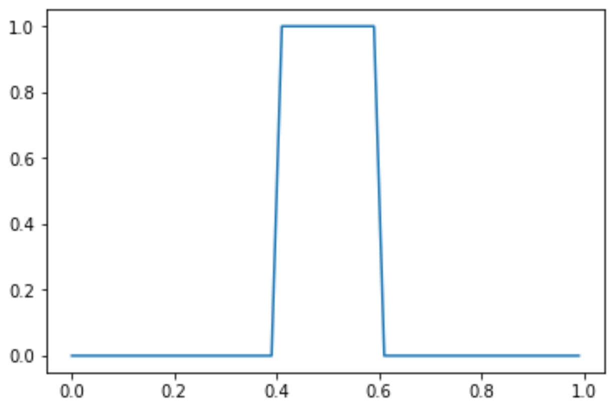
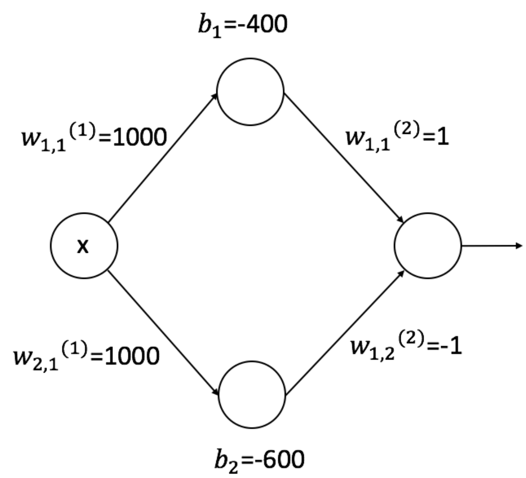
1 Neural networks and Universal Approximation Theorem

1.1

a.





Empirically, I need two hidden neurons to make an approximation of a step function.

b.

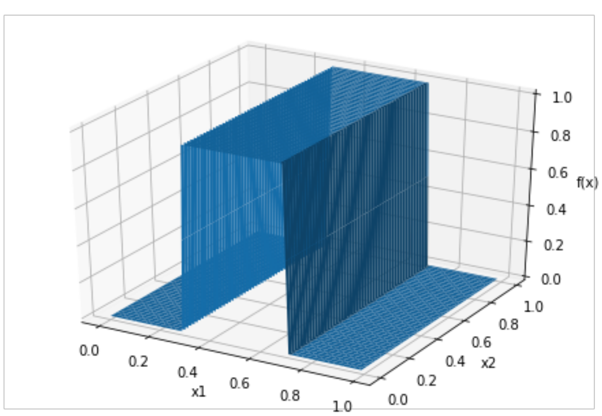
(1) The steepness of the step-up part of the bump is in proportion to $w_{1,1}{}^{(1)}$ and the steepness of the step-down part of the bump is in proportion to $w_{2,1}{}^{(1)}$.

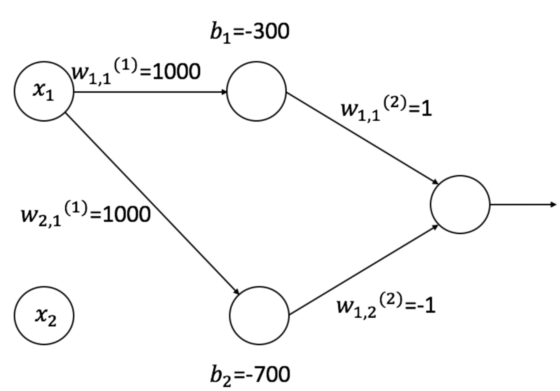
(2) The step-up location is $\frac{-b_1}{w_{1,1}{}^{(1)}}$ and the step-down location is $\frac{-b_2}{w_{2,1}{}^{(1)}}$.

(3) The height of the bump equals to $w_{1,1}{}^{(2)}$ and -$w_{1,2}{}^{(2)}$.

1.2

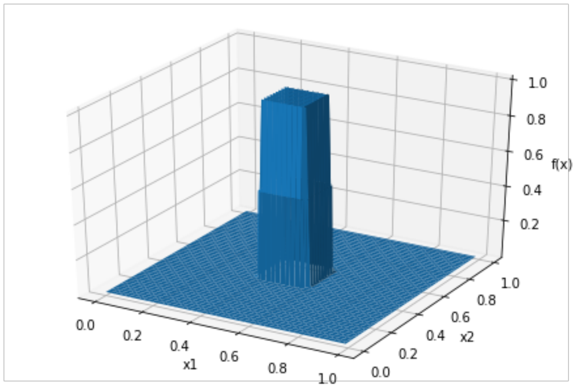
a.

