Assignment 2:
Neural Networks

Name: Nimrat Bedi
NXB200004
CS6375.002

Part 1
Theoretical Part

Page: ____
Date: ____

ASSIGNMENT 2

Neural Networks

1.1) Revisiting Backpropagation Algorithm

a) tanh activation fn

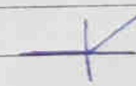
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

derivative of $\tanh(x)$ is:-

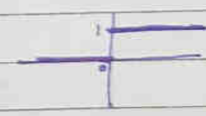
$$\tanh'(x) = 1 - [\tanh(x)]^2 \text{ or } 1/\cosh^2(x)$$

b) ReLu Activation fn

$$\text{Relu}(x) = \begin{cases} x & x > 0 \\ 0 & x <= 0 \end{cases}$$



$$\text{Derivative } \text{Relu}'(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$



Backpropagation Algorithm for any function $f(x)$ here $f(x)$ can be $\tanh(x)$ or $\text{ReLu}(x)$

Error for example d is

$$E_d(w) = \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2 \quad \text{—(1)}$$

t_k = target output we ~~got~~ want as output

o_k = output we got

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} \qquad \eta = \text{learning rate}$$

Page:
Date:

$\frac{\partial E_d}{\partial w_{ji}} = ?$

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}}$$

we know $\frac{\partial net_j}{\partial w_{ji}} = x_{ji}$

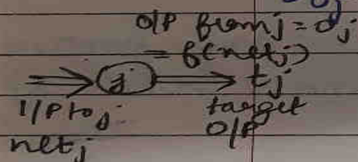
For $\frac{\partial E_d}{\partial net_j}$ we have 2 cases:-

1) j is an output unit

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j}$$

Differenciating (1) with respect to o_j

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\left[\frac{1}{2}\sum_{k \in o/p} (t_k - o_k)^2\right]$$



$$= \frac{\partial}{\partial o_j}\left[\frac{1}{2}(t_j - o_j)^2\right]$$

$$\frac{\partial E_d}{\partial o_j} = -(t_j - o_j)$$

$$\frac{\partial E_d}{\partial o_j} = o_j - t_j$$

second term is $f'(x)$

$f'(x)$ for tanh(x) is $1 - (\tanh^2(x))$

$f'(x)$ for ReLu(x) is $\begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$

$$\frac{\partial o_j}{\partial net_j} = f'(x) = 1 - \tanh(x)^2 \text{ or } \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$

Putting all together we get

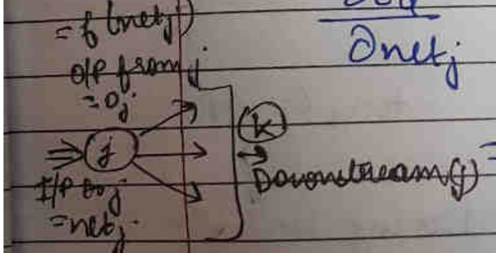
$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\, f'(x) = -\delta_j$$

$$\boxed{\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}} \Rightarrow = \eta \underbrace{(t_j - o_j)\, f'(x)}_{\delta_j} x_{ji}$$

$$\boxed{\Delta w_{ji} = \eta\, \delta_j\, x_{ji}}$$

Case II j is a hidden unit

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream} \frac{\partial E_d}{\partial net_k} \cdot \frac{\partial net_k}{\partial net_j}$$



$$= \sum_{k \in Downstream} -\delta_k \cdot \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

$$\delta = \sum_{k \in Downstream} -\delta_k\, w_{kj}\, f'(x)$$

For $\tanh(x)$ $\quad = -\delta_k w_{kj} (1 - (\tanh(x))^2)$

For ReLU(x) $\quad = -\delta_k w_{kj} (1)$ or $-\delta_k w_{kj} (0)$

$$\begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$

Putting all together

$$\delta_j = f'(x) \sum_{k \in DS} \delta_k w_{kj}$$

Page:
Date:

Plugging in its original eq:

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

$\therefore$ For case I $\delta_j = (t_j - o_j) f'(x)$

For case II $\delta_j = f'(x) \sum_{k \in Downstream} \delta_k w_{kj}$

$f(x)$ is function which can be $\tanh(x)$ or $Relu(x)$

1.2 Gradient Descent

output = $O = w_0 + w_1(x_1 + x_1^2) + \dots \dots + w_n(x_n + x_n^2)$

gradient descent training rule?

