delta w_{ij} = n(a \cdot o_j \cdot (1-o_j) \cdot (d_j - o_j) \cdot x_{ij} - 2rTw_{ij})}$$

Question 6

Assume RBF network using Gaussian RBF function. Use pseudo-inverse to calculate weight values of the output node, validate results with respect to the 4 instances provided:

ANN HW 4

6) Given the following input X & RBF
 $t_1 = (0.1, 0.1)$ $t_2 = (0.9, 0.9)$

Using Gaussian RBF, calc weights:

 $\varphi_1 = e^{-\|x - t_1\|^2}$
 $\varphi_2 = e^{-\|x - t_2\|^2}$

New Plot

Our data is now separable.

x_1	x_2	y	φ_1	φ_2	y
0	0	-1	$e^{-0.2}$	$e^{-1.62}$	-1
1	0	1	$e^{-0.2}$	$e^{-1.62}$	1
0	1	1	$e^{-0.2}$	$e^{-1.62}$	1
1	1	-1	$e^{-0.2}$	$e^{-1.62}$	-1

$x_1: (0,0) \rightarrow \varphi_1 = e^{-\sqrt{0.1^2 + 0.1^2}} = e^{-0.2} \approx 0.818$

$\varphi_2 = e^{-\sqrt{0.9^2 + 0.9^2}} = e^{-1.62} \approx 0.197$

$x_2: (1,0) \rightarrow \varphi_1 = e^{-\sqrt{0.1^2 + 0.9^2}} = e^{-0.91} \approx 0.403$

$\varphi_2 = e^{-\sqrt{0.9^2 + 0.9^2}} = e^{-1.62} \approx 0.197$

$x_3: (0,1) \rightarrow \varphi_1 = e^{-\sqrt{0.1^2 + 0.9^2}} = e^{-0.91} \approx 0.403$

$\varphi_2 = e^{-\sqrt{0.9^2 + 0.9^2}} = e^{-1.62} \approx 0.197$

$x_4: (1,1) \rightarrow \varphi_1 = e^{-\sqrt{0.1^2 + 0.9^2}} = e^{-0.2}$

$\varphi_2 = e^{-\sqrt{0.9^2 + 0.9^2}} = e^{-1.62} \approx 0.197$

$\Phi = \begin{bmatrix} 1 & 0.818 & 0.197 \\ 1 & 0.403 & 0.403 \\ 1 & 0.403 & 0.403 \\ 1 & 0.197 & 0.818 \end{bmatrix}$, then $\underline{\Phi}^{-1} = \begin{bmatrix} 1.928 & 2.428 & 2.428 & -1.928 \\ 3.197 & -2.392 & -2.392 & 1.587 \\ 1.587 & -2.392 & -2.392 & 3.197 \end{bmatrix}$

$\underline{\Phi} [w_0, w_1, w_2]^T = [d_1, d_2, d_3, d_4]^T \Rightarrow [w_0, w_1, w_2]^T = \underline{\Phi}^{-1} [d_1, d_2, d_3, d_4]^T$

Weights = $\underline{\Phi}^{-1} [-1 \ 1 \ 1 \ -1]^T = \begin{bmatrix} 8.712 \\ -9.568 \\ -9.568 \end{bmatrix} \begin{array}{l} w_0 \\ w_1 \\ w_2 \end{array}$

using sign function with threshold of 0 b/c labels
 acc -1 & 1

$y = \begin{cases} 1 & \text{if } v \geq 0 \\ -1 & \text{otherwise} \end{cases}$

$w_0 = 8.712$

$\sum w_i \varphi_i = V$

$\cancel{x_1: (8.712 \times 1) + (-9.568 \times 0.818) + (-9.568 \times 0.197) = -1, < 0 \rightarrow -1}$

$\cancel{x_2: (8.712 \times 1) + (-9.568 \times 0.403) + (-9.568 \times 0.403) = 0.999, > 0 \rightarrow 1}$

$x_3: \text{ same as } x_2 \rightarrow 0.999 > 0 \rightarrow 1$

$x_4: (8.712 \times 1) + (-9.568 \times 0.197) + (-9.568 \times 0.818) = -1.001 < 0 \rightarrow -1$

The new weights are $w_0 = 8.712$, $w_1 = -9.568$, $w_2 = -9.568$, they produced correct output when validated against input.