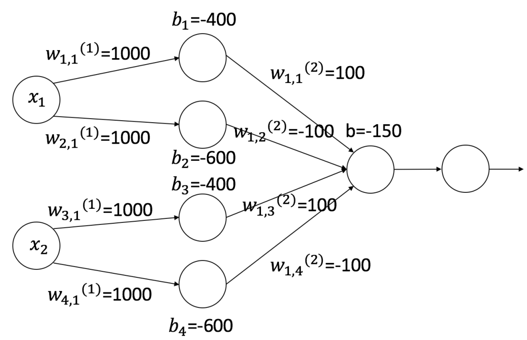




Empirically, I need two 1st-layer hidden neurons to make an approximation of a step function in three-dimension.

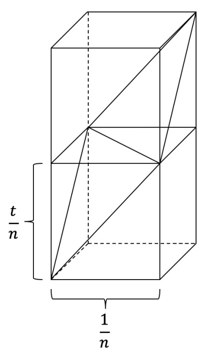
b.





Empirically, I need four 1st-layer hidden neurons to make an approximation of a tower function in three-dimension.

c.



Since the maximum absolute value of the gradient for both directions is t, and the tower function has width 1/n, the height of the tower should be t/n. The error is then the sum of the volume of two pyramids, which is $2\times\frac{1}{3}\times\frac{1}{2}\times\frac{1}{n}\times\frac{1}{n}\times\frac{t}{n}=\frac{t}{3n^3}$

To make sure that the maximum error for each tower function used is ϵ, $\frac{t}{3n^3}<\epsilon$

$$n>\sqrt[3]{\frac{t}{3\epsilon}}$$

So, the minimum number of tower functions that satisfies the conditions is ceil $(\sqrt[3]{\frac{t}{3\epsilon}})$

The greater the gradient limit $\rightarrow$ the larger the error bound $\rightarrow$ the more number of the total neurons is required.

2 EM

a. Use EM algorithm to derive the estimation.

x_i is the number of failures for each visit.

In the E-step, we compute $\mathrm{p}\ (\mathrm{z}=\mathrm{k}|x_i,\theta_k)$

$$\mathrm{p}\ (\mathrm{z}=\mathrm{k}|x_i,\theta_k)=\frac{0.5{\theta_k}^{x_i}(1-\theta_k)^{n-x_i}}{0.5{\theta_1}^{x_i}(1-\theta_1)^{n-x_i}+0.5{\theta_2}^{x_i}(1-\theta_2)^{n-x_i}}$$

$$=\frac{{\theta_k}^{x_i}(1-\theta_k)^{n-x_i}}{{\theta_1}^{x_i}(1-\theta_1)^{n-x_i}+{\theta_2}^{x_i}(1-\theta_2)^{n-x_i}}$$

In the M-step, we compute the θ_k that maximizes $E_{\mathrm{p}\ (\mathrm{z}=\mathrm{k}|x_i,\theta_k)}[log\ \mathrm{p}\ (x_n|\mathrm{z}=\mathrm{k},\theta_k)]$ for each k.

$\sum_{i=1}^{m}E_{\mathrm{p}\ (\mathrm{z}=\mathrm{k}|x_i,\theta_k)}[log\ \mathrm{p}\ (x_n|\mathrm{z}=\mathrm{k},\theta_k)]$=$\sum_{i=1}^{m}\mathrm{p}\ (\mathrm{z}=1|x_i,\theta_1)\log[\binom{\mathrm{n}}{x_i}{\theta_1}^{x_i}(1-\theta_1)^{n-x_i}]+\mathrm{p}\ (\mathrm{z}=2|x_i,\theta_2)\log[\binom{\mathrm{n}}{x_i}{\theta_2}^{x_i}(1-\theta_2)^{n-x_i}]$

