Optimising Model Constraints – Report

**Part 1 -** Find constraints that maximise the amount of images accurately represented by the model

To achieve this goal, I first decided to try to logically analyse the constraints myself to ensure that all strokes were included in the background 17 x 17 square. I put the minimum angle for theta1 to pi/2, the maximum angle for theta2 to pi, the max coordinate for x and y at 17 and the max length of 23 each. This ensured all strokes would be contained in this square but only got me an accuracy in the 30-40% range.

When iterating through to find the accuracy of my constraints, I knew that I wouldn’t have the time to iterate through all 1000 digits in order to find out so I decided to set it up so that it ran 100 of the 1000 digits using the imfit function and then returned the chimin value of each digit to a list. This was done with the following code:

*list = foreach (i=1:100) %dopar%*

*{*

*z = matrix(d1[,i],16,16)*

*res = imfit2(z)*

*chimin = res[11]*

*return (chimin)*

*}*

After this I wasn’t sure what to do and I just messed around with different combinations and widths but I still had accuracy levels getting 30-40 results out of 100. After an hour or two, it then hit me that the best thing to do was to iterate through all of the original digit images in order to find out the pattern that most of the images follow. I understood that the imfit function randomly generates strokes and then picks the strokes with the best accuracy to form the model so I knew the more that i restricted the area that these lines could be drawn in, the better the estimator could be. I iterated through the images using the following code:

*dev.set(3)*

*par(mfrow = c(1,1))*

*for(i in c(500:1000))*

*{*

*z = matrix(d1[,i],16,16)*

*image(c(1:16),c(1:16),z[,16:1],col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))*

*readline()*

*}*

From analysing these images, I devised the angle constraints, length constraints, width constraints and the x and y coordinate constraints. These constraints are as follows:

0.55 \* pi < theta1[i] < pi\* 0.8

theta1[i]+0.17\*pi < theta2[i] < pi\*1.06

10 < x1[i] < 17

10 < y1[i] < 17

10 < x2[i] < 17 And x1[i]

10 < y2[i] < 17 And y1[i]

10 < len1[i] < 22

2 < width1[i] < 5

10 < len1[i] < 18

2 < width1[i] < 57

I saw that the individual strokes stay within a certain angle range and are never shorter than length 10. I was also able to narrow down the starting points to a 7 x 7 square where the coordinated are greater than x = 10 and y = 10. The width was a bit harder as some strokes were very thick on the original but despite this, i decided to keep my constraints reasonably small.

In order to see individual comparisons between my model and the original image, I created a loop to iterate output both of these side by side so i could compare them by eye in order to improve my constraints. The code that I used to do this is below:

*dev.set(2)*

*par(mfrow=c(1,2))*

*for(i in c(1:1000))*

*{*

*z = matrix(d1[,i],16,16)*

*image(c(1:16),c(1:16),z[,16:1],col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))*

*res2 = imfit2(z)*

*s1=stroke(res2[1],res2[2],res2[3],res2[4],res2[5])*

*s2=stroke(res2[6],res2[7],res2[8],res2[9],res2[10])*

*im = s1+s2*

*im[im > 1] = 1*

*image(c(1:16),c(1:16),256\*im,col=gray(c(0:256)/256))*

*readline()*

*}*

The final constraints above got me an accuracy of between 63% and 68% when I ran it over different samples of 100 digits from the 1000. I feel this model is good in comparison to the 50% model that Alistair had in the class with his constraints.

When I ran this over the 1000 digits, I got XXX of them correct.

**Part 2** - Implement Simulated Annealing and see if this improves the number of good fits

To implement simulated annealing in my code, I changed the code where chimin was set in the imfit function to:

t = 1000/i

if (chi[i] - chimin < 0)

{

chimin = chi[i]

imin = i

}

else

{

if(exp(-(chi[i]-chimin)/t) > runif(1))

{

chimin = chi[i]

imin = i

}

}

This allowed me to have cooling rate “t” and for this rate to go from very large towards 0 as the loop iterates and “i” increases. This cooling rate had a small effect on the output bumping its accuracy up to the 65%- 70% range. This was a nice small change but I wanted to see how other cooling rates would affect it.

So I decided to change the cooling rate to:

*t = 10000 \* (0.85^(i))*

This rate also starts big and converged to 0 and had a better effect on results. It took the range to as high as 74% which was a really good improvement.

As a result of these findings, I concluded that my model was already a good model but that it was still getting caught in those local minima which was helped along using simulated annealing. While there wasn’t massive improvements, when you get into improving those last few tough digits to model, every improvement that you get is a bonus. It was cool to see how the different cooling rates affect the models performance and I was interested to see how I could improve on this rate yet again.

The full code that I used throughout this project is below:

#Read in digits

d1=scan("Zl1d.dat",nlines=1000,n=256000)

d1=matrix(d1,256,1000)

#Create Stroke function

stroke = function(x1,y1,theta,len,width)

{

x=c(1:16)

y=c(1:16)

x=x-x1

y=y-y1

u1=x\*cos(theta)

u2=y\*sin(theta)

#u=ones %\*% t(u1) + t(ones %\*% t(u2))

u=matrix(u1,16,16)+t(matrix(u2,16,16))

v1=-x\*sin(theta)

v2=y\*cos(theta)

#v=(ones %\*% t(v1) + t(ones %\*% t(v2)))

v=matrix(v1,16,16)+t(matrix(v2,16,16))

m = (v > 0 & v < len)

exp(-u\*u/width)\*m

}

# draw one stroke

s = stroke(16.5,16.5,pi\*0.75,22,3)

image(c(1:16),c(1:16),256\*s,col=gray(c(0:256)/256))

# draw 2 strokes

s1 = stroke(15,15,pi\*0.85,15,5)

s2 = stroke(15,15,pi\*1,15,5)

im=s1+s2

image(c(1:16),c(1:16),256\*im,col=gray(c(0:256)/256))

#select particular image

z = matrix(d1[,2],16,16)

#Get image of z

image(c(1:16),c(1:16),z[,16:1],col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))

#Draw strokes

s1=stroke(14,14,0.55\*pi,15,2)

s2=stroke(13,13,1.0\*pi,15,3)

im=s1+s2

im[im > 1] = 1

# show image of 1 stroke

image(c(1:16),c(1:16),256\*s1,col=gray(c(0:256)/256))

# show image of 2 strokes

image(c(1:16),c(1:16),256\*im,col=gray(c(0:256)/256))

#Define distance function

dist = function(a,b)

{

sum((a-b)^2)

}

imfit = function(z)

{

theta1=0

theta2=0

x1=0

y1=0

x2=0

y2=0

chi=0

len1=0

width1=0

len2=0

width2=0

theta1[1] = 0.55\*pi

theta2[1] = 0.8\*pi

x1[1] = 15

y1[1] = 15

x2[1] = 15

y2[1] = 15

len1[1] = 15

width1[1] = 2.0

len2[1] = 15

width2[1] = 2.0

imin=1

chimin=10000000

for(i in c(1:10000))

{

theta1[i] = theta1[imin] + 0.5\*(runif(1)-0.5)

theta2[i] = theta2[imin] + 0.5\*(runif(1)-0.5)

x1[i] = x1[imin] + 5.0\*(runif(1)-0.5)

y1[i] = y1[imin] + 5.0\*(runif(1)-0.5)

x2[i] = x2[imin] + 5.0\*(runif(1)-0.5)

y2[i] = y2[imin] + 5.0\*(runif(1)-0.5)

len1[i] = len1[imin] + 5\*(runif(1)-0.5)

width1[i] = max(width1[imin] + 7.0\*(runif(1)-0.5),0)

len2[i] = len2[imin] + 5\*(runif(1)-0.5)

width2[i] = max(width2[imin] + 7.0\*(runif(1)-0.5),0)

s1=stroke(x1[i],y1[i],theta1[i],len1[i],width1[i])

s2=stroke(x2[i],y2[i],theta2[i],len2[i],width2[i])

im=s1+s2

im[im > 1] = 1

chi[i] = dist(256\*im,z[,16:1])

if(chi[i] < chimin)

{

chimin = chi[i]

imin =i

}

}#end of for loop

c(x1[imin],y1[imin],theta1[imin],len1[imin],width1[imin],x2[imin],y2[imin],theta2[imin],len2[imin],width2[imin])

}

imfit2 = function(z)