Question 7

Design and implement the following face recognition task using Neural Networks:

DL 100 face images from CMU to build two class classification tasks. Must specify faces in positive or negative classes (left vs right). Include all faces in final submission.

Face images were downloaded from CMU (<http://www.cs.cmu.edu/~tom/faces.html>) in low resolution format. For this model, we are only interested in two classes of face images: heads turned left (positive) and heads turned right (negative). The original data set images are organized by person, and contain images with heads facing left, right, up, and down. Left and right facing images were combined to create two new directories, each appropriately containing only positive (left) or negative (right) images.

```
SB-FL-46ZTHD6:Desktop justin.johnson$ cp faces_4/
.DS_Store an2i/ boland/ chaf/ choon/ glickman/ kawamura/ megak/ night/ saavik/ sz24/
Positive/ at33/ bpm/ cheyer/ dameln/ karyadi/ kk49/ mitchell/ phoebe/ steffi/ tammo/
SB-FL-46ZTHD6:Desktop justin.johnson$ cp faces_4/*/*left*.pgm FaceData/Positive/
SB-FL-46ZTHD6:Desktop justin.johnson$ cp faces_4/*/*right*.pgm FaceData/Negative/
SB-FL-46ZTHD6:Desktop justin.johnson$ ls FaceData/Positive/
an2i_left_angry_open_4.pgm cheyer_left_angry_open_4.pgm kawamura_left_angry_sunglasses_4.pgm phoebe_left_happy_open_4.pgm
an2i_left_angry_sunglasses_4.pgm cheyer_left_angry_sunglasses_4.pgm kawamura_left_happy_open_4.pgm phoebe_left_happy_sunglasses_4.pgm
an2i_left_happy_open_4.pgm cheyer_left_happy_open_4.pgm kawamura_left_happy_sunglasses_4.pgm phoebe_left_neutral_open_4.pgm
an2i_left_happy_sunglasses_4.pgm cheyer_left_happy_sunglasses_4.pgm kawamura_left_neutral_open_4.pgm phoebe_left_neutral_sunglasses_4.pgm
an2i_left_neutral_open_4.pgm cheyer_left_neutral_open_4.pgm kawamura_left_neutral_sunglasses_4.pgm phoebe_left_sad_open_4.pgm
an2i_left_neutral_sunglasses_4.pgm cheyer_left_neutral_sunglasses_4.pgm kawamura_left_sad_open_4.pgm phoebe_left_sad_sunglasses_4.pgm
an2i_left_sad_open_4.pgm cheyer_left_sad_open_4.pgm kawamura_left_sad_sunglasses_4.pgm saavik_left_angry_open_4.pgm
an2i_left_sad_sunglasses_4.pgm cheyer_left_sad_sunglasses_4.pgm kk49_left_angry_open_4.pgm saavik_left_angry_sunglasses_4.pgm
```

The above image displays the splitting of data into positive/negative classes. A preview of the positive file names are also displayed. There are 157 positive images and 155 negative images.

Two images, one positive and one negative, were loaded into R and plotted:



Left image is positive (head turned left) and right image is negative (head turned right). Images are low resolution (32 x 30) to reduce computation cost.

