*“Today you are You, that is truer than true. There is no one alive who is Youer than You.”*  
Dr. Seuss

*“Explanations exist; they have existed for all time; there is always a well-known solution to every human problem — neat, plausible, and wrong.”*  
H L Mencken

**Introduction**

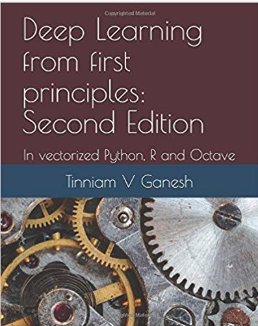
In this 6th instalment of ‘Deep Learning from first principles in Python, R and Octave-Part6’, I look at a couple of different initialization techniques used in Deep Learning, L2 regularization and the ‘dropout’ method. Specifically, I implement “He initialization” & “Xavier Initialization”. My earlier posts in this series of Deep Learning included

1. [Part 1](https://gigadom.wordpress.com/2018/01/04/deep-learning-from-basic-principles-in-python-r-and-octave-part-1/) – In the 1st part, I implemented logistic regression as a simple 2 layer Neural Network  
2. [Part 2](https://gigadom.wordpress.com/2018/01/11/deep-learning-from-first-principles-in-python-r-and-octave-part-2/) – In part 2, implemented the most basic of Neural Networks, with just 1 hidden layer, and any number of activation units in that hidden layer. The implementation was in vectorized Python, R and Octave  
3. [Part 3](https://gigadom.wordpress.com/2018/01/30/deep-learning-from-first-principles-in-python-r-and-octave-part-3/) -In part 3, I derive the equations and also implement a L-Layer Deep Learning network with either the relu, tanh or sigmoid activation function in Python, R and Octave. The output activation unit was a sigmoid function for logistic classification  
4. [Part 4](https://gigadom.wordpress.com/2018/02/26/deep-learning-from-first-principles-in-python-r-and-octave-part-4/) – This part looks at multi-class classification, and I derive the Jacobian of a Softmax function and implement a simple problem to perform multi-class classification.  
5. [Part 5](https://gigadom.wordpress.com/2018/03/23/deep-learning-from-first-principles-in-python-r-and-octave-part-5/) – In the 5th part, I extend the L-Layer Deep Learning network implemented in Part 3, to include the Softmax classification. I also use this L-layer implementation to classify MNIST handwritten digits with Python, R and Octave.

The code in Python, R and Octave are identical, and just take into account some of the minor idiosyncrasies of the individual language. In this post, I implement different initialization techniques (random, He, Xavier), L2 regularization and finally dropout. Hence my generic L-Layer Deep Learning network includes these additional enhancements for enabling/disabling initialization methods, regularization or dropout in the algorithm. It already included sigmoid & softmax output activation for binary and multi-class classification, besides allowing relu, tanh and sigmoid activation for hidden units.

A video presentation of regularization and initialization techniques can be also be viewed in [Neural Networks 6](https://www.youtube.com/watch?v=3Xf_5p6VkPU)

This R Markdown file and the code for Python, R and Octave can be cloned/downloaded from **Github** at [DeepLearning-Part6](https://github.com/tvganesh/DeepLearning-Part6)

Checkout my book ‘Deep Learning from first principles: Second Edition – In vectorized Python, R and Octave’. My book starts with the implementation of a simple 2-layer Neural Network and works its way to a generic L-Layer Deep Learning Network, with all the bells and whistles. The derivations have been discussed in detail. The code has been extensively commented and included in its entirety in the Appendix sections. My book is available on Amazon as [paperback](https://www.amazon.com/dp/1791596177)($18.99) and in [kindle version](https://www.amazon.com/dp/B07LBG542L)($9.99/Rs449).

You may also like my companion book “Practical Machine Learning with R and Python:Second Edition- Machine Learning in stereo” available in Amazon in [paperback](https://www.amazon.com/dp/1983035661)($10.99) and [Kindle](https://www.amazon.com/dp/B07DFKSCWZ)($7.99/Rs449) versions. This book is ideal for a quick reference of the various ML functions and associated measurements in both R and Python which are essential to delve deep into Deep Learning.

**1. Initialization techniques**

The usual initialization technique is to generate Gaussian or uniform random numbers and multiply it by a small value like 0.01. Two techniques which are used to speed up convergence is the [He](https://www.cv-foundation.org/openaccess/content_iccv_2015/papers/He_Delving_Deep_into_ICCV_2015_paper.pdf) initialization or [Xavier](http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf). These initialization techniques enable gradient descent to converge faster.

**1.1 a Default initialization – Python**

This technique just initializes the weights to small random values based on Gaussian or uniform distribution

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

import sklearn.linear\_model

import pandas as pd

import sklearn

import sklearn.datasets

exec(open("DLfunctions61.py").read())

#Load the data

train\_X, train\_Y, test\_X, test\_Y = load\_dataset()

# Set the layers dimensions

layersDimensions = [2,7,1]

# Train a deep learning network with random initialization

parameters = L\_Layer\_DeepModel(train\_X, train\_Y, layersDimensions, hiddenActivationFunc='relu', outputActivationFunc="sigmoid",learningRate = 0.6, num\_iterations = 9000, initType="default", print\_cost = True,figure="fig1.png")

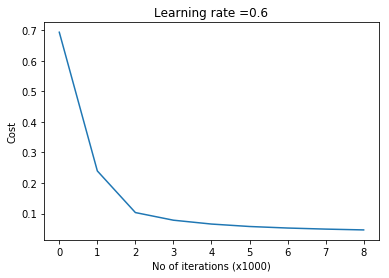
# Clear the plot

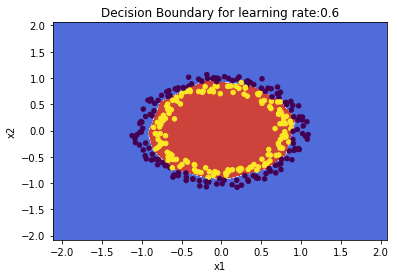
plt.clf()

plt.close()

# Plot the decision boundary

plot\_decision\_boundary(lambda x: predict(parameters, x.T), train\_X, train\_Y,str(0.6),figure1="fig2.png")





**1.1 b He initialization – Python**

‘He’ initialization attributed to [He et al](https://www.cv-foundation.org/openaccess/content_iccv_2015/papers/He_Delving_Deep_into_ICCV_2015_paper.pdf), multiplies the random weights by  
\sqrt{\frac{2}{dimension\ of\ previous\ layer}}

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

import sklearn.linear\_model

import pandas as pd

import sklearn

import sklearn.datasets

exec(open("DLfunctions61.py").read())

#Load the data

train\_X, train\_Y, test\_X, test\_Y = load\_dataset()

# Set the layers dimensions

layersDimensions = [2,7,1]

# Train a deep learning network with He initialization

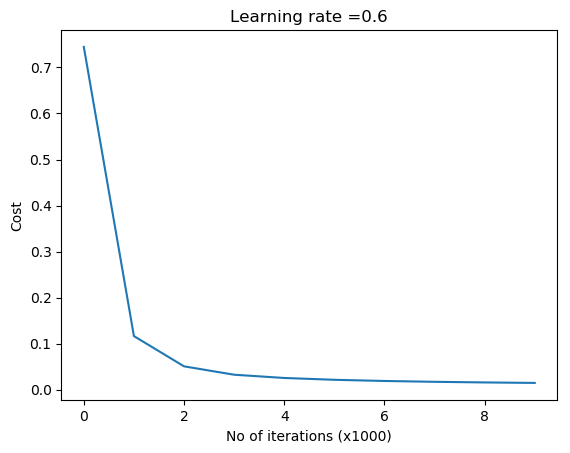
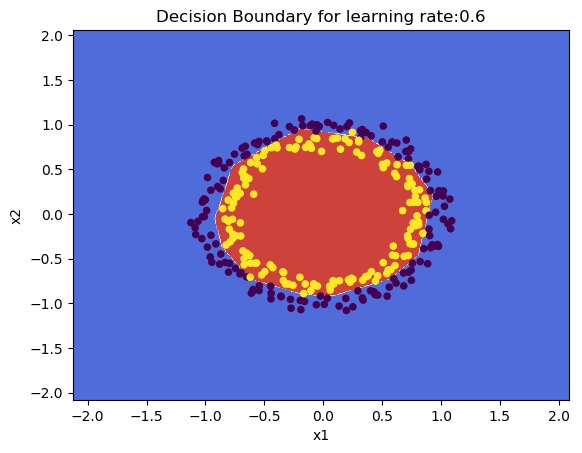
parameters = L\_Layer\_DeepModel(train\_X, train\_Y, layersDimensions, hiddenActivationFunc='relu', outputActivationFunc="sigmoid", learningRate =0.6, num\_iterations = 10000,initType="He",print\_cost = True, figure="fig3.png")

plt.clf()

plt.close()

# Plot the decision boundary

plot\_decision\_boundary(lambda x: predict(parameters, x.T), train\_X, train\_Y,str(0.6),figure1="fig4.png")

**1.1 c Xavier initialization – Python**

[Xavier](http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf) initialization multiply the random weights by  
\sqrt{\frac{1}{dimension\ of\ previous\ layer}}

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

import sklearn.linear\_model

import pandas as pd

import sklearn

import sklearn.datasets

exec(open("DLfunctions61.py").read())

#Load the data

train\_X, train\_Y, test\_X, test\_Y = load\_dataset()

# Set the layers dimensions

layersDimensions = [2,7,1]

# Train a L layer Deep Learning network

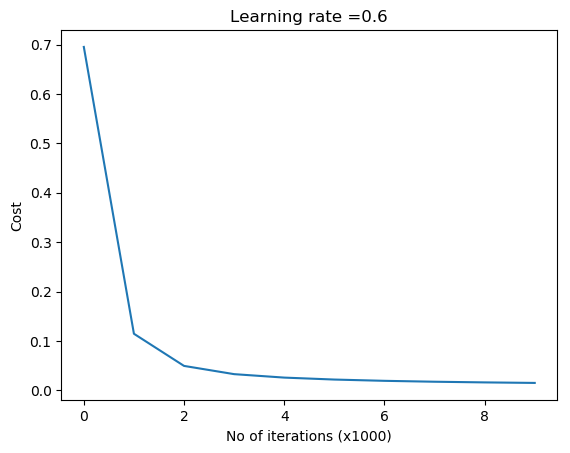
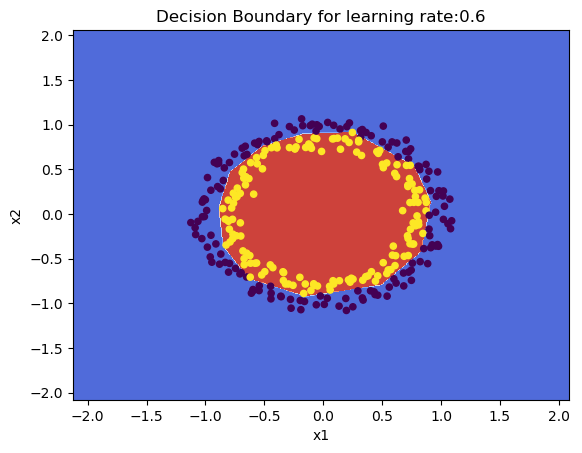
parameters = L\_Layer\_DeepModel(train\_X, train\_Y, layersDimensions, hiddenActivationFunc='relu', outputActivationFunc="sigmoid",

learningRate = 0.6,num\_iterations = 10000, initType="Xavier",print\_cost = True,

figure="fig5.png")

# Plot the decision boundary

plot\_decision\_boundary(lambda x: predict(parameters, x.T), train\_X, train\_Y,str(0.6),figure1="fig6.png")

**1.2a Default initialization – R**

source("DLfunctions61.R")

