Compsci 571 HW6

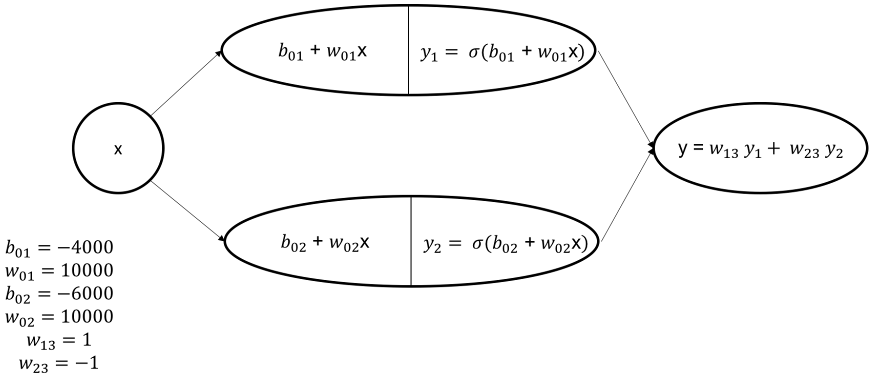
Yilin Gao (yg95)

April 10, 2018

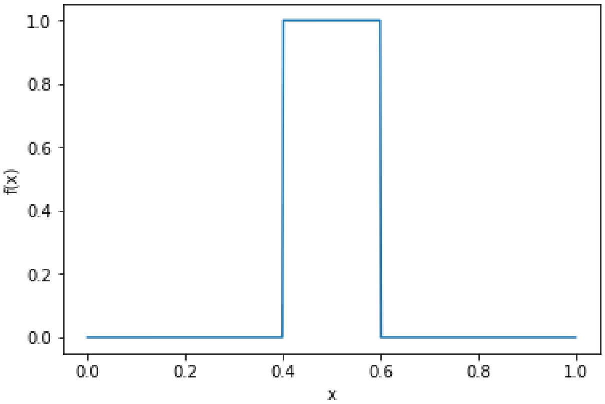
1 Neural Networks and Universal Approximation Theorem

1.1

(a) The NN architecture is like following:



The implementation is in `q1.ipynb`. The approximated function is as following:

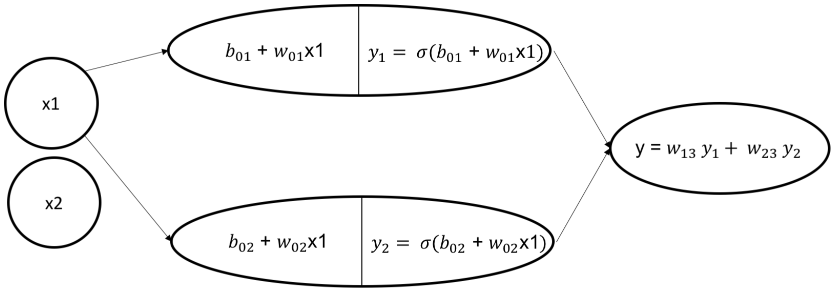


The minimal number of hidden neurons is 2, because the bump is a combination of 2 step functions, and each neuron (with the sigmoid activation function) is able to approximate one step function with any given step direction, location and height.

(b) In the NN, w_{01} determines the steepness of the step-up part of the bump, w_{02} determines the steepness of the step-down part of the bump. $-\frac{b_{01}}{w_{01}}$ determines the step-up location, $-\frac{b_{02}}{w_{02}}$ determines the step-down location. And w_{13} and w_{23} determine the height of the bump.