Reference used: Tom Mitchell Book (Pg 91-92)

Single unit neuron = Perceptron for which output O is given by:

$$O(\vec{x}) = \vec{w} \cdot \vec{x}$$

we know error ie:

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

D = set of training examples

Weight Update :-

Page:
Date:

We know $w_i^{new} = w_i^{old} - \eta\left(\frac{\partial E_d}{\partial w_i}\right)$

$$\frac{\partial E_d}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}\frac{\partial (t_d - o_d)^2}{\partial w_i}$$

$$= \frac{1}{2}\sum_{d\in D} 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \frac{1\times \cancel{2}}{\cancel{2}}\sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}.\vec{x}_d)$$

$$\frac{\partial E_d}{\partial w_i} = \sum_{d\in D}(t_d - o_d)(-x_{id}) \quad \text{--- (1)}$$

Here output given is :-

$$O = w_0 + w_1(x_1 + x_1^2) + \ldots\ldots + w_n(x_n + x_n^2)$$

We adapt the derivation (1) and consider o/p o.

$$\frac{\partial E}{\partial w_i} = \sum_{x\in X}(\overset{t_d}{\uparrow}{out_x} - \overset{o_d}{\uparrow}{o_x})\frac{\partial}{\partial w_i}\left(out_x - (w_0 + w_1x_1 + w_1x_1^2 + \ldots + w_nx_n + w_nx_n^2)\right) = \sum_{x\in X}(out_x - o_x)(-x_{ix} - x_{ix}^2)$$

Therefore the gradient descent training rule is :-

$$\frac{\partial E}{\partial w_i} = \sum_{x\in X}(out_x - o_x)\frac{\partial}{\partial w_i}\left(out_x - (w_0 + w_1x_{1x} + w_1x_{1x}^2 + \cdots + w_nx_{nx} + w_nx_{nx}^2)\right) = \sum_{x\in X}(out_x - o_x)(-x_{ix} - x_{ix}^2)$$

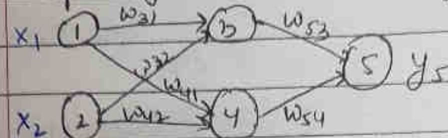
Ans

with learning rate is :

$$\eta\frac{\partial E}{\partial x} = \eta\sum_{x\in X}(out_x - o_x) - (x_{ix} + x_{ix}^2)$$

1.3 Comparing Activation function

Input layer — O/P layer



Hidden layer

Activation function is $h(x)$

a) O/P of neural net y_5 with general activation fn $h(x)$

Output

1,2) $X_1 = i_1 \quad X_2 = i_2$

	Output
3. $net_3 = w_{31} x_1 + w_{32} x_2$	$x_3 = h(net_3)$
4. $net_4 = w_{41} x_1 + w_{42} x_2$	$x_4 = h(net_4)$
5. $net_5 = w_{53} x_3 + w_{54} x_4$	$x_5 = h(net_5)$

b) $X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$

$$w^{(1)} = \begin{pmatrix} w_{3,1} & w_{3,2} \\ w_{4,1} & w_{4,2} \end{pmatrix}$$

$$w^{(2)} = \begin{pmatrix} w_{5,3} & w_{5,4} \end{pmatrix}$$

Output of a neural net in vector form

Input coming in x_3 & x_4

$$\begin{pmatrix} x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} w_{3,1} & w_{3,2} \\ w_{4,1} & w_{4,2} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Output of $\begin{pmatrix} x_3 \\ x_4 \end{pmatrix}$ will be suppose $\begin{pmatrix} y_3 \\ y_4 \end{pmatrix}$

Page: ______
Date: __/__/__

$$\therefore \begin{pmatrix} y_3 \\ y_4 \end{pmatrix} = h\begin{pmatrix} x_3 \\ x_4 \end{pmatrix}$$

Output for y_5 will be

$$y_5 = h[\bar{w}_2 \underbrace{h[\bar{w}_1 \bar{x}]}_{\text{This is output of } y_3, y_4}]$$