#Load the data

z <- as.matrix(read.csv("circles.csv",header=FALSE))

x <- z[,1:2]

y <- z[,3]

X <- t(x)

Y <- t(y)

#Set the layer dimensions

layersDimensions = c(2,11,1)

# Train a deep learning network

retvals = L\_Layer\_DeepModel(X, Y, layersDimensions,

hiddenActivationFunc='relu',

outputActivationFunc="sigmoid",

learningRate = 0.5,

numIterations = 8000,

initType="default",

print\_cost = True)

#Plot the cost vs iterations

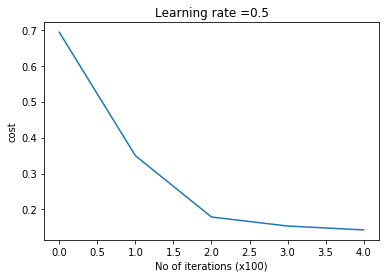
iterations <- seq(0,8000,1000)

costs=retvals$costs

df=data.frame(iterations,costs)

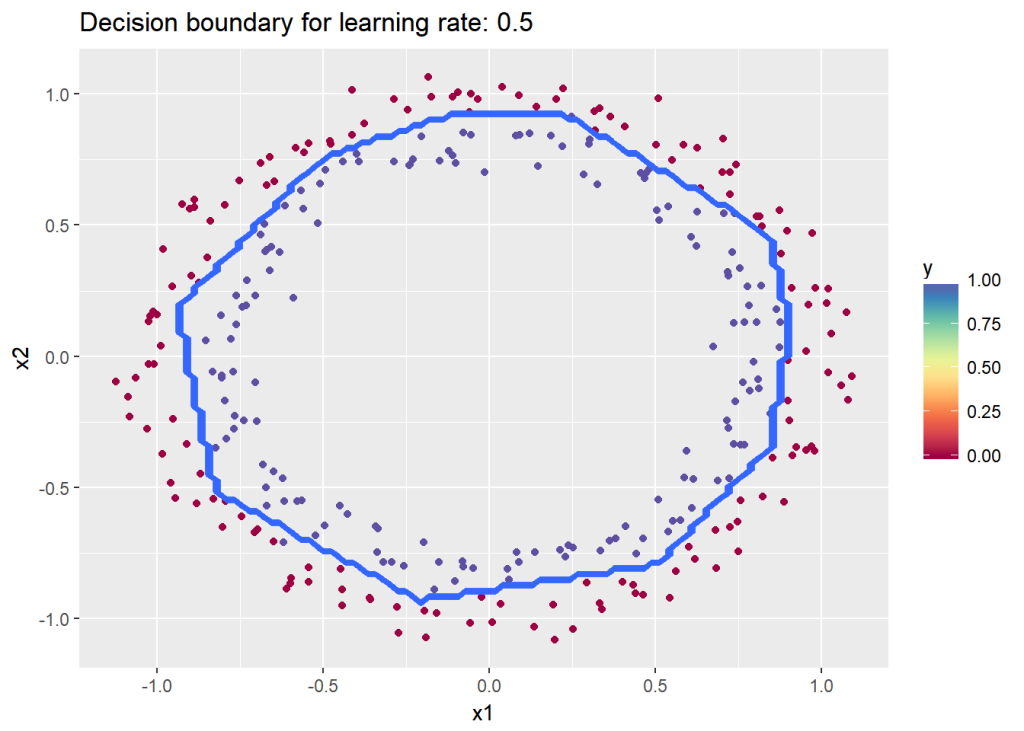
ggplot(df,aes(x=iterations,y=costs)) + geom\_point() + geom\_line(color="blue") +

ggtitle("Costs vs iterations") + xlab("No of iterations") + ylab("Cost")



# Plot the decision boundary

plotDecisionBoundary(z,retvals,hiddenActivationFunc="relu",lr=0.5)



**1.2b He initialization – R**

The code for ‘He’ initilaization in R is included below

# He Initialization model for L layers

# Input : List of units in each layer

# Returns: Initial weights and biases matrices for all layers

# He initilization multiplies the random numbers with sqrt(2/layerDimensions[previouslayer])

HeInitializeDeepModel <- function(layerDimensions){

set.seed(2)

# Initialize empty list

layerParams <- list()

# Note the Weight matrix at layer 'l' is a matrix of size (l,l-1)

# The Bias is a vectors of size (l,1)

# Loop through the layer dimension from 1.. L

# Indices in R start from 1

for(l in 2:length(layersDimensions)){

# Initialize a matrix of small random numbers of size l x l-1

# Create random numbers of size l x l-1

w=rnorm(layersDimensions[l]\*layersDimensions[l-1])

# Create a weight matrix of size l x l-1 with this initial weights and

# Add to list W1,W2... WL

# He initialization - Divide by sqrt(2/layerDimensions[previous layer])

layerParams[[paste('W',l-1,sep="")]] = matrix(w,nrow=layersDimensions[l],

ncol=layersDimensions[l-1])\*sqrt(2/layersDimensions[l-1])

layerParams[[paste('b',l-1,sep="")]] = matrix(rep(0,layersDimensions[l]),

nrow=layersDimensions[l],ncol=1)

}

return(layerParams)

}

The code in R below uses He initialization to learn the data

source("DLfunctions61.R")

# Load the data

z <- as.matrix(read.csv("circles.csv",header=FALSE))

x <- z[,1:2]

y <- z[,3]

X <- t(x)

Y <- t(y)

# Set the layer dimensions

layersDimensions = c(2,11,1)

# Train a deep learning network

retvals = L\_Layer\_DeepModel(X, Y, layersDimensions,

hiddenActivationFunc='relu',

outputActivationFunc="sigmoid",

learningRate = 0.5,

numIterations = 9000,

initType="He",

print\_cost = True)

#Plot the cost vs iterations

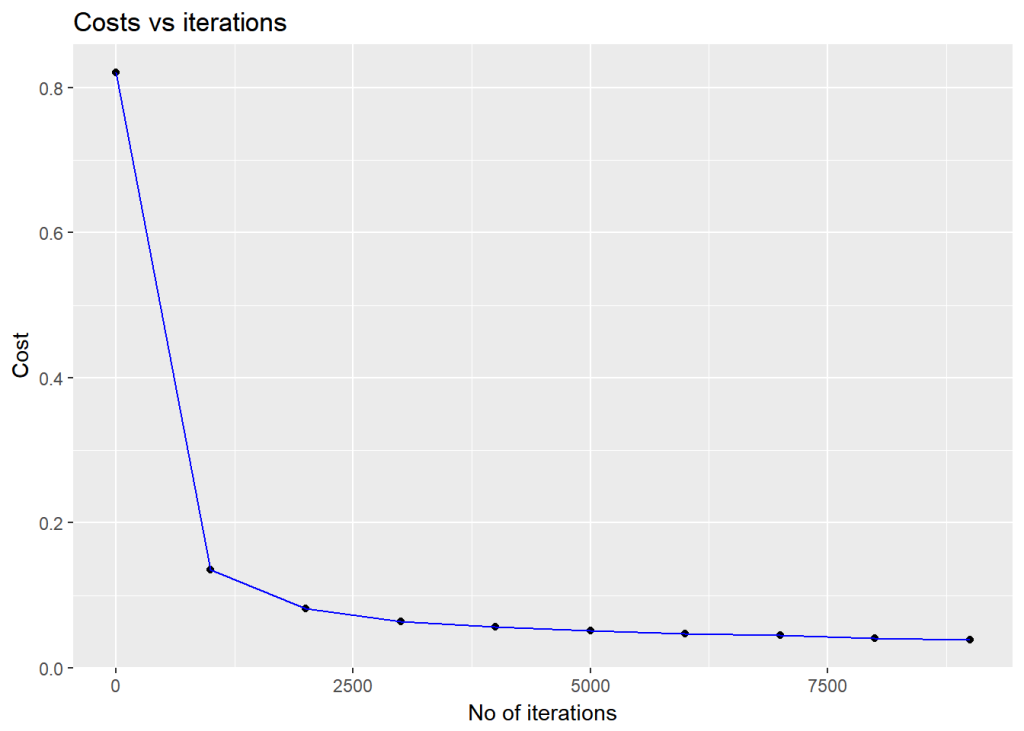
iterations <- seq(0,9000,1000)

costs=retvals$costs

df=data.frame(iterations,costs)

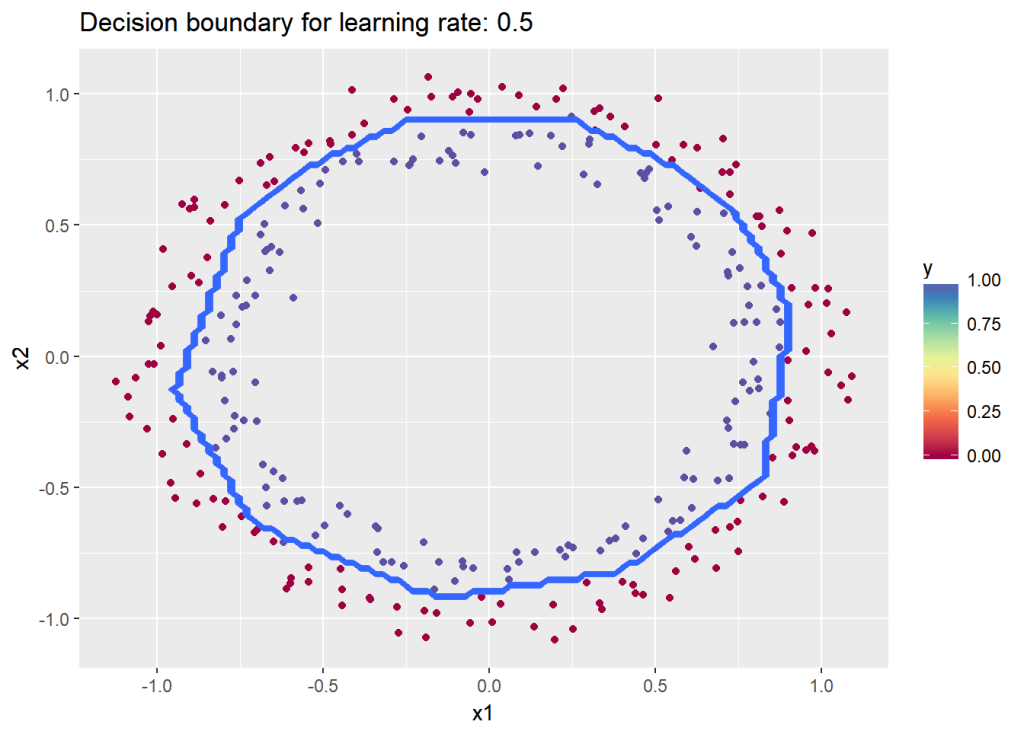
ggplot(df,aes(x=iterations,y=costs)) + geom\_point() + geom\_line(color="blue") +

ggtitle("Costs vs iterations") + xlab("No of iterations") + ylab("Cost")



# Plot the decision boundary

plotDecisionBoundary(z,retvals,hiddenActivationFunc="relu",0.5,lr=0.5)



**1.2c Xavier initialization – R**

## Xav initialization

# Set the layer dimensions

layersDimensions = c(2,11,1)

# Train a deep learning network

retvals = L\_Layer\_DeepModel(X, Y, layersDimensions,

hiddenActivationFunc='relu',

outputActivationFunc="sigmoid",

learningRate = 0.5,

numIterations = 9000,

initType="Xav",

print\_cost = True)

#Plot the cost vs iterations

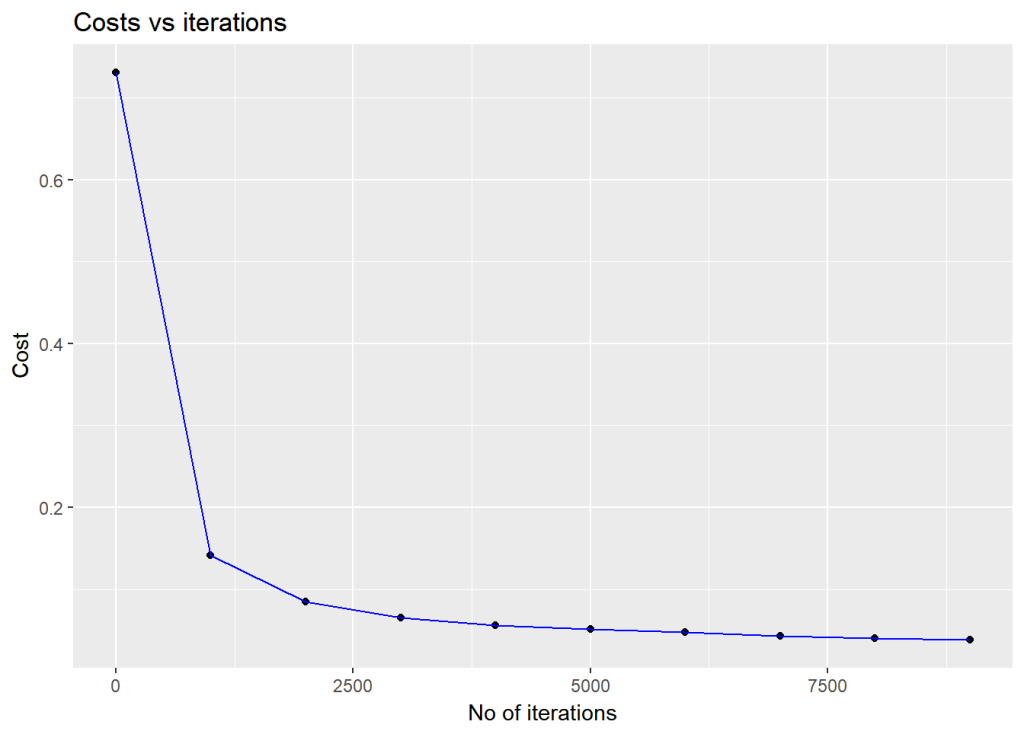
iterations <- seq(0,9000,1000)

costs=retvals$costs

df=data.frame(iterations,costs)

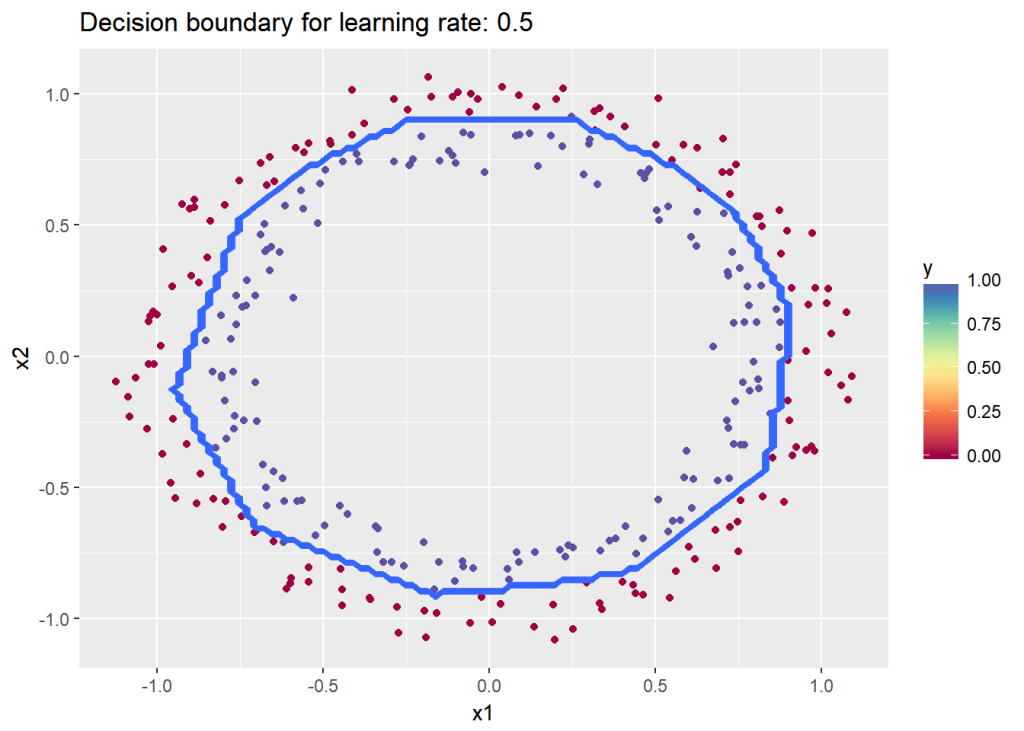
ggplot(df,aes(x=iterations,y=costs)) + geom\_point() + geom\_line(color="blue") +

ggtitle("Costs vs iterations") + xlab("No of iterations") + ylab("Cost")



# Plot the decision boundary

plotDecisionBoundary(z,retvals,hiddenActivationFunc="relu",0.5)



**1.3a Default initialization – Octave**

source("DL61functions.m")

# Read the data

data=csvread("circles.csv");

X=data(:,1:2);

Y=data(:,3);

# Set the layer dimensions

layersDimensions = [2 11 1]; #tanh=-0.5(ok), #relu=0.1 best!

# Train a deep learning network

[weights biases costs]=L\_Layer\_DeepModel(X', Y', layersDimensions,

hiddenActivationFunc='relu',

outputActivationFunc="sigmoid",

learningRate = 0.5,

lambd=0,

keep\_prob=1,

numIterations = 10000,

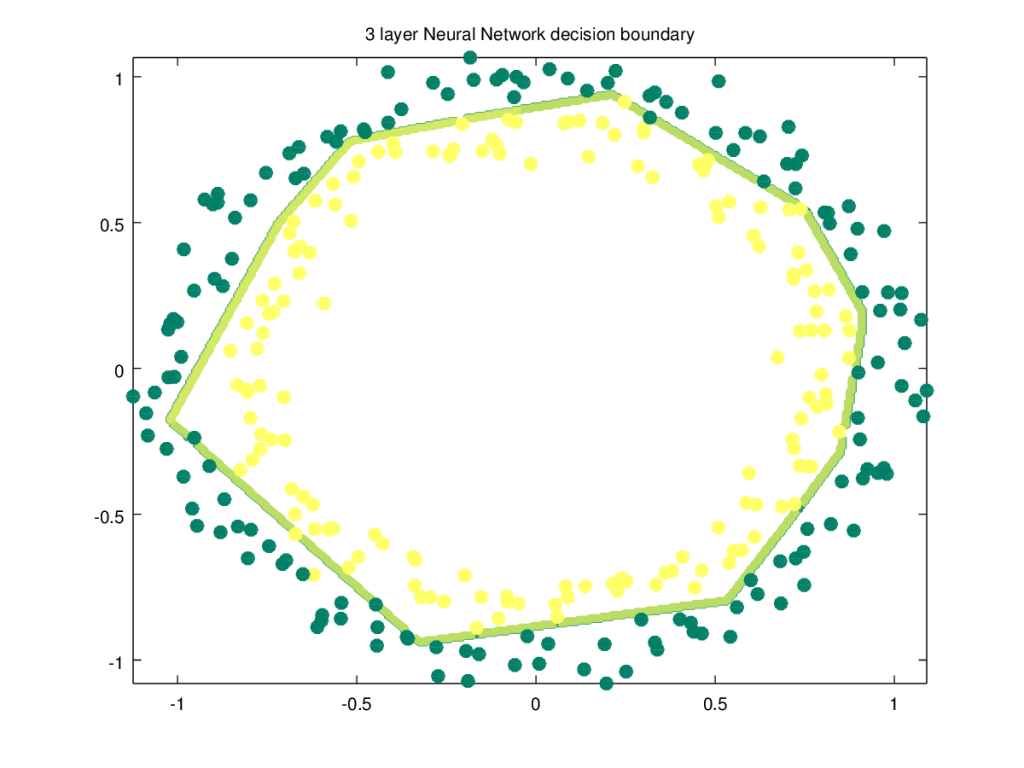
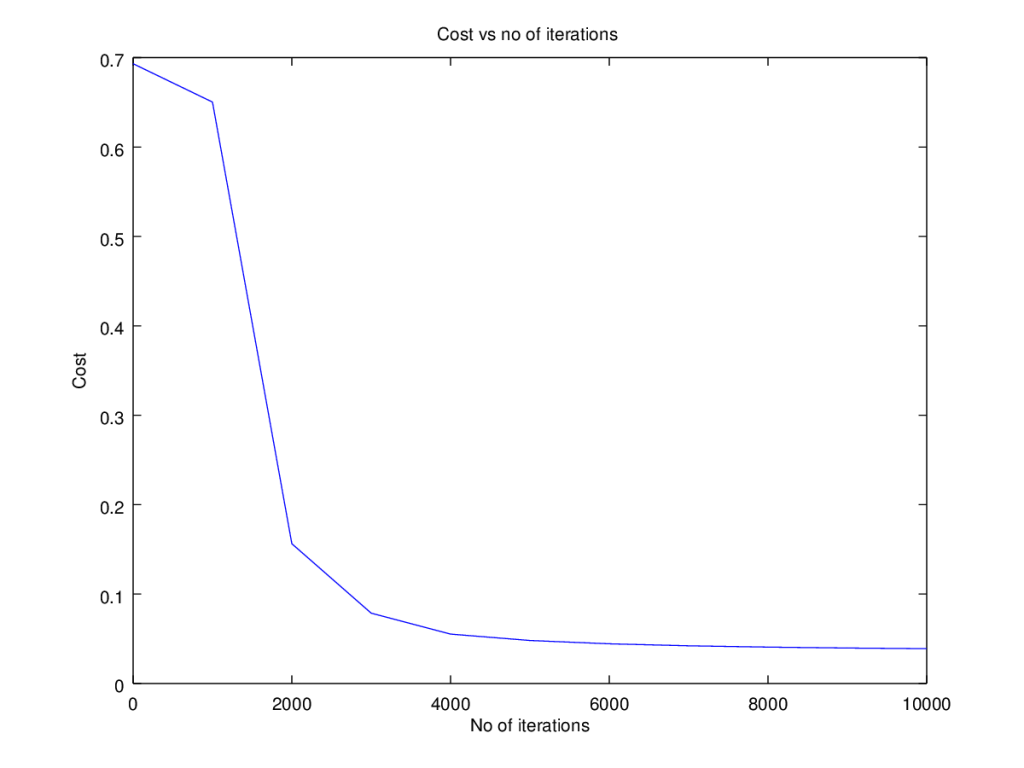
initType="default");

# Plot cost vs iterations

plotCostVsIterations(10000,costs)

#Plot decision boundary

plotDecisionBoundary(data,weights, biases,keep\_prob=1, hiddenActivationFunc="relu")

****

**1.3b He initialization – Octave**

source("DL61functions.m")

#Load data

data=csvread("circles.csv");

X=data(:,1:2);

Y=data(:,3);

# Set the layer dimensions

layersDimensions = [2 11 1]; #tanh=-0.5(ok), #relu=0.1 best!