1.2

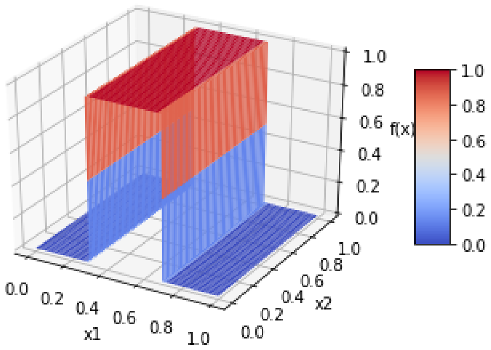
(a) The NN architecture is like following:


x1
x2
$b_{01} + w_{01}$x1
$y_1 = \sigma(b_{01} + w_{01}$x1)
$b_{02} + w_{02}$x1
$y_2 = \sigma(b_{02} + w_{02}$x1)
y = $w_{13}\, y_1 + w_{23}\, y_2$

$b_{01} = -3000$
$w_{01} = 10000$
$b_{02} = -7000$
$w_{02} = 10000$
$w_{13} = 1$
$w_{23} = -1$

There is no edge between the input cell for x_2 and the hidden layer.

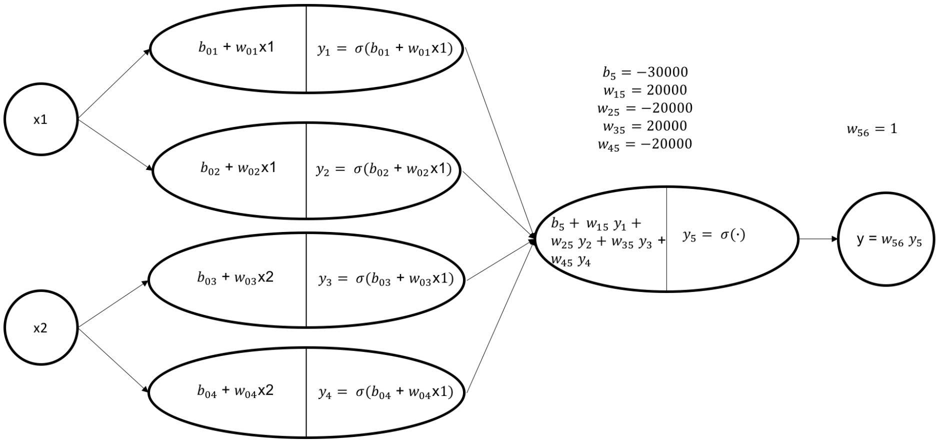
The implementation is in `q1.ipynb`. The approximated function is as following:



The minimal number of hidden neurons is 2. Because the 2D bump is only in the direction of x_1, so we can regard it same as the one in part 1, and ignore the input x_2.

(b) The NN architecture is like following:

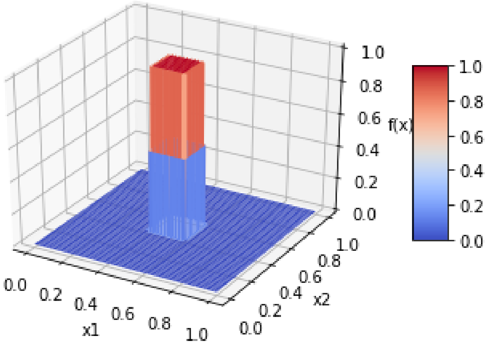
$b_{01} = -4000$
$w_{01} = 10000$
$b_{02} = -6000$
$w_{02} = 10000$
$b_{03} = -4000$
$w_{03} = 10000$
$b_{04} = -6000$
$w_{04} = 10000$


x1
x2
$b_{01} + w_{01}$x1
$y_1 = \sigma(b_{01} + w_{01}$x1)
$b_{02} + w_{02}$x1
$y_2 = \sigma(b_{02} + w_{02}$x1)
$b_{03} + w_{03}$x2
$y_3 = \sigma(b_{03} + w_{03}$x1)
$b_{04} + w_{04}$x2
$y_4 = \sigma(b_{04} + w_{04}$x1)
$b_5 + w_{15}\, y_1 + w_{25}\, y_2 + w_{35}\, y_3 + w_{45}\, y_4$
$y_5 = \sigma(\cdot)$
y = $w_{56}\, y_5$

$b_5 = -30000$
$w_{15} = 20000$
$w_{25} = -20000$
$w_{35} = 20000$
$w_{45} = -20000$

$w_{56} = 1$

The implementation is in `q1.ipynb`. The approximated function is as following:



The minimal number of hidden neuron in the first layer is 4. Because the 2D bump is a combination of a bump in x_1 direction and another bump in x_2 direction, and each bump needs 2 hidden neurons according to previous question.