If we take the derivative of the expected value above with regard to θ_k and equate it to zero, we can get:

$$\theta_k=\frac{\sum_{i=1}^{m}\mathrm{p}\ (\mathrm{z}=\mathrm{k}|x_i,\theta_k)\frac{x_i}{n}}{\sum_{i=1}^{m}\mathrm{p}\ (\mathrm{z}=\mathrm{k}|x_i,\theta_k)}$$

b. EM algorithm implementation:

```
theta1:  0.5, theta2:  0.5
theta1:  0.5433333333333333, theta2:  0.2985264202060478
theta1:  0.7933298011576458, theta2:  0.2933333333333334
theta1:  0.7933333333333333, theta2:  0.2933333333333334
theta1:  0.7933333333333333, theta2:  0.2933333333333334
```

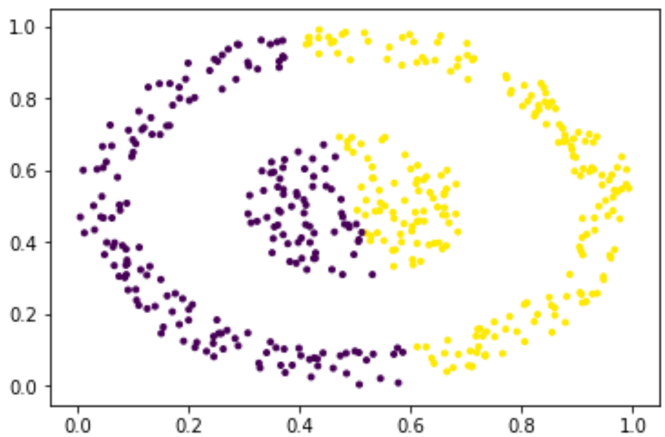
Three iterations before convergence.

3 Clustering

a. k-means algorithm implementation

k was set to 2:

The group assignment for each point in the dataset is shown in the following figure:



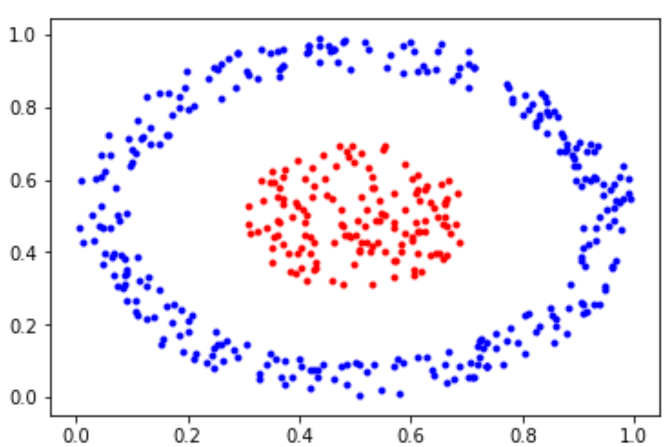
The mean for each group is as follows:

```
[[0.28639575 0.43987188]
 [0.71934657 0.53760382]]
```

b. hierarchical agglomerative clustering algorithm implementation:

k was set to 2:

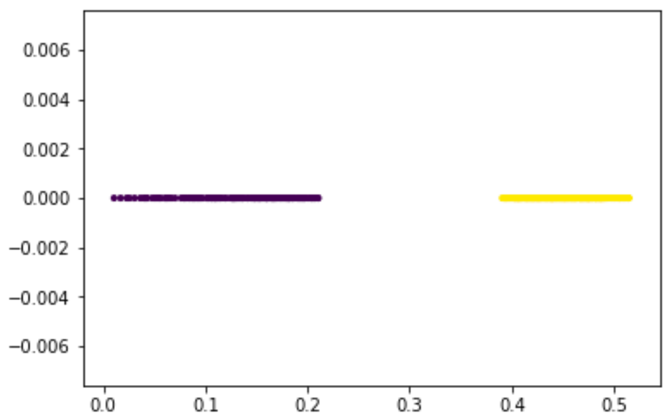
The group assignment for each point in the dataset is shown in the following figure:



c. From the two figures above, we can see hierarchical agglomerative clustering algorithm performs better than k-means algorithm on this dataset. The reason is that this dataset is well-separated and the shape of clusters are not spherical. So, hierarchical agglomerative clustering is useful here but k-means is not a good choice.

d. Zero-center the dataset and transform the Cartesian coordinates (x, y) of the dataset to polar coordinates (theta, radius). Append a column vector of zeros to the radius vector (also a column vector) gives the transformed dataset. Run k-means algorithm on the transformed dataset.

The classification result plotted on the transformed dataset:



The classification result plotted on the original dataset: