CS-331: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

ASSIGNMENT 4: Genetic Algorithms and Neural Networks

Instructor: Mian Muhammad Awais

TAs: Habiba Farrukh, Aiza Anjum Chaudhry, Fatima Tariq, Hamza Mahmood and Maryam Khalid

Submission Deadline: Monday 23rd November, 2015

As before, this assignment can be done in groups of 2 (same groups as last time). The evaluation of the assignment will be on individual basis. Members of the same group will get marks depending on the working of tasks they have implemented and their performance in viva. So make sure you contribute equally in the assignment.

Also, each member will be required to know what the other has done. One of the aims of these assignments is to ensure that you get the concepts as they are practically applied. With groups, you can divide tasks but in the end you need to know all that has been asked in the assignment.

All the submitted codes will be tested against each other and the solutions present online for plagiarism and in case of slightest doubt the concerned case will be forwarded to the Disciplinary Committee.

As always please reach out to the course staff for any help regarding the assignment and its configuration.

Part 1: Genetic Algorithms (40 Points)

Requirements for this part:
● MATLAB

Genetic Algorithms are widely used in image processing and compression. One application of genetic algorithms in image compression is recreating images using a fixed number of semitransparent polygons (refer to the following link: http://rogeralsing.com/2008/12/07/genetic-programming-evolution-of-mona-lisa/).

This task is a simplified version of the project, re-creating images using individual pixels instead of semi-transparent overlapping polygons. This simplification helps you in taking the assumption that the color at a particular position in an image is purely based on the RGB value of the pixel at that position (and not the resulting color of semitransparent overlapping polygons).

For this task you are required to implement a version of genetic algorithm which takes a target image (Mona Lisa for example) and some random images and tries to reach an approximate looking version of the target image from the random images.
Note: All random images should have the same dimension as the target image. You can upload these images from your computer or generate them in MATLAB.

Hints and Assumptions:

• For a given image of n by m pixels, treat the image as a long array of pixels of length n*m. The array can then be treated as a DNA string while the individual pixels can be treated as genes on DNA string.
• The initial population/number of DNA strings is fixed and will remain the same in each generation.
• A fitness function chooses a small fraction of actual population of DNA strings to move to next generation.
• CrossOver can only happen between same indexed pixels of two images.
• Mutation on a pixel can be caused by randomly generating new RGB values for that pixel.
• The process of crossover, mutation and fittest member selection continues until at least one member of the population becomes an approximate version of the target image.

Note: Make sure you add a pause in each loop in code. If your code is taking too long to finish, try approximating an image with a smaller dimension size 25 x 25 pixels, or reducing the threshold of the final approximation. For your implementation, provide the following details at the end of your code in comments
• Initial population
• Fraction of population moving to next generation
• Fitness function used
• Probability of cross-over

• Probability of mutation

Submit the answers to the following questions in answers_[RollNo1_RollNo2].txt file.
a. What is the effect of restricting crossover on only the first half of the DNA strings.
b. Does the time taken to reach the approximated image decrease with increasing

1) Probability of crossover
2) Probability of mutation

You also need to submit 10 images, showing how you reached the approximated version of the target image from the random images. For this you can choose the fittest image after every 100th generation (depending on how many generations your code takes). Name the images showing which generation they belong to.

Part 2: Neural Networks (60 Points)

Requirements For This Assignment

- MATLAB with the Neural Networks toolbox installed
- A smartphone or tablet for generating samples of your handwriting

Note: You may use online sketching tools but they might not be very accurate

Introduction

In this assignment you are required to come up with a neural network which recognizes handwritten Urdu alphabets. You will train the neural network using your own data sets. Testing will be done by using other data sets.

خ xē ख़े	ح baṙī ḥē बड़ी हे	چ čē चे	ج jīm जीम	ث s̱ē से	ٹ ṫē टे	ت tē ते	پ pē पे	ب bē बे	ا alif अलिफ़
ص ṣuād सुआद	ش šīn शीन	س sīn सीन	ژ žē झ़े	ز zē ज़े	ڑ ṙē ड़े	ر rē रे	ذ z̲āl ज़ाल	ڈ ḋāl डाल	د dāl दाल
ل lām लाम	گ gāf गाफ़	ک kāf काफ़	ق qāf क़ाफ़	ف fē फ़े	غ ġǣn ग़ैन	ع 'ǣn ऐन	ظ ẓōē ज़ोए	ط ṭōē तोए	ض źuād ज़ुआद
		ے baṙī yē बड़ी ये	ی čẖōṫī yē छोटी ये	ء hamzah हमज़ा	ھ dō-čašmī ẖē दोचश्मी हे	ہ čẖōṫī hē छोटी हे	و wāō वाओ	ن nūn नून	م mīm मीम

Figure 1: Alphabets to train and test your network on

Steps:

1) Generate a training set for the Urdu alphabets using a smartphone, tablet or a desktop/online application. You can check out the following links:
   http://drawisland.com/
   http://sketchtoy.com/
   Make sure that one image has only one character and that each image size is 150 by 150. For more accuracy, make sure that the letters are thin and dots are thick. From the first link (for the sketching application), this means that letters should be written with size 2px and dots with at least 10px. Try to make your letters as clear as possible. You will be required to upload this data and you will be graded on its quality: Letters can be recognized without ambiguity when we look at them.
   For each character, submit 3 training images and 1 test image.
2) Load each image in matlab. Loading multiple images in one go: (optional)
   http://stackoverflow.com/questions/15655177/loadalltheimagesfromadirectory
3) Binarize the image using the function im2bw. Set the threshold value to 0.4. You can also try out other values.
4) Get frame of the character in the image. This essentially produces another image with the character positioned at the center. However, the size of the image changes.