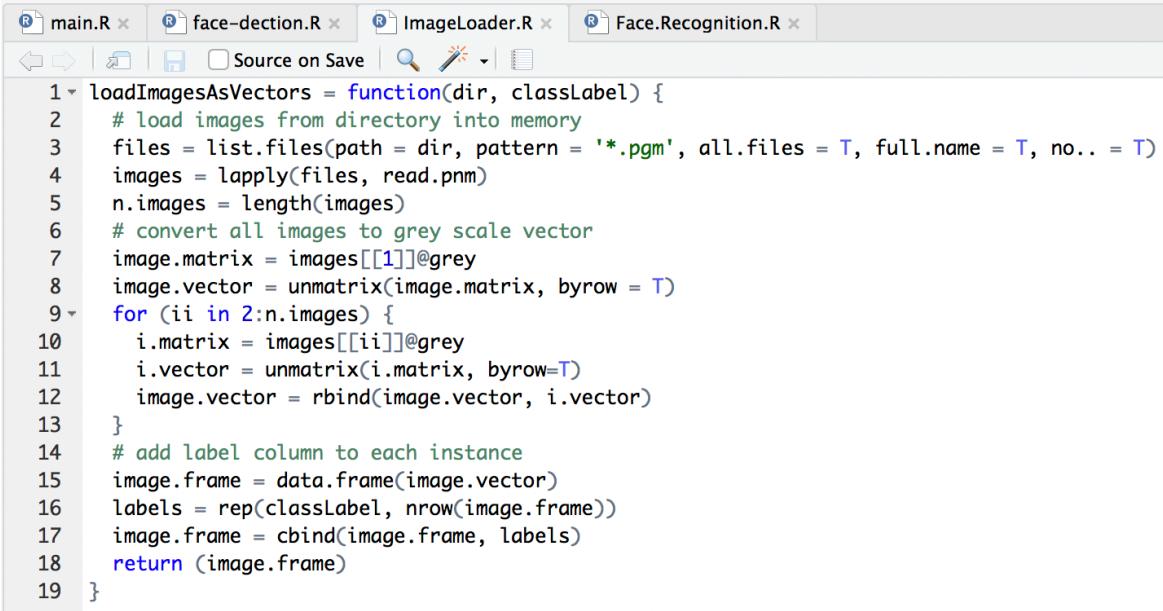
Below image contains the code required to read and plot images:

```

1 # Homework 4
2 # Problem 7
3
4 # load dependencies
5 library(pixmap)
6 library(gdata)
7 library(neuralnet)
8 source('ImageLoader.R')
9
10
11 # define data path
12 positiveDir = '../FaceData/Positive'
13 negativeDir = '../FaceData/Negative'
14
15
16 # plot 1 positive and 1 negative sample, to preview
17 posFile = list.files(positiveDir)[1]
18 posFile = read.pnm(paste(positiveDir, posFile, sep='/'))
19 plot(posFile)
20 negFile = list.files(negativeDir)[1]
21 negFile = read.pnm(paste(negativeDir, negFile, sep='/'))
22 plot(negFile)
23

```

A helper function was created to load images from file into memory, convert each file to a matrix of grey scale pixel intensities, flatten matrix to a vector, and add a column for class label. The help function then takes all image instances (as vectors) and row binds them, producing a matrix where each row is an image and each column is a pixel intensity value. This helper function is based off the one presented in lecture.



The screenshot shows the RStudio interface with multiple tabs open at the top: 'main.R', 'face-dection.R', 'ImageLoader.R', and 'Face.Recognition.R'. The 'ImageLoader.R' tab is currently active, displaying the following R code:

```

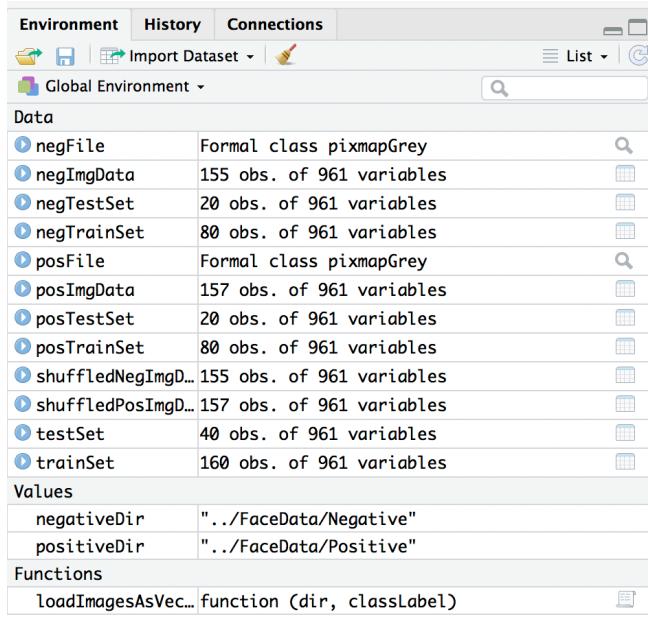
1 loadImagesAsVectors = function(dir, classLabel) {
2   # load images from directory into memory
3   files = list.files(path = dir, pattern = '*.pgm', all.files = T, full.name = T, no.. = T)
4   images = lapply(files, read.pnm)
5   n.images = length(images)
6   # convert all images to grey scale vector
7   image.matrix = images[[1]]@grey
8   image.vector = unmatrix(image.matrix, byrow = T)
9   for (ii in 2:n.images) {
10     i.matrix = images[[ii]]@grey
11     i.vector = unmatrix(i.matrix, byrow=T)
12     image.vector = rbind(image.vector, i.vector)
13   }
14   # add label column to each instance
15   image.frame = data.frame(image.vector)
16   labels = rep(classLabel, nrow(image.frame))
17   image.frame = cbind(image.frame, labels)
18   return (image.frame)
19 }