# Train a deep learning network

[weights biases costs]=L\_Layer\_DeepModel(X', Y', layersDimensions,

hiddenActivationFunc='relu',

outputActivationFunc="sigmoid",

learningRate = 0.5,

lambd=0,

keep\_prob=1,

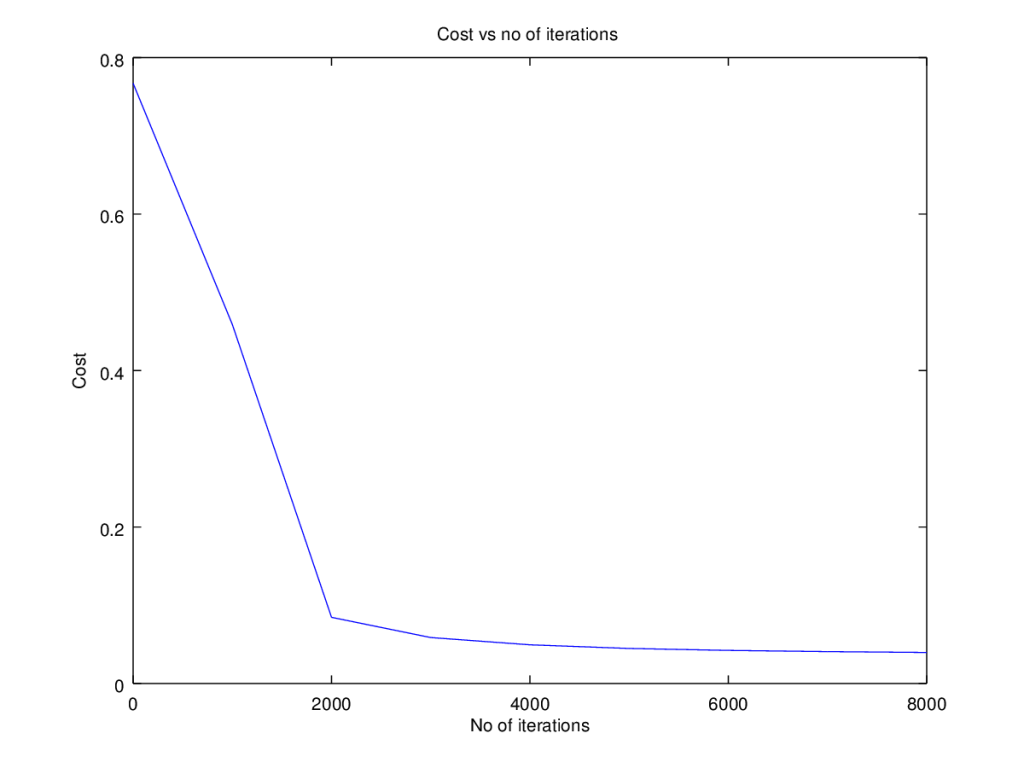
numIterations = 8000,

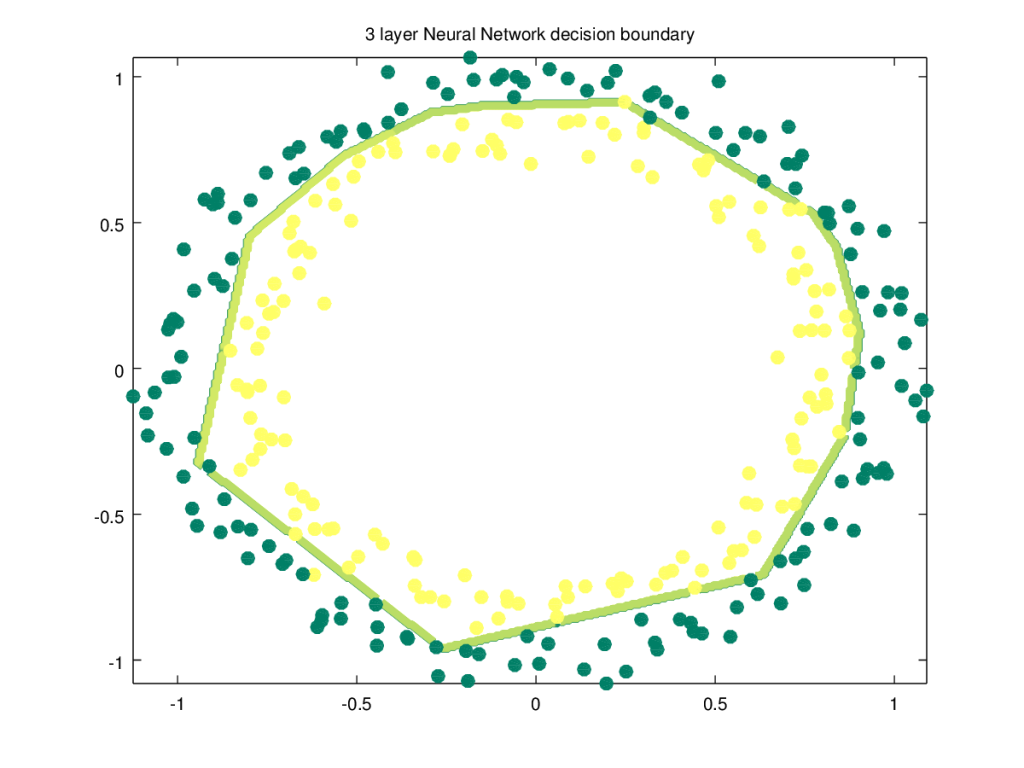
initType="He");

plotCostVsIterations(8000,costs)

#Plot decision boundary

plotDecisionBoundary(data,weights, biases,keep\_prob=1,hiddenActivationFunc="relu")





**1.3c Xavier initialization – Octave**

The code snippet for Xavier initialization in Octave is shown below

source("DL61functions.m")

# Xavier Initialization for L layers

# Input : List of units in each layer

# Returns: Initial weights and biases matrices for all layers

function [W b] = XavInitializeDeepModel(layerDimensions)

rand ("seed", 3);

# note the Weight matrix at layer 'l' is a matrix of size (l,l-1)

# The Bias is a vectors of size (l,1)

# Loop through the layer dimension from 1.. L

# Create cell arrays for Weights and biases

for l =2:size(layerDimensions)(2)

W{l-1} = rand(layerDimensions(l),layerDimensions(l-1))\* sqrt(1/layerDimensions(l-1)); # Multiply by .01

b{l-1} = zeros(layerDimensions(l),1);

endfor

end

The Octave code below uses Xavier initialization

source("DL61functions.m")

#Load data

data=csvread("circles.csv");

X=data(:,1:2);

Y=data(:,3);

#Set layer dimensions

layersDimensions = [2 11 1]; #tanh=-0.5(ok), #relu=0.1 best!

# Train a deep learning network

[weights biases costs]=L\_Layer\_DeepModel(X', Y', layersDimensions,

hiddenActivationFunc='relu',

outputActivationFunc="sigmoid",

learningRate = 0.5,

lambd=0,

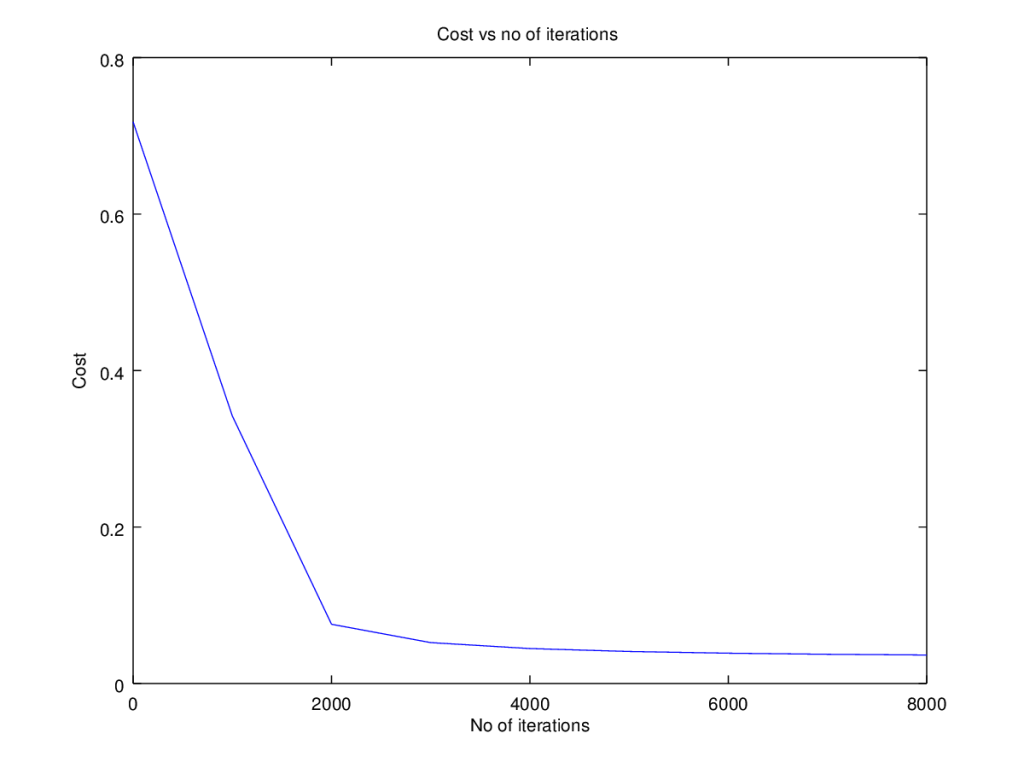
keep\_prob=1,

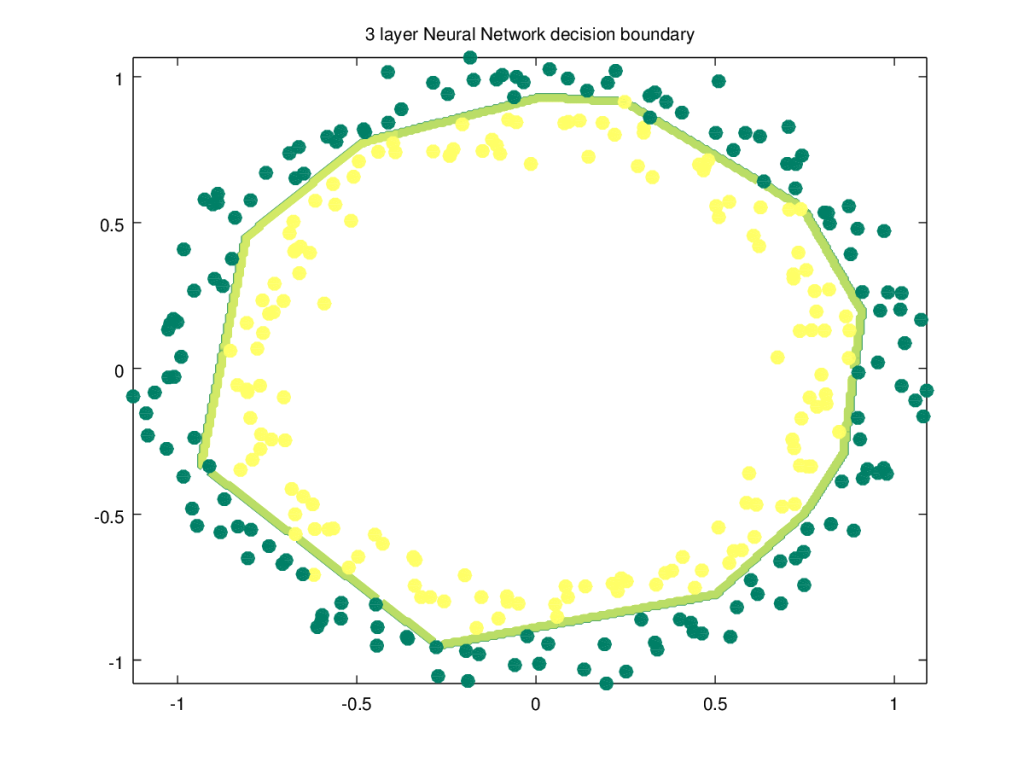
numIterations = 8000,

initType="Xav");

plotCostVsIterations(8000,costs)

plotDecisionBoundary(data,weights, biases,keep\_prob=1,hiddenActivationFunc="relu")





**2.1a Regularization : Circles data – Python**

The cross entropy cost for Logistic classification is given as J = \frac{1}{m}\sum_{i=1}^{m}y^{i}log((a^{L})^{(i)}) - (1-y^{i})log((a^{L})^{(i)}) The regularized L2 cost is given by

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

import sklearn.linear\_model

import pandas as pd

import sklearn

import sklearn.datasets

exec(open("DLfunctions61.py").read())

#Load the data

train\_X, train\_Y, test\_X, test\_Y = load\_dataset()

# Set the layers dimensions

layersDimensions = [2,7,1]

# Train a deep learning network

parameters = L\_Layer\_DeepModel(train\_X, train\_Y, layersDimensions, hiddenActivationFunc='relu',

outputActivationFunc="sigmoid",learningRate = 0.6, lambd=0.1, num\_iterations = 9000,

initType="default", print\_cost = True,figure="fig7.png")

# Clear the plot

plt.clf()

plt.close()

# Plot the decision boundary

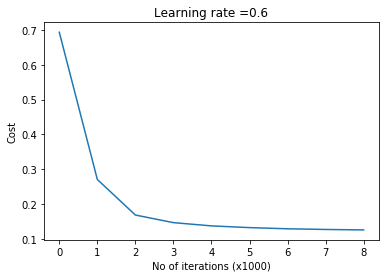
plot\_decision\_boundary(lambda x: predict(parameters, x.T), train\_X, train\_Y,str(0.6),figure1="fig8.png")

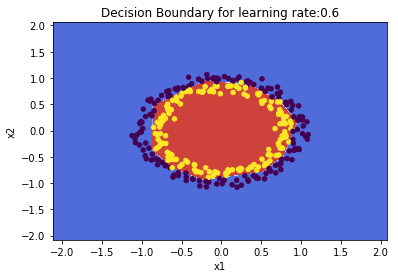
plt.clf()

plt.close()

#Plot the decision boundary

plot\_decision\_boundary(lambda x: predict(parameters, x.T,keep\_prob=0.9), train\_X, train\_Y,str(2.2),"fig8.png",)





**2.1 b Regularization: Spiral data  – Python**

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

import sklearn.linear\_model

import pandas as pd

import sklearn

import sklearn.datasets

exec(open("DLfunctions61.py").read())

N = 100 # number of points per class

D = 2 # dimensionality

K = 3 # number of classes

X = np.zeros((N\*K,D)) # data matrix (each row = single example)

y = np.zeros(N\*K, dtype='uint8') # class labels

for j in range(K):

ix = range(N\*j,N\*(j+1))

r = np.linspace(0.0,1,N) # radius

t = np.linspace(j\*4,(j+1)\*4,N) + np.random.randn(N)\*0.2 # theta

X[ix] = np.c\_[r\*np.sin(t), r\*np.cos(t)]

y[ix] = j

# Plot the data

plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral)

plt.clf()

plt.close()

#Set layer dimensions

layersDimensions = [2,100,3]

y1=y.reshape(-1,1).T

# Train a deep learning network

parameters = L\_Layer\_DeepModel(X.T, y1, layersDimensions, hiddenActivationFunc='relu', outputActivationFunc="softmax",

learningRate = 1,lambd=1e-3, num\_iterations = 5000, print\_cost = True,figure="fig9.png")

plt.clf()

plt.close()

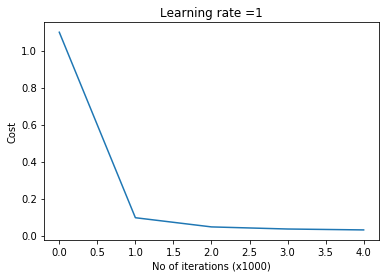
W1=parameters['W1']

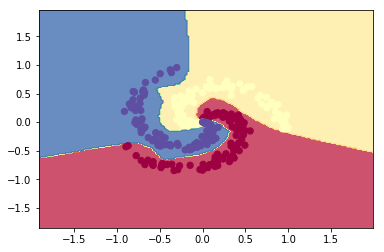
b1=parameters['b1']

W2=parameters['W2']

b2=parameters['b2']

plot\_decision\_boundary1(X, y1,W1,b1,W2,b2,figure2="fig10.png")





**2.2a Regularization: Circles data  – R**

source("DLfunctions61.R")

#Load data

df=read.csv("circles.csv",header=FALSE)

z <- as.matrix(read.csv("circles.csv",header=FALSE))

x <- z[,1:2]

y <- z[,3]

X <- t(x)

Y <- t(y)

#Set layer dimensions

layersDimensions = c(2,11,1)

# Train a deep learning network

retvals = L\_Layer\_DeepModel(X, Y, layersDimensions,

hiddenActivationFunc='relu',

outputActivationFunc="sigmoid",

learningRate = 0.5,

lambd=0.1,

numIterations = 9000,

initType="default",

print\_cost = True)

#Plot the cost vs iterations

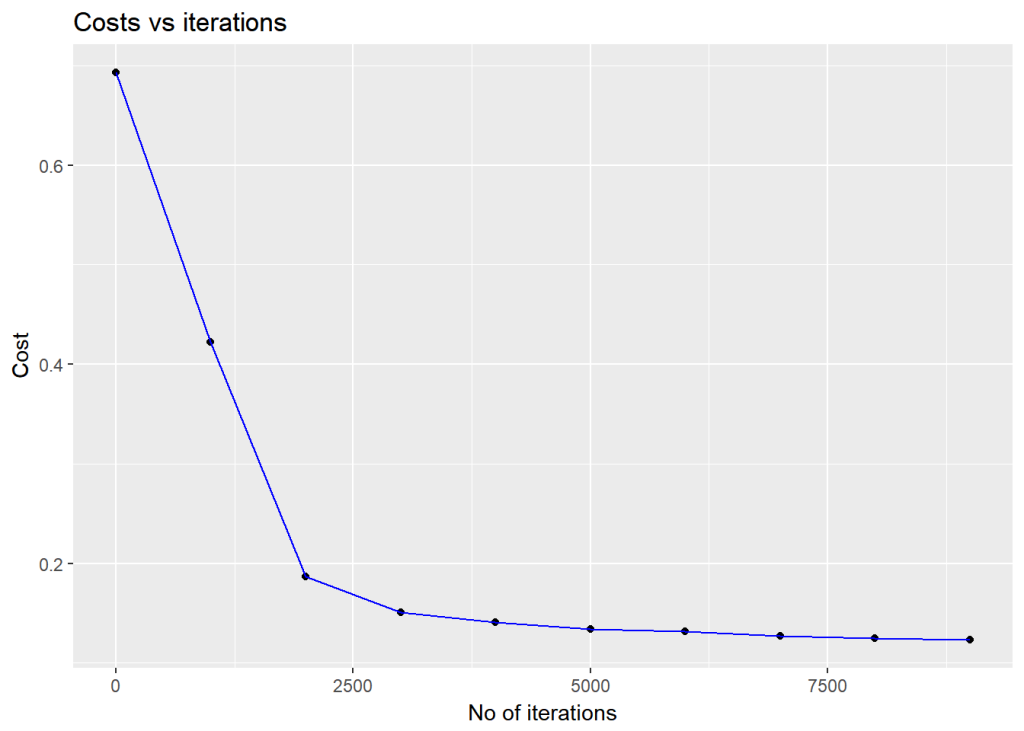
iterations <- seq(0,9000,1000)

costs=retvals$costs

df=data.frame(iterations,costs)

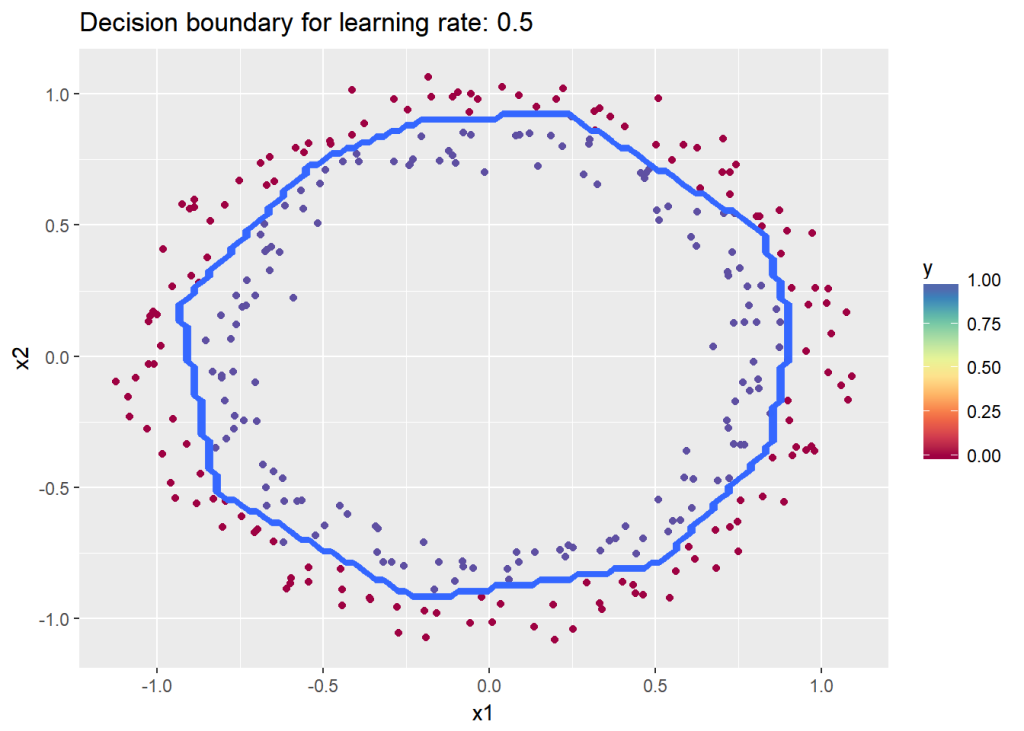
ggplot(df,aes(x=iterations,y=costs)) + geom\_point() + geom\_line(color="blue") +

ggtitle("Costs vs iterations") + xlab("No of iterations") + ylab("Cost")



# Plot the decision boundary

plotDecisionBoundary(z,retvals,hiddenActivationFunc="relu",0.5)



**2.2b Regularization:Spiral data – R**

# Read the spiral dataset

#Load the data

source("DLfunctions61.R")

Z <- as.matrix(read.csv("spiral.csv",header=FALSE))

# Setup the data

X <- Z[,1:2]

y <- Z[,3]

X <- t(X)

Y <- t(y)

layersDimensions = c(2, 100, 3)

# Train a deep learning network

retvals = L\_Layer\_DeepModel(X, Y, layersDimensions,

hiddenActivationFunc='relu',

outputActivationFunc="softmax",

learningRate = 0.5,

lambd=0.01,

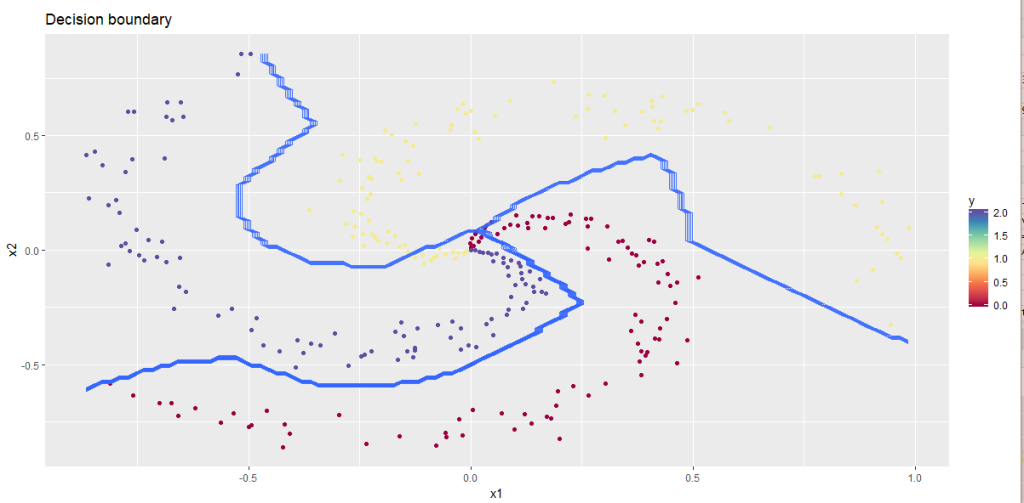
numIterations = 9000,

print\_cost = True)

print\_cost = True)

parameters<-retvals$parameters

plotDecisionBoundary1(Z,parameters)

  
**2.3a Regularization: Circles data – Octave**

source("DL61functions.m")

#Load data

data=csvread("circles.csv");

X=data(:,1:2);

Y=data(:,3);

layersDimensions = [2 11 1]; #tanh=-0.5(ok), #relu=0.1 best!