(c) For a grid (x_1, x_2) such that $x_1 \in [x_1^*, x_1^* + \frac{1}{n}]$ and $x_2 \in [x_2^*, x_2^* + \frac{1}{n}]$, assume the tower function we use to approximate has height $h^* = f(x_1^*, x_2^*)$.

Because the grid is small enough, we can assume for any fixed x_2^0, $f(x_1, x_2^0)$ is linear with $x_1 \in [x_1^*, x_1^* + \frac{1}{n}]$, and for any fixed x_1^0, $f(x_1^0, x_2)$ is linear with $x_2 \in [x_2^*, x_2^* + \frac{1}{n}]$. So we have: $f(x_1, x_2^0) = \frac{\partial f(x_1, x_2^0)}{\partial x_1}(x_1 - x_1^*)$ for $x_1 \in [x_1^*, x_1^* + \frac{1}{n}]$, and $f(x_1^0, x_2) = \frac{\partial f(x_1^0, x_2)}{\partial x_2}(x_2 - x_2^*)$ for $x_2 \in [x_2^*, x_2^* + \frac{1}{n}]$.

Because the maximum absolute value of the gradient for both directions is t, $|\frac{\partial f(x_1, x_2^0)}{\partial x_1}| \leq t$, and $|\frac{\partial f(x_1^0, x_2)}{\partial x_2}| \leq t$.

The approximation error on the grid $= \int_{x_1^*}^{x_1^* + \frac{1}{n}} \int_{x_2^*}^{x_2^* + \frac{1}{n}} [f(x_1, x_2) - h^*] dx_1 dx_2$. If we view it geometrically, we can easily find under this setting, it is a regular 3D object that we can calculate its volume with know data. When $|\frac{\partial f(x_1, x_2^0)}{\partial x_1}| = t$, and $|\frac{\partial f(x_1^0, x_2)}{\partial x_2}| = t$, its maximal volume is $\frac{t}{n^3}$.

So we have $\frac{t}{n^3} \leq \epsilon$, $n \geq \sqrt[3]{\frac{t}{\epsilon}}$. So the minimum number of tower functions to make such approximation is $n^2 = \sqrt[3]{\frac{t^2}{\epsilon^2}}$.

One tower function has 5 hidden neurons in total. So the total number of required hidden neurons $= 5\sqrt[3]{\frac{t^2}{\epsilon^2}}$. When the gradient limit is larger, the total required number of hidden neurons is also larger. When the error bound is smaller, the total required number of hidden neurons is smaller.

2 EM

(a) Notations used:

i: index for laundry visit, $i \in \{1, \dots, m\}$,

z_i: the machine used in visit i, $z_i \in \{1, 2\}$,

$\vec{X}_i$: random variable to represent possible working conditions of the chosen machine in visit i, $\vec{X}_i \in R^n$, and its each value equals to 0 (not broken) or 1 (broken),

$\vec{x}_i$: the observed working conditions of the chosen machine in visit i, $\vec{x}_i \in R^n$, and its each value equals to 0 (not broken) or 1 (broken),

w_k: the probability to use machine k in one laundry visit, $k \in \{1, 2\}$, and $\sum_{k=1}^{2} w_k = 1$,

θ_k: the probability that machine k breaks down, $k \in \{1, 2\}$.

So we could have: $P(z_i = k | w) = w_k$.