{

theta1=0

theta2=0

x1=0

y1=0

x2=0

y2=0

chi=0

len1=0

width1=0

len2=0

width2=0

theta1[1] = 0.55\*pi

theta2[1] = 0.8\*pi

x1[1] = 15

y1[1] = 15

x2[1] = 15

y2[1] = 15

len1[1] = 15

width1[1] = 2.0

len2[1] = 15

width2[1] = 2.0

imin=1

chimin=10000000

for(i in c(1:10000))

{

theta1[i] = min(max(theta1[imin] + 0.5\*(runif(1)-0.5), 0.55 \* pi), pi\* 0.8)

theta2[i] = min(max(theta2[imin] + 0.5\*(runif(1)-0.5),theta1[i]+0.17\*pi, 0.8\*pi),pi\*1.06)

x1[i] = max(min(17,x1[imin] + 5.0\*(runif(1)-0.5)),10)

y1[i] = max(min(17,y1[imin] + 5.0\*(runif(1)-0.5)),10)

x2[i] = max(min(17,x2[imin] + 5.0\*(runif(1)-0.5)),10, x1[i])

y2[i] = max(min(17,y2[imin] + 5.0\*(runif(1)-0.5)),10, y1[i])

len1[i] = min(max(10.0,len1[imin] + 5\*(runif(1)-0.5)),22.0)

width1[i] = min(max(width1[imin] + 7.0\*(runif(1)-0.5),0.5),5.0)

len2[i] = min(max(10.0,len2[imin] + 5\*(runif(1)-0.5)), 18.0)

width2[i] = min(max(width2[imin] + 7.0\*(runif(1)-0.5),0.5),7.0)

s1=stroke(x1[i],y1[i],theta1[i],len1[i],width1[i])

s2=stroke(x2[i],y2[i],theta2[i],len2[i],width2[i])

im=s1+s2

im[im > 1] = 1

chi[i] = dist(256\*im,z[,16:1])

t = 10000/(1 + (0.85\*i))

if (chi[i] - chimin < 0)

{

chimin = chi[i]

imin = i

}

else

{

if(exp(-(chi[i]-chimin)/t) > runif(1))

{

chimin = chi[i]

imin = i

}

}

}#end of for loop

c(x1[imin],y1[imin],theta1[imin],len1[imin],width1[imin],x2[imin],y2[imin],theta2[imin],len2[imin],width2[imin],chimin)

}

# -----COMPARE SPECIFIC IMAGE-------

# iterate through the images

dev.set(3)

par(mfrow = c(1,1))

for(i in c(500:1000))

{

z = matrix(d1[,i],16,16)

image(c(1:16),c(1:16),z[,16:1],col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))

readline()

}

# set up to show 2 images

dev.set(2)

par(mfrow=c(1,2))

# select image

j = 5

z = matrix(d1[,j],16,16)

# get the results of the old fitting function

#res1 = imfit(z)

#res1

# get the results of the new fitting function

res2 = imfit2(z)

res2

# output the original image

image(c(1:16),c(1:16),z[,16:1],col=gray(c(0:256)/256))

# create the new fitted image

s1=stroke(res2[1],res2[2],res2[3],res2[4],res2[5])

s2=stroke(res2[6],res2[7],res2[8],res2[9],res2[10])

im = s1+s2

im[im > 1] = 1

# output the fitted image

image(c(1:16),c(1:16),256\*im,col=gray(c(0:256)/256))

# iterate through side by side images

dev.set(2)

par(mfrow=c(1,2))

for(i in c(1:1000))

{

z = matrix(d1[,i],16,16)

image(c(1:16),c(1:16),z[,16:1],col=gray(c(0:256)/256),xlab="",ylab="",mar=c(1,1,1,1))

res2 = imfit2(z)

s1=stroke(res2[1],res2[2],res2[3],res2[4],res2[5])

s2=stroke(res2[6],res2[7],res2[8],res2[9],res2[10])

im = s1+s2

im[im > 1] = 1

image(c(1:16),c(1:16),256\*im,col=gray(c(0:256)/256))

readline()

}

------------------------------------------------------------------

# set up dual core

library(foreach)

library(iterators)

library(doParallel)

numcores=detectCores()

registerDoParallel(numcores)

# iterate through digits

list = foreach (i=1:100) %dopar%

{

z=matrix(d1[,i],16,16)

res=imfit2(z)

chimin = res[11]

return (chimin)

}

# show list of chimins

list

# count the number of digits that are well modeled

num = length(which(list[1:100] < 600000))

num