Input of y_5 will be

$$x_5 = (w_{5,3} \;\; w_{5,4}) \begin{pmatrix} y_3 \\ y_4 \end{pmatrix}$$

$$\downarrow$$

$$h\begin{bmatrix} x_3 \\ x_4 \end{bmatrix}$$

$\therefore$ Output for y_5 will be

$$y_5 = h[\bar{w}_2 \, h(\bar{w}_1 \bar{x})]$$

c Neural Nets created using the 2 activation fns can generate the same function

Sigmoid $= h_s(x) = \dfrac{1}{1+e^{-x}} = \dfrac{e^{x}}{1+e^{x}}$

Tanh : $h_t(x) = \dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

$$1 - h_s(x) = h_s(-x)$$

$$\text{as } \; 1 - \frac{1}{1+e^{-x}} = \frac{1}{1+e^{x}}$$

For $h_t(x) = \dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

(Adding & subtracting e^{-x}) in numerator $\quad = \dfrac{e^{x}+e^{-x}-2e^{-x}}{e^{x}+e^{-x}}$

$$= 1 - \frac{2e^{-x}}{e^{x}+e^{-x}}$$

Page:
Date:

$$= 1 - \frac{2}{e^{2x}+1}$$

Now from stigmoid perceptive :-

$$1 - h_s(x) = h_s(-x)$$

$$1 - \frac{2}{e^{2x}+1} = 1 - 2h_s(-2x)$$

$$\cancel{1 - 2h_s}$$

$$= 1 - 2(1 - h_s(2x))$$

$$= 1 - 2 + 2h_s(2x)$$

$$\boxed{h_t(x) = 2h_s(2x) - 1}$$

Written clearly relationship between $\tanh(x) = h_t(x)$ and $h_s(x)$ is :-

$$h_t(x) = 2h_s(2x) - 1$$

Output y_s will be $\sigma(net_s)$ for stigmoid and for $\tanh(x)$ activation the output y_s will be $2\sigma(2net_s) - 1$
∴ The output functions are same with the parameters differing only by linear transformations and constants.

1.4 Gradient Descent with Weight Penalty

$$E(\vec{w}) = \frac{1}{2}\sum_{d \in D}\sum_{k \in outputs}(t_{kd} - o_{kd})^2 + \gamma\sum_{ij} w_{ji}^2$$

What is new update rule

Page:
Date:

Normal defination of error is:-

$$E = \frac{1}{2} \sum_{k \in outputs} (t_k - O_k)^2$$

$O_k = \sum w_{ji}$

We know

$$W_{ji}^{(new)} = W_{ji}^{old} - \eta \frac{\partial E}{\partial w_{ji}}$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \left[\frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - O_{kd})^2 \right] + \frac{\partial}{\partial w_{ji}} \left[\gamma \sum_{i,j} w_{ji}^2 \right]$$

$$= \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - O_{kd})(-x_{id}) + 2\gamma \sum w_{ji}$$

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}$$

$$= -\eta \sum \left[(t_{kd} - O_{kd})(-x_{id}) + 2\gamma w_{ji} \right]$$

$$\Delta w_{ji} = \eta \sum \left[(t_{kd} - O_{kd})(x_{id}) - 2\gamma w_{ji} \right]$$

$$\therefore W_{ji}^{new} = W_{ji}^{old} + \Delta w_{ji}$$

$$\downarrow$$

$$\eta \sum \left[(t_{kd} - O_{kd}) x_{id} - 2\gamma w_{ji} \right]$$

with weight penality.

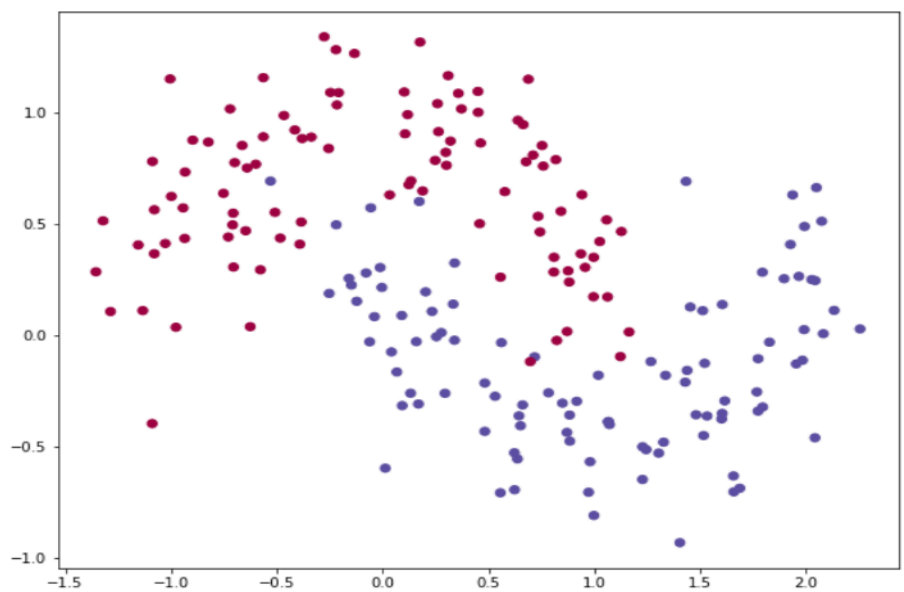
Part 2

Code a neural network having at least one hidden layer.

Dataset used here is from sklearn called as make_moons.
The dataset generated has 2 classes, plotted as red and blue points.

In [3]:
```
np.random.seed(0)
X, y = sklearn.datasets.make_moons(200, noise=0.20)
plt.scatter(X[:,0], X[:,1], s=40, c=y, cmap=plt.cm.Spectral)
```

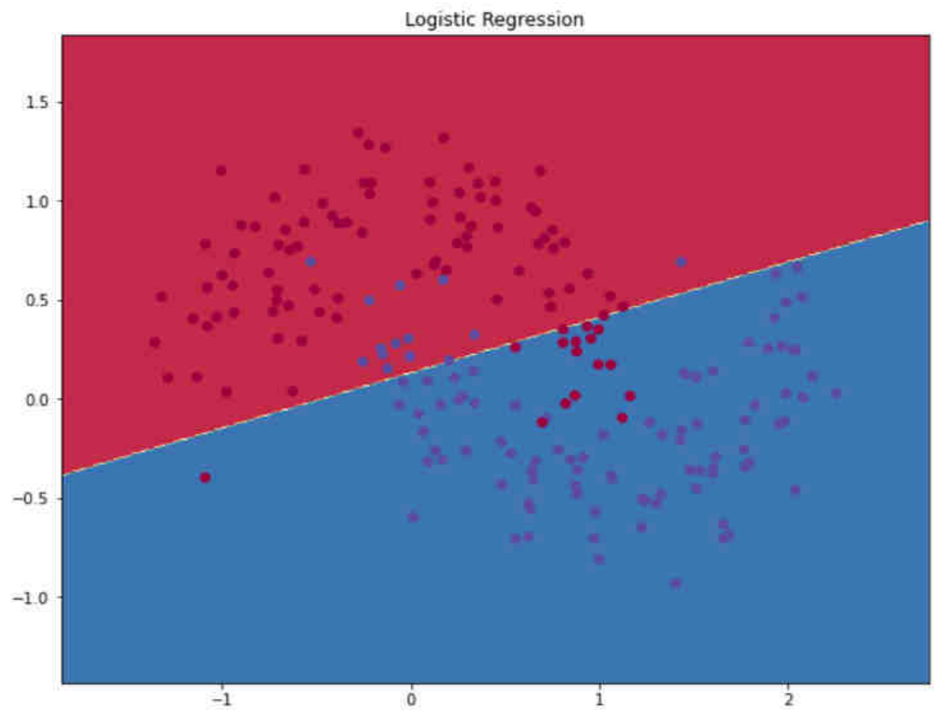
Out[3]: <matplotlib.collections.PathCollection at 0x1177ab610>



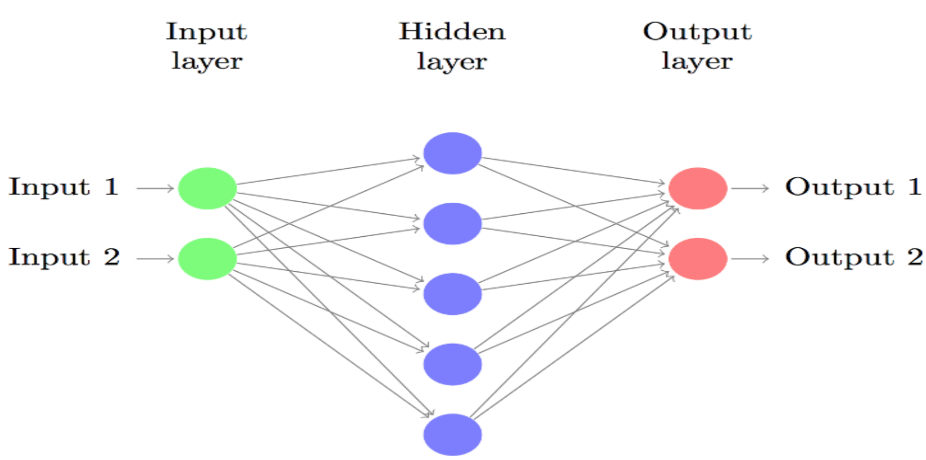
The data is not linearly separable; we can't draw a straight line that separates the two classes.
If we apply Logistic regression we can see below its not separating 2 classes properly, that's why we use neural networks to separate non linear data.

```
In [7]: # Plot the decision boundary
        plot_decision_boundary(lambda x: clf.predict(x))
        plt.title("Logistic Regression")

Out[7]: Text(0.5, 1.0, 'Logistic Regression')
```



Now I have build a 3- layer neural network with one input layer,1 hidden layer and one output layer. The number of nodes in inpiut layer as well as output layer is 2.



The activation function I have used is Tanh but we can use stigmoid as well as ReLu.

The derivative of tanh x is 1-$\tanh^2$ x

Forward propagation to calculate predictions:

z1 = X.dot(W1) + b1
a1 = np.tanh(z1)
z2 = a1.dot(W2) + b2
exp_scores = np.exp(z2)

here z_i is the input of the layer i and a_i is the output of the layer I after the activatio function is applied. W1,b1,w2,b2 are parameter of the network

Applying the backpropagation formula:

$$\delta_3 = \hat{y} - y$$

$$\delta_2 = (1 - \tanh^2 z_1) \circ \delta_3 W_2^T$$

$$\frac{\partial L}{\partial W_2} = a_1^T \delta_3$$

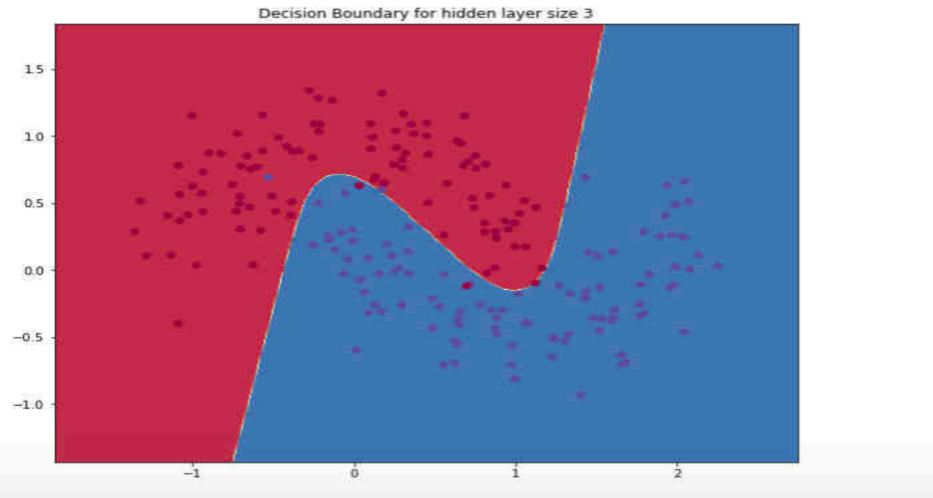
$$\frac{\partial L}{\partial b_2} = \delta_3$$

$$\frac{\partial L}{\partial W_1} = x^T \delta 2$$

$$\frac{\partial L}{\partial b_1} = \delta 2$$

A network with a hidden layer of size 3 :

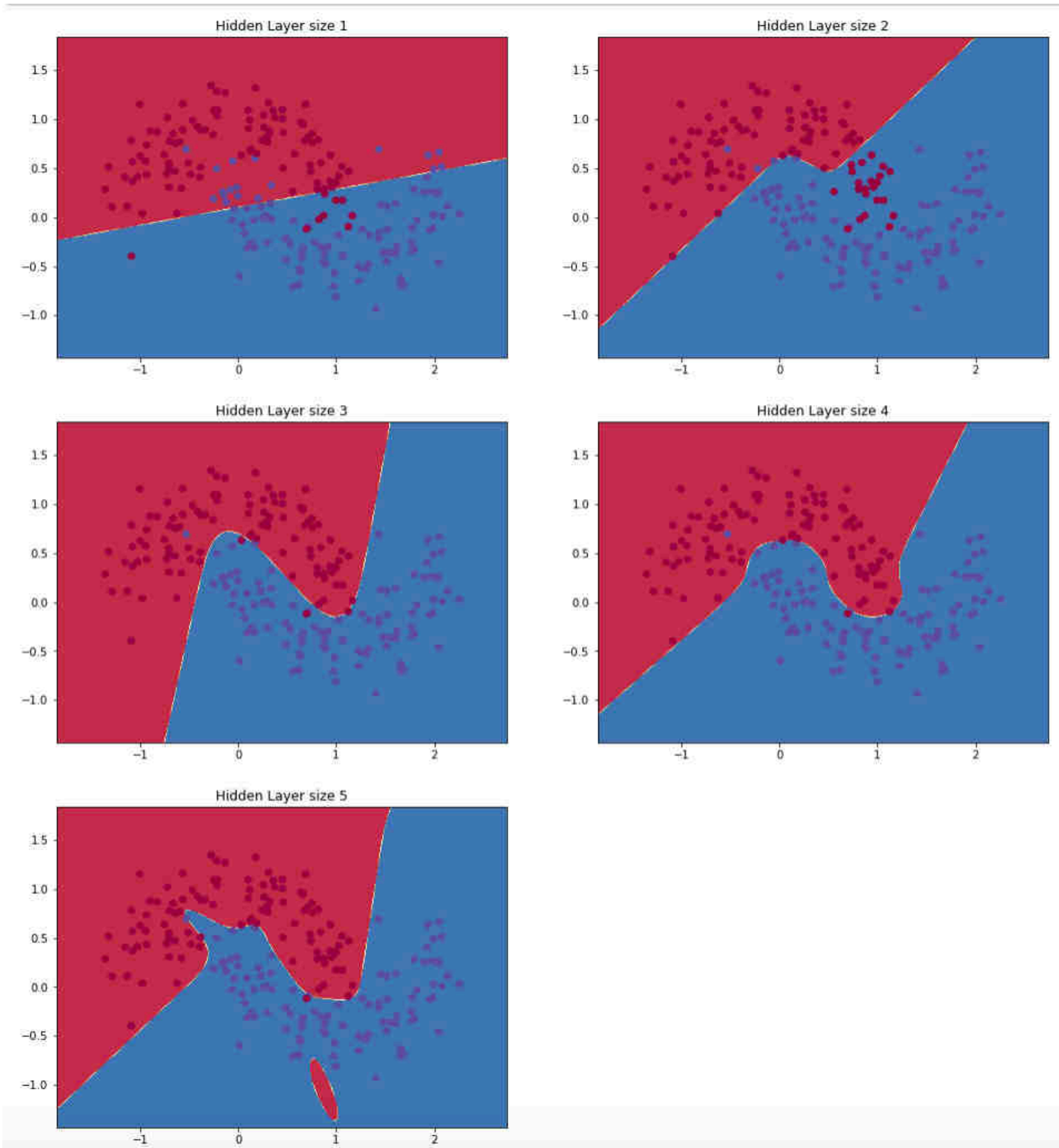
```
Out[12]: Text(0.5, 1.0, 'Decision Boundary for hidden layer size 3')
```



Training error is:

```
Loss after iteration 0: 0.432387
Loss after iteration 1000: 0.068947
Loss after iteration 2000: 0.068901
Loss after iteration 3000: 0.071218
Loss after iteration 4000: 0.071253
Loss after iteration 5000: 0.071278
Loss after iteration 6000: 0.071293
Loss after iteration 7000: 0.071303
Loss after iteration 8000: 0.071308
Loss after iteration 9000: 0.071312
Loss after iteration 10000: 0.071314
Loss after iteration 11000: 0.071315
Loss after iteration 12000: 0.071315
Loss after iteration 13000: 0.071316
Loss after iteration 14000: 0.071316
Loss after iteration 15000: 0.071316
Loss after iteration 16000: 0.071316
Loss after iteration 17000: 0.071316
Loss after iteration 18000: 0.071316
Loss after iteration 19000: 0.071316
```

Varying the hidden layer size:



For stigmoid function we have to change tanh and instead add this

```
def stigmoid(x):
return 1/(1+np.exp(-x))

def stigmoid_der(x):
return stigmoid(x) * (1-stigmoid(x))
```

Similarly do for Relu

```
def  ReLu(x):
return x      (x>0)

def dReLu(x):
return 1    (x>0)
```