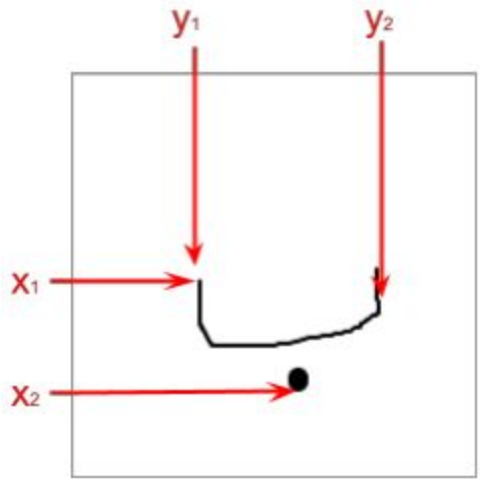

Figure 2: Coordinates for the character frame

5) Resize the image using imresize. Specify the number of rows and columns to be 30. You can play around with these values too. Just make sure that every image is resized to the same number of rows and columns for one network.
6) Convert the image to a 1 by 900 matrix from a 30 by 30 matrix using reshape.
7) Concatenate each matrix vertically to produce a matrix with 38 (each representing the urdu character) columns and 900 rows. For this, iteratively use the matlab function horzcat.
8) Produce a 38 by 38 identity matrix using the function eye(n), where n is the number of rows and columns in the matrix.
9) Now you are ready to create neural networks for character recognition. Use the following link for help:
   http://www.mathworks.com/help/nnet/examples/character-recognition.html

10) The above link explains how to train a network with a set of images and test the network using the same images with some added noise. Plot the performance of your trained networks.
11) Test your network with other sample images and check if your network can identify them correctly. Print the input alphabet and corresponding output from the network.

Important Things To Consider

- Use a minimum of 3 drawings for each alphabet.
- Consider a smaller number of alphabets if it is taking a long time to train your network. For example, start with 10 alphabets.
- You can also save/load your neural networks after training them, so you do not have to go through the whole process again if you make any changes in the rest of your code.
- Naming of your sample images matters. MATLAB uploads multiple images from a folder in an order. You need to upload a complete set of alphabets together in lexicographical order, followed by the next set, as many times as the number of training images you have for each set.
- Remember to use image sizes of 150 x 150, no larger. You can use a tablet or smartphone to help draw the images.
- Try out different thickness and sizes of the characters and use `imshow` at different steps to observe how the image preprocessing part affects the characters. For example, if you make your characters too thin, image resizing might cause characters to become disconnected.
- If you increase the size of the images for neural network training, your results will be better, but the training time will increase.
- The number of hidden layers of neurons in your network depend on the number of alphabets you are training the network for. If you're training for 38 characters, it is recommended that you set 37 hidden layer neurons. You can check out performance for different number of neurons as well.
- You can output using numbers to denote the alphabet number or print out the name of the alphabet in Roman Urdu if you do not have the Urdu font installed.

Along with your code files, you need to submit:

- 3 different images for every alphabet that you used to train your network
- 1-2 different images for every alphabet that you used to test your network
- a .jpg/.png file containing a graph that compares percentage errors in networks trained with/without noise

Appendix: Code Snippet for Pre-processing

```
char1 = imread('urduchar1.png');
%  Binarize image
char1BW = im2bw(char1, 0.4);
%  figure,
%  imshow(char1BW)        %show image

%  *** CENTER AND RESIZE CHARACTER ***
%find all co-ordinates for the character in the image
[I,J] = find(char1BW==0);
%plot the character to see if the coordinates are correct
%  figure,
%  plot(J, I)
%find x1, x2, y1 and y2 (as given in Figure 2 of the
Assignment
Statement)
x1 = 1;
if(min(I)-1 > 1)
     x1 = min(I)-1;
end

x2 = 1;
if(min(I)+1 < length(I))
     x2 = max(I)+1;
end

y1 = 1;
if(min(J)-1 > 1)
     y1 = min(J)-1;
end

y2 = 1;
if(min(J)+1 < length(J))
     y2 = max(J)+1;
end

%calculate average difference between length and width
meanDiff = floor(abs(((x2 - x1) - (y2 - y1))/2));
```

```
%check whether the character has greater length or width

if(x2 - x1 > y2 - y1) %if width is greater
      %trim the extra image
      char1BW([1:x1, x2:end],:)=[];
      %add average difference to length and trim the extra
      image char1BW(:,[1:(y1-meanDiff), (y2+meanDiff):end])=[];
elseif(x2 - x1 < y2 - y1) %if length is greater
      %add average difference to width and trim the extra
      image char1BW([1:(x1-meanDiff), (x2+meanDiff):end],:)=[];
      char1BW(:,[1:y1, y2:end])=[];
else %if both are equal
      %trim the extra image
      char1BW([1:x1, x2:end],:)=[];
      char1BW(:,[1:y1, y2:end])=[];
end

imshow(char1BW); %check image again

%resize image and check results again
char1Resize = imresize(char1BW,[30
30]);

imshow(char1Resize);

%convert to 1 by 900 array (from a 30 by 30 array)
sizeChar1 = size(char1Resize);
char1Row = reshape(char1Resize',1,sizeChar1(1)*sizeChar1(2));
%   length(char1Row)       %check lengths for all arrays (they
should all be equal)
%   length(char2Row)
%   length(char3Row)
      .
      .
      .

%   vertically concatenate all 38 matrices to form the following
matrix
X = ...horzcat(horzcat(char1Row', char2Row'), char3Row')...); %38 by
900 matrix
T = eye(38); %38 by 38 identity matrix
%   Your data is ready for training/testing
```