```

The below screenshot displays the code required to load all images as vectors and randomly partition into train and test data sets. Using a balanced data set of 200 images, 160 will be used for training and 40 will be used for evaluation.

```
25 # load image data as vectors
26 # each row is instance
27 # each column is grey intensity for given pixel
28 posImgData = loadImagesAsVectors('../FaceData/Positive', 1)
29 negImgData = loadImagesAsVectors('../FaceData/Negative', 0)
30
31
32 # we will use 100 positive and 100 negative instances for train/test sets
33 # will split the data 80/20 for training and testing
34 shuffledPosImgData = posImgData[sample(1:nrow(posImgData)), ]
35 shuffledNegImgData = negImgData[sample(1:nrow(negImgData)), ]
36 posTrainSet = shuffledPosImgData[1: 80, ]
37 negTrainSet = shuffledNegImgData[1: 80, ]
38 posTestSet = shuffledPosImgData[81: 100, ]
39 negTestSet = shuffledNegImgData[81: 100, ]
40
41 trainSet = rbind(posTrainSet, negTrainSet)
42 testSet = rbind(posTestSet, negTestSet)
43
```

I've included a screenshot of the R Studio workspace, allowing easy confirmation of data set that has been loaded into memory:



The screenshot shows the R Studio interface with the 'Environment' tab selected. The 'Global Environment' pane displays a list of objects and their details. The objects listed are:

Object	Type	Dimensions
negFile	Formal class pixmapGrey	
negImgData	155 obs. of 961 variables	
negTestSet	20 obs. of 961 variables	
negTrainSet	80 obs. of 961 variables	
posFile	Formal class pixmapGrey	
posImgData	157 obs. of 961 variables	
posTestSet	20 obs. of 961 variables	
posTrainSet	80 obs. of 961 variables	
shuffledNegImgD...	155 obs. of 961 variables	
shuffledPosImgD...	157 obs. of 961 variables	
testSet	40 obs. of 961 variables	
trainSet	160 obs. of 961 variables	

Below the data section, there are sections for 'Values' and 'Functions'. The 'Values' section contains two entries: 'negativeDir' and 'positiveDir', both with the value "'../FaceData/Negative'" and "'../FaceData/Positive'" respectively. The 'Functions' section contains one entry: 'loadImagesAsVec...', which is a function defined with parameters 'dir' and 'classLabel'.

Our training set contains 160 instances, each with 961 variables. The first 960 variables are the pixel intensities, and the last is the instance label. Similarly, our test set contains 40 instances, each with 961 variables.