# Train a deep learning network

[weights biases costs]=L\_Layer\_DeepModel(X', Y', layersDimensions,

hiddenActivationFunc='relu',

outputActivationFunc="sigmoid",

learningRate = 0.5,

lambd=0.2,

keep\_prob=1,

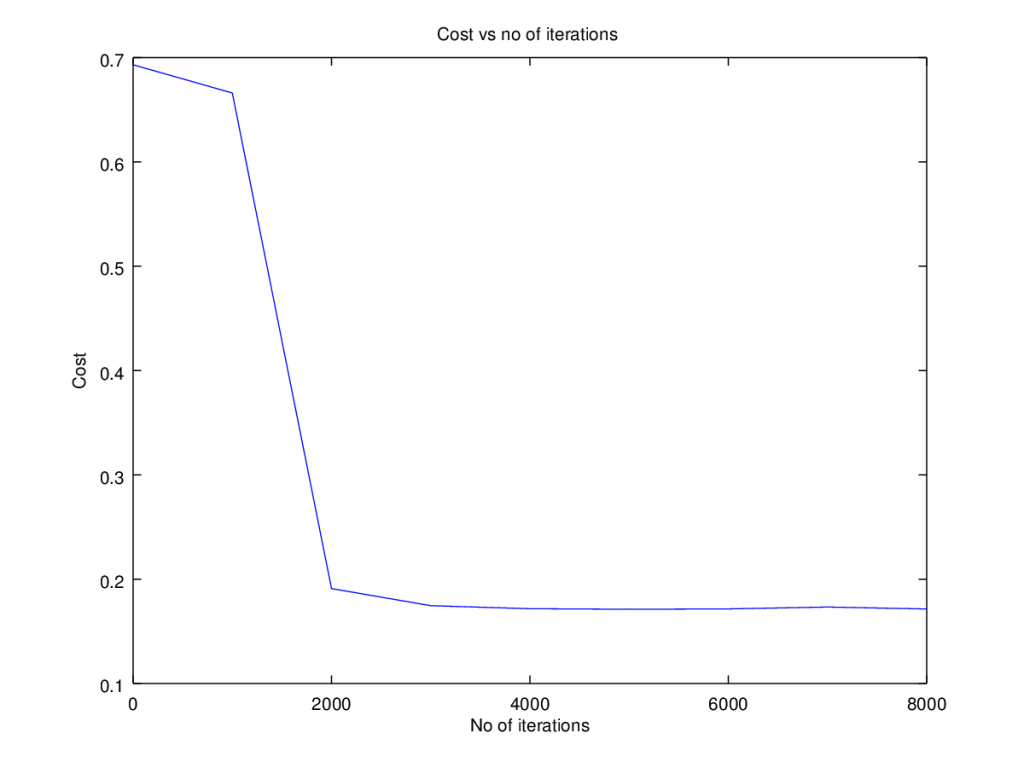
numIterations = 8000,

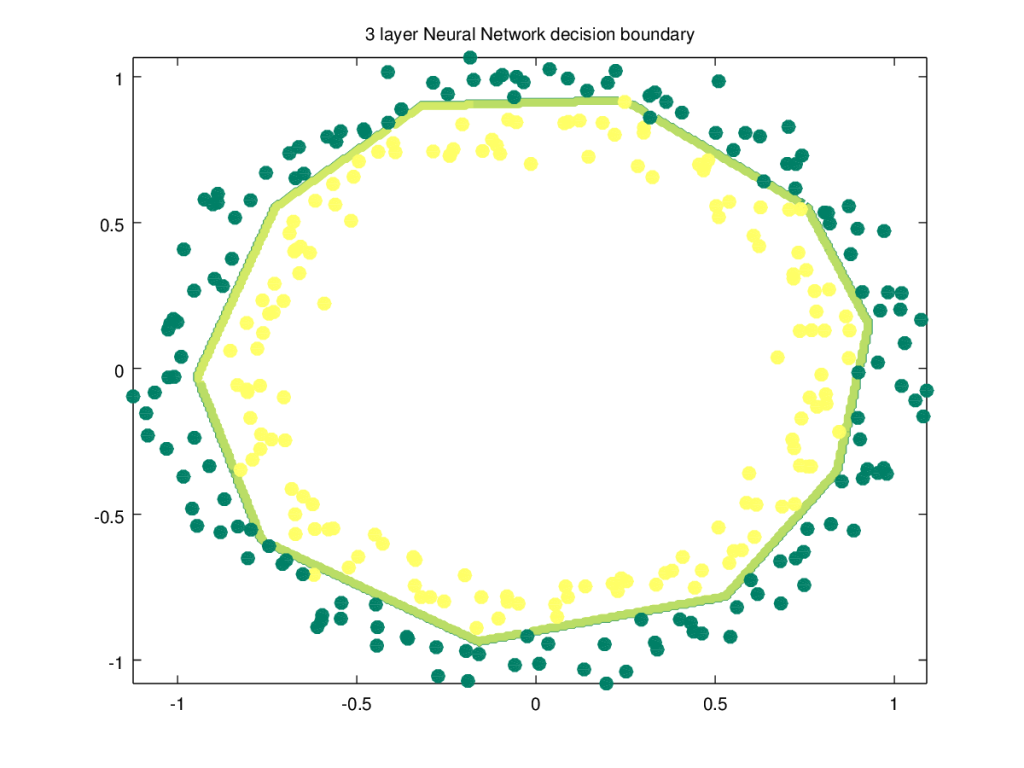
initType="default");

plotCostVsIterations(8000,costs)

#Plot decision boundary

plotDecisionBoundary(data,weights, biases,keep\_prob=1,hiddenActivationFunc="relu")





**2.3b Regularization:Spiral data  2 – Octave**

source("DL61functions.m")

data=csvread("spiral.csv");

# Setup the data

X=data(:,1:2);

Y=data(:,3);

layersDimensions = [2 100 3]

# Train a deep learning network

[weights biases costs]=L\_Layer\_DeepModel(X', Y', layersDimensions,

hiddenActivationFunc='relu',

outputActivationFunc="softmax",

learningRate = 0.6,

lambd=0.2,

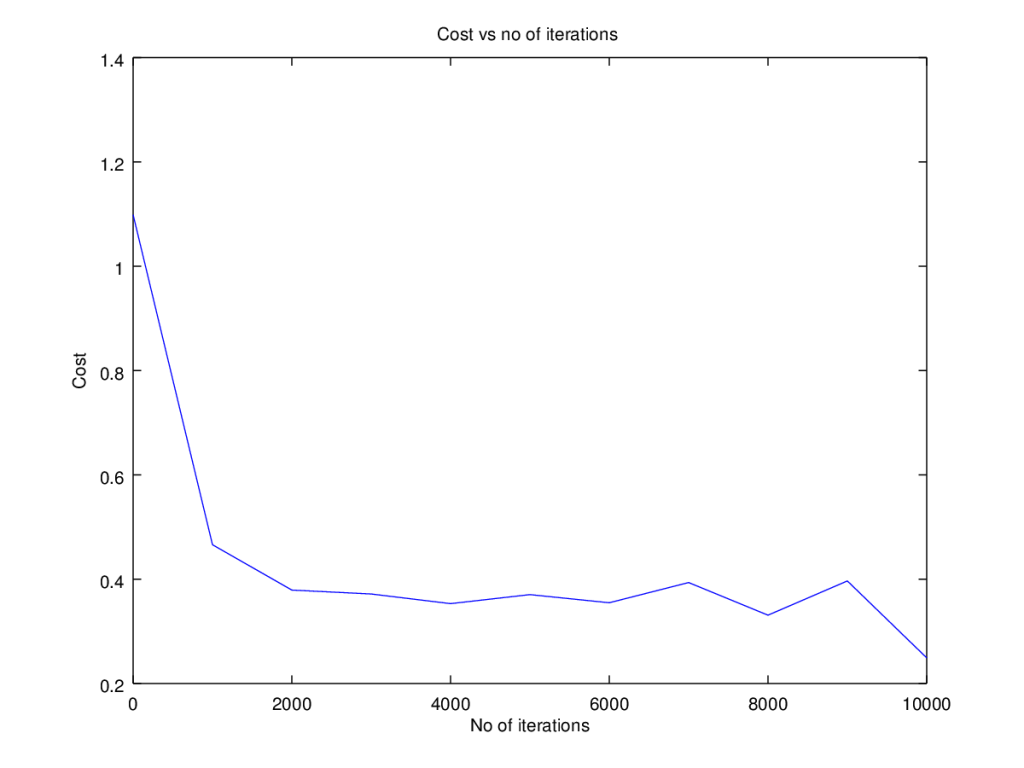
keep\_prob=1,

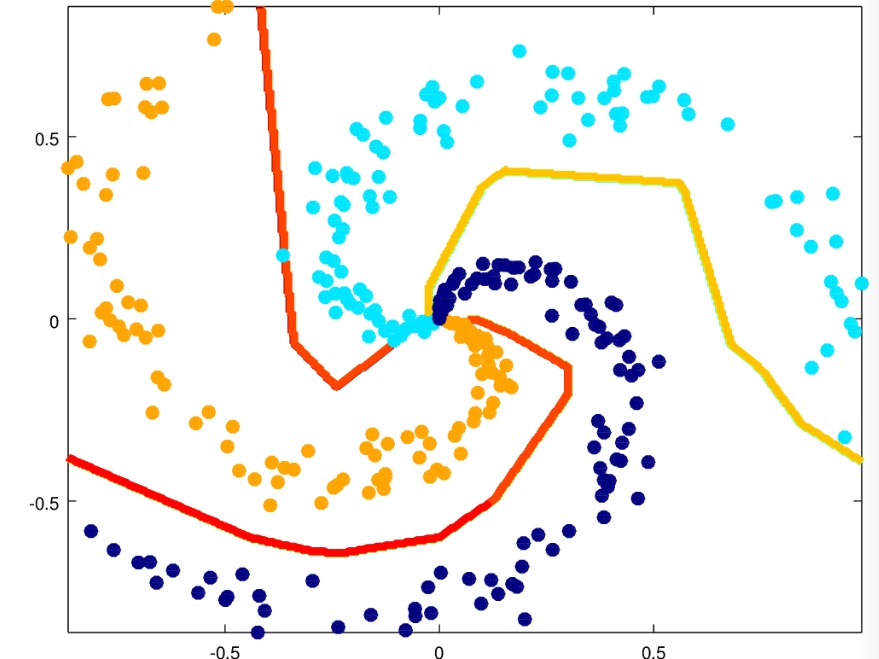
numIterations = 10000);

plotCostVsIterations(10000,costs)

#Plot decision boundary

plotDecisionBoundary1(data,weights, biases,keep\_prob=1,hiddenActivationFunc="relu")





**3.1 a Dropout: Circles data – Python**

The ‘dropout’ regularization technique was used with great effectiveness, to prevent overfitting  by Alex Krizhevsky, Ilya Sutskever and Prof Geoffrey E. Hinton in the [Imagenet classification with Deep Convolutional Neural Networks](https://www.nvidia.cn/content/tesla/pdf/machine-learning/imagenet-classification-with-deep-convolutional-nn.pdf)

The technique of dropout works by dropping a random set of activation units in each hidden layer, based on a ‘keep\_prob’ criteria in the forward propagation cycle. Here is the code for Octave. A ‘dropoutMat’ is created for each layer which specifies which units to drop **Note**: The same ‘dropoutMat has to be used which computing the gradients in the backward propagation cycle. Hence the dropout matrices are stored in a cell array.

 for l =1:L-1

...

D=rand(size(A)(1),size(A)(2));

D = (D < keep\_prob) ;

# Zero out some hidden units

A= A .\* D;

# Divide by keep\_prob to keep the expected value of A the same

A = A ./ keep\_prob;

# Store D in a dropoutMat cell array

dropoutMat{l}=D;

...

endfor

In the backward propagation cycle we have

for l =(L-1):-1:1

...

D = dropoutMat{l};

# Zero out the dAl based on same dropout matrix

dAl= dAl .\* D;

# Divide by keep\_prob to maintain the expected value

dAl = dAl ./ keep\_prob;

...

endfor

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

import sklearn.linear\_model

import pandas as pd

import sklearn

import sklearn.datasets

exec(open("DLfunctions61.py").read())

#Load the data

train\_X, train\_Y, test\_X, test\_Y = load\_dataset()

# Set the layers dimensions

layersDimensions = [2,7,1]

# Train a deep learning network

parameters = L\_Layer\_DeepModel(train\_X, train\_Y, layersDimensions, hiddenActivationFunc='relu',

outputActivationFunc="sigmoid",learningRate = 0.6, keep\_prob=0.7, num\_iterations = 9000,

initType="default", print\_cost = True,figure="fig11.png")

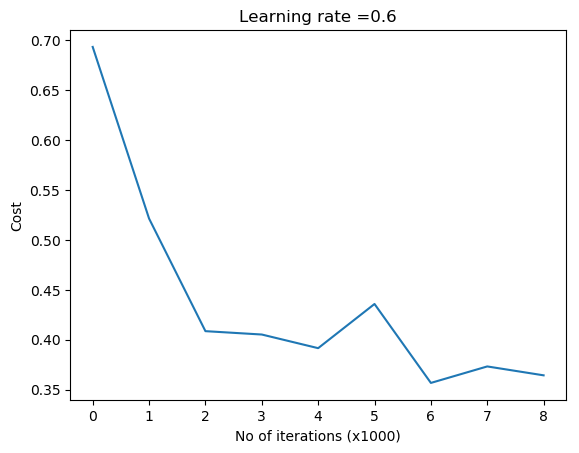
# Clear the plot

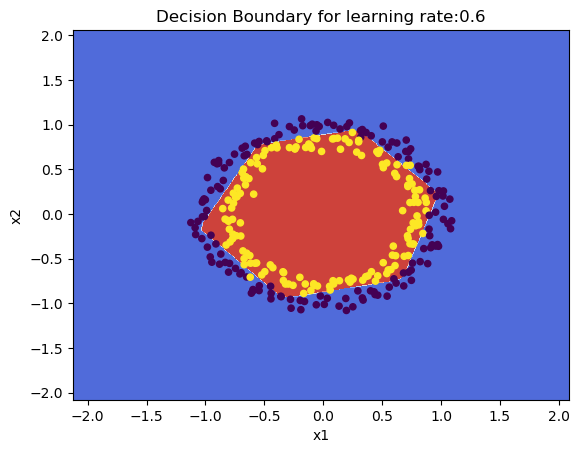
plt.clf()

plt.close()

# Plot the decision boundary

plot\_decision\_boundary(lambda x: predict(parameters, x.T,keep\_prob=0.7), train\_X, train\_Y,str(0.6),figure1="fig12.png")





**3.1b Dropout: Spiral data – Python**

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

import sklearn.linear\_model

import pandas as pd

import sklearn

import sklearn.datasets

exec(open("DLfunctions61.py").read())

# Create an input data set - Taken from CS231n Convolutional Neural networks,

# http://cs231n.github.io/neural-networks-case-study/

N = 100 # number of points per class

D = 2 # dimensionality

K = 3 # number of classes

X = np.zeros((N\*K,D)) # data matrix (each row = single example)

y = np.zeros(N\*K, dtype='uint8') # class labels

for j in range(K):

ix = range(N\*j,N\*(j+1))

r = np.linspace(0.0,1,N) # radius

t = np.linspace(j\*4,(j+1)\*4,N) + np.random.randn(N)\*0.2 # theta

X[ix] = np.c\_[r\*np.sin(t), r\*np.cos(t)]

y[ix] = j

# Plot the data

plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral)

plt.clf()

plt.close()

layersDimensions = [2,100,3]

y1=y.reshape(-1,1).T

# Train a deep learning network

parameters = L\_Layer\_DeepModel(X.T, y1, layersDimensions, hiddenActivationFunc='relu', outputActivationFunc="softmax",

learningRate = 1,keep\_prob=0.9, num\_iterations = 5000, print\_cost = True,figure="fig13.png")

plt.clf()

plt.close()

W1=parameters['W1']

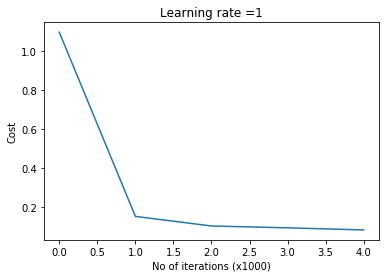
b1=parameters['b1']

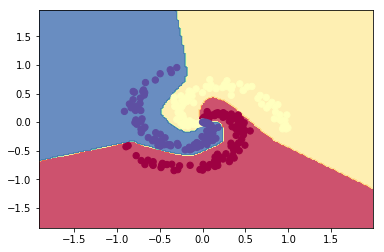
W2=parameters['W2']

b2=parameters['b2']

#Plot decision boundary

plot\_decision\_boundary1(X, y1,W1,b1,W2,b2,figure2="fig14.png")





**3.2a Dropout: Circles data – R**

source("DLfunctions61.R")

#Load data

df=read.csv("circles.csv",header=FALSE)

z <- as.matrix(read.csv("circles.csv",header=FALSE))

x <- z[,1:2]

y <- z[,3]

X <- t(x)

Y <- t(y)

layersDimensions = c(2,11,1)

# Train a deep learning network

retvals = L\_Layer\_DeepModel(X, Y, layersDimensions,

hiddenActivationFunc='relu',

outputActivationFunc="sigmoid",

learningRate = 0.5,

keep\_prob=0.8,

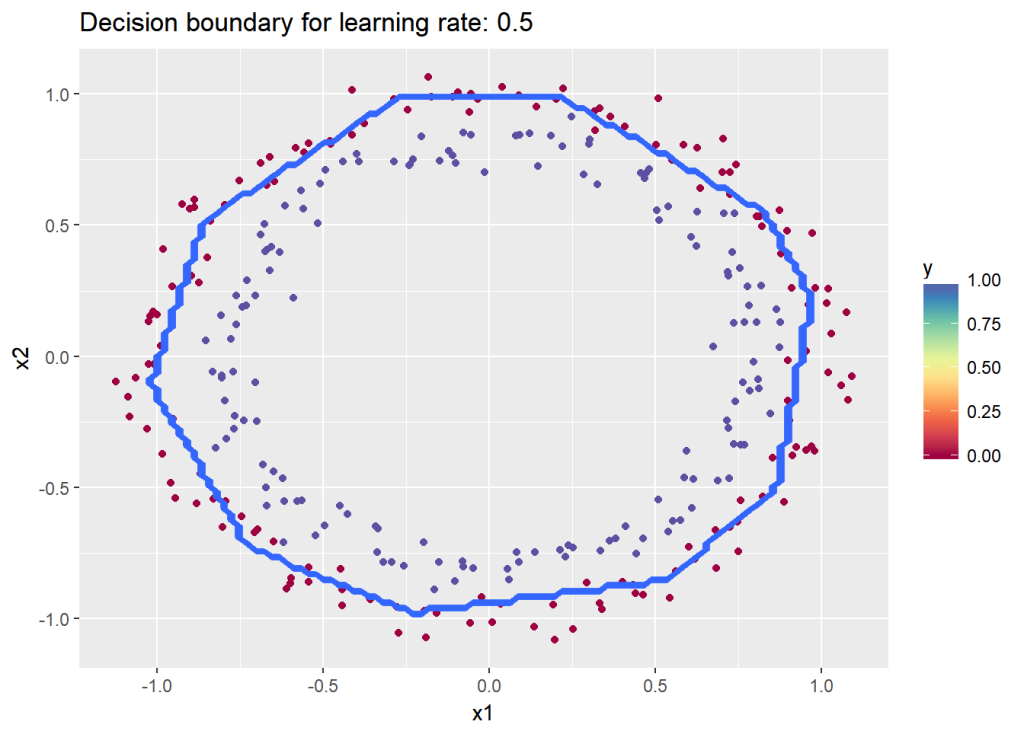
numIterations = 9000,

initType="default",

print\_cost = True)

# Plot the decision boundary

plotDecisionBoundary(z,retvals,keep\_prob=0.6, hiddenActivationFunc="relu",0.5)



**3.2b Dropout: Spiral data – R**

# Read the spiral dataset

source("DLfunctions61.R")

# Load data

Z <- as.matrix(read.csv("spiral.csv",header=FALSE))

# Setup the data

X <- Z[,1:2]

y <- Z[,3]

X <- t(X)

Y <- t(y)

# Train a deep learning network

retvals = L\_Layer\_DeepModel(X, Y, layersDimensions,

hiddenActivationFunc='relu',

outputActivationFunc="softmax",

learningRate = 0.1,

keep\_prob=0.90,

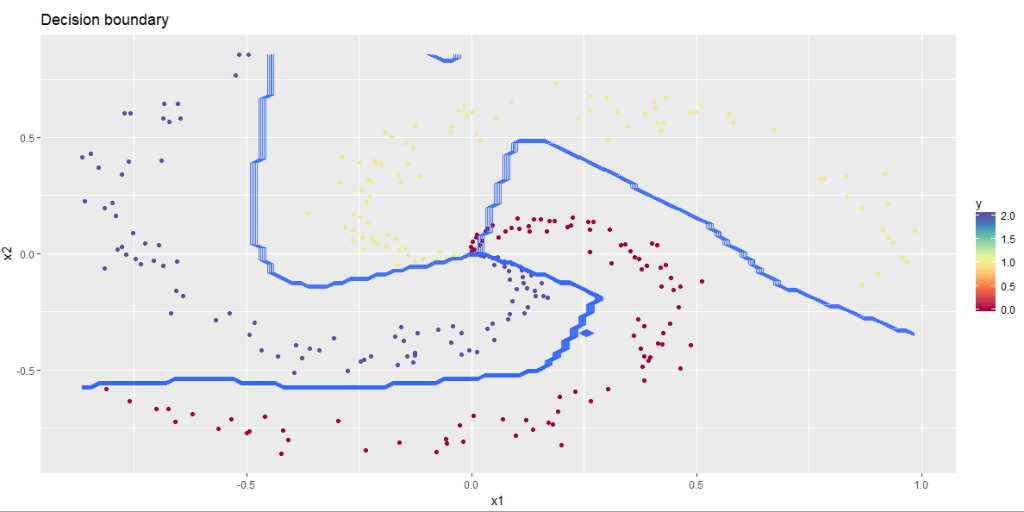
numIterations = 9000,

print\_cost = True)

parameters<-retvals$parameters

#Plot decision boundary

plotDecisionBoundary1(Z,parameters)



**3.3a Dropout: Circles data – Octave**

data=csvread("circles.csv");

X=data(:,1:2);

Y=data(:,3);

layersDimensions = [2 11 1]; #tanh=-0.5(ok), #relu=0.1 best!

# Train a deep learning network

[weights biases costs]=L\_Layer\_DeepModel(X', Y', layersDimensions,

hiddenActivationFunc='relu',

outputActivationFunc="sigmoid",

learningRate = 0.5,

lambd=0,

keep\_prob=0.8,

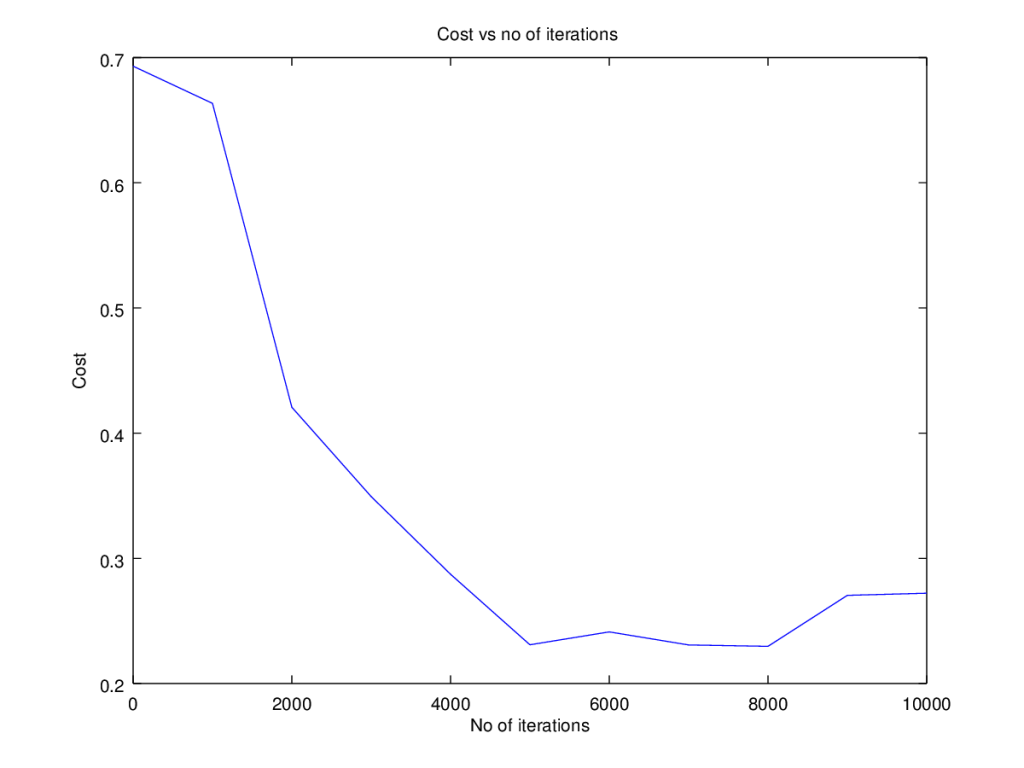
numIterations = 10000,

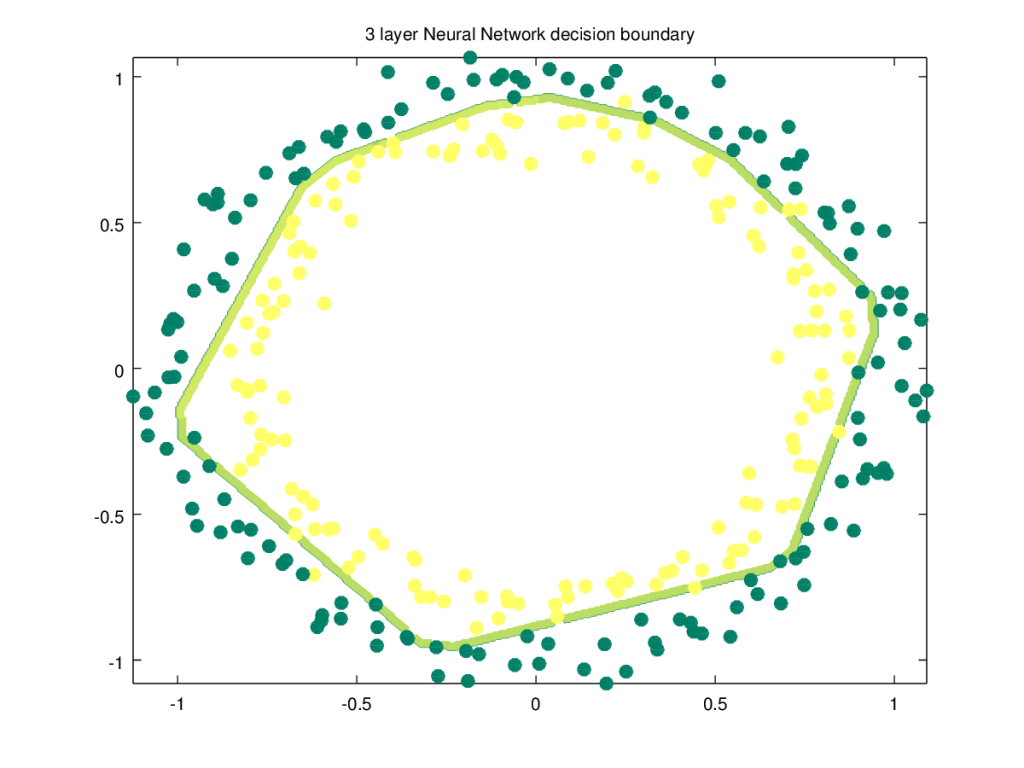
initType="default");

plotCostVsIterations(10000,costs)

#Plot decision boundary

plotDecisionBoundary1(data,weights, biases,keep\_prob=1, hiddenActivationFunc="relu")





**3.3b Dropout  Spiral data – Octave**

source("DL61functions.m")

data=csvread("spiral.csv");

# Setup the data

X=data(:,1:2);

Y=data(:,3);

layersDimensions = [numFeats numHidden numOutput];

# Train a deep learning network

[weights biases costs]=L\_Layer\_DeepModel(X', Y', layersDimensions,

hiddenActivationFunc='relu',

outputActivationFunc="softmax",

learningRate = 0.1,

lambd=0,

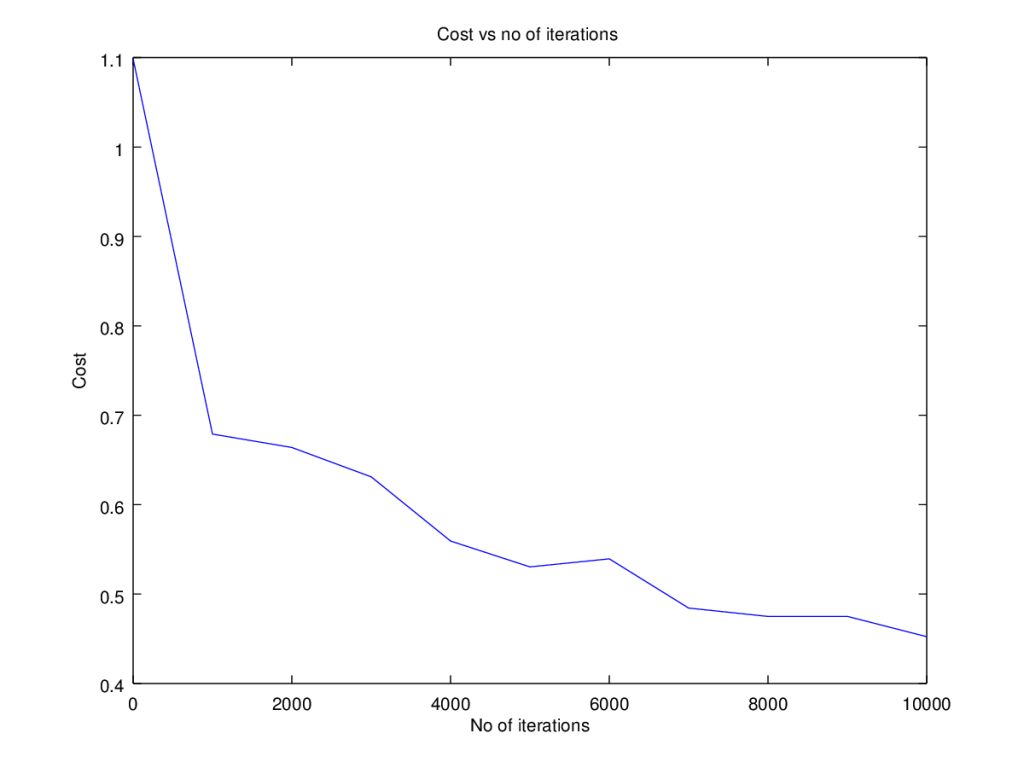
keep\_prob=0.8,

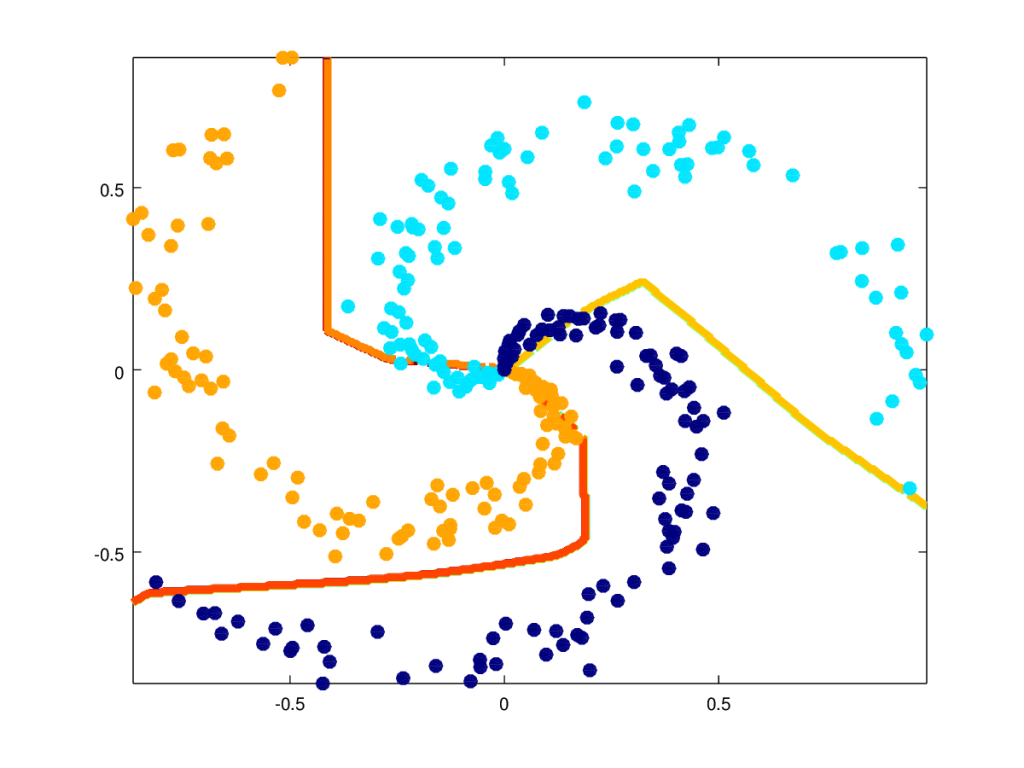
numIterations = 10000);

plotCostVsIterations(10000,costs)

#Plot decision boundary

plotDecisionBoundary1(data,weights, biases,keep\_prob=1, hiddenActivationFunc="relu")





**Note**: The Python, R and Octave code can be cloned/downloaded from Github at [DeepLearning-Part6](https://github.com/tvganesh/DeepLearning-Part6)  
**Conclusion**  
This post further enhances my earlier L-Layer generic implementation of a Deep Learning network to include options for initialization techniques, L2 regularization or dropout regularization