Next the R neuralnet package is used to train 5 different multi-layer feed forward neural network architectures using the training set. All neural networks will include 960 input nodes, that is one node for each pixel of the image. They will also include 1 output node, allowing for binary classification of images as positive (head turned left) or negative (head turned right). Just one hidden layer will be used, and the total number of hidden nodes will be varied between 3 and 11.

```
44 # specify formula (a symbolic description of the model to be fitted)
45 n = names(trainSet)
46 myform = as.formula(paste('labels ~ ', paste(n[!n %in% 'labels'], collapse = ' + ')))
47
48 # mark the label index
49 labelIdx = length(trainSet)
50
51 # we don't need to add bias, the neuralnet library will do for us
52
53 # 1st network - 3 hidden nodes
54 face.3.classifier <- neuralnet(myform, trainSet, hidden=3, rep=500, linear.output=FALSE, threshold=0.1)
55 face.3.prediction<-compute(face.3.classifier, testSet[,-labelIdx])
56 face.3.results = face.3.prediction$net.result
57 face.3.results = lapply(face.3.results, round)
58 face.3.results = unlist(face.3.results)
59
60 # 2nd network - 5 hidden nodes
61 face.5.classifier <- neuralnet(myform, trainSet, hidden=5, rep=500, linear.output=FALSE, threshold=0.1)
62 face.5.prediction<-compute(face.5.classifier, testSet[,-labelIdx])
63 face.5.results = face.5.prediction$net.result
64 face.5.results = lapply(face.5.results, round)
65 face.5.results = unlist(face.5.results)
66
67 # 3rd network - 7 hidden nodes
68 face.7.classifier <- neuralnet(myform, trainSet, hidden=7, rep=500, linear.output=FALSE, threshold=0.1)
69 face.7.prediction<-compute(face.7.classifier, testSet[,-labelIdx])
70 face.7.results = face.7.prediction$net.result
71 face.7.results = lapply(face.7.results, round)
72 face.7.results = unlist(face.7.results)
73
74
75 # 4th network - 9 hidden nodes
76 face.9.classifier <- neuralnet(myform, trainSet, hidden=9, rep=500, linear.output=FALSE, threshold=0.1)
77 face.9.prediction<-compute(face.9.classifier, testSet[,-labelIdx])
78 face.9.results = face.9.prediction$net.result
79 face.9.results = lapply(face.9.results, round)
80 face.9.results = unlist(face.9.results)
81
82
83 # 5th network - 11 hidden nodes
84 face.11.classifier <- neuralnet(myform, trainSet, hidden=11, rep=500, linear.output=FALSE, threshold=0.1)
85 face.11.prediction<-compute(face.11.classifier, testSet[,-labelIdx])
86 face.11.results = face.11.prediction$net.result
87 face.11.results = lapply(face.11.results, round)
88 face.11.results = unlist(face.11.results)
```

Once training is complete, the 5 networks are evaluated using the test set. The output of the network is a value in the range (0, 1), as produced by the output node's activation function. For each network, this result is converted to a 0 or 1 by applying the round function to each result. This is the equivalent to a function which returns 1 if the probability of being equal to 1 is greater than 0.5. These results can now be compared to the test set's labels to generate a confusion matrix.

```
88
89 # generate confusion amtrixes for each network
90 face.3.confMatrix = table(testSet$labels, face.3.results)
91 face.5.confMatrix = table(testSet$labels, face.5.results)
92 face.7.confMatrix = table(testSet$labels, face.7.results)
93 face.9.confMatrix = table(testSet$labels, face.9.results)
94 face.11.confMatrix = table(testSet$labels, face.11.results)
95
```

The following table compares the results from each network:

Network #	Hidden Nodes	TP	FP	TN	FN	Accuracy	Error
1	3	20	0	20	0	100%	0%
2	5	20	0	20	0	100%	0%
3	7	20	0	20	0	100%	0%
4	9	20	0	20	0	100%	0%
5	11	20	2	18	0	95%	5%

When trained with 160 images and tested with a new set of 40 images, the model's performed shockingly well. Additional iterations were run, and since the images are sampled at random, the results varied with each iteration as expected. In general, all models performed very well with 100% accuracy. In several cases, networks with 9 or 11 hidden nodes yielded higher errors on the test set. It is likely that if test and training error were visualized, overfitting would be observed on the networks with more hidden nodes, therefore causing errors on the test set.