And we could see that given $z_i = k$ and θ_k, $\vec{X}_i$ follows multivariate Bernoulli distribution, $P(\vec{X}_i = \vec{x}_i | z_i = k, \theta_k) = \prod_{j=1}^{n} \theta_k^{x_{ij}} (1 - \theta_k)^{(1 - x_{ij})}$.

The EM algorithm works as following:

```
Initialize $\vec{w}$ and $\vec{\theta}$ as $\vec{w}_0$ and $\vec{\theta}_0$;
for $t = 0, 1, \ldots$ do
    E step: Compute $P(z_i = k|\vec{x}_i, \vec{w}_t, \vec{\theta}_t) =: \gamma_{ik}^{t+1}$ ;
    M step: Maximize the auxilary function $A(\vec{w}, \vec{\theta}, \vec{w}_t, \vec{\theta}_t)$ by choosing $\vec{w}_{t+1}$ and $\vec{\theta}_{t+1}$.
    if $\vec{w}$ and $\vec{\theta}$ have converged then
        Break the for loop;
    end
end
```

So in t'th E step:

$$\gamma_{ik}^{t+1} =: P(z_i = k|\vec{x}_i, \vec{w}_t, \vec{\theta}_t) = \frac{P(z_i=k, \vec{X}_i=\vec{x}_i|\vec{w}_t,\vec{\theta}_t)}{P(\vec{X}_i=\vec{x}_i|\vec{w}_t,\vec{x}_t)} = \frac{P(\vec{X}_i=\vec{x}_i|z_i,\vec{w}_i,\vec{\theta}_i)P(z_i=k|\vec{w}_t,\vec{\theta}_t)}{P(\vec{X}_i=\vec{x}_i|\vec{w}_t,\vec{\theta}_t)} = \frac{[\prod_{j=1}^n \theta_{kt}^{x_{ij}}(1-\theta_{kt})^{(1-x_{ij})}]w_{kt}}{\sum_{k=1}^2([\prod_{j=1}^n \theta_{kt}^{x_{ij}}(1-\theta_{kt})^{(1-x_{ij})}]w_{kt})}.$$

In t'th M step:

$$A(\vec{w}, \vec{\theta}, \vec{w}_t, \vec{\theta}_t) = \sum_{i=1}^m \sum_{k=1}^2 \gamma_{ik}^{t+1} \log P(\vec{X}_i = \vec{x}_i, z_i = k|\vec{w}, \vec{\theta}) = \sum_{i=1}^m \sum_{k=1}^2 \gamma_{ik}^{t+1} \log P(z_i = k|\vec{w}, \vec{\theta})P(\vec{X}_i = \vec{x}_i|z_i = k, \vec{w}, \vec{\theta}) = \sum_{i=1}^m \sum_{k=1}^2 \gamma_{ik}^{t+1} \log[w_{kt} \prod_{j=1}^n \theta_k^{x_{ij}}(1-\theta_k)^{(1-x_{ij})}] = \sum_{i=1}^m \sum_{k=1}^2 \gamma_{ik}^{t+1}[\log w_{kt} + \sum_{j=1}^n (x_{ij} \log \theta_k + (1 - x_{ij}) \log(1 - \theta_k))]$$

We need to maximize $A(\vec{w}, \vec{\theta}, \vec{w}_t, \vec{\theta}_t)$ subject to the constraint that $w_{1(t+1)} + w_{2(t+1)} = 1$.

With the Lagrangian method, set the partial derivatives of $A(.)$ w.r.t. θ_k and w_k equal to 0, we could get $\theta_{k(t+1)} = \frac{\sum_{i=1}^m \gamma_{ik}^{t+1} \sum_{j=1}^n x_{ij}/n}{\sum_{i=1}^m \gamma_{ik}^{t+1}}$, and $w_{k(t+1)} = \frac{\sum_{i=1}^m \gamma_{ik}^{t+1}}{m}$.

(b) See the implementation in `q2.ipynb`. For the specific simulated data left in the notebook, the estimated $\hat{\theta}_1 = 0.77$ and estimated $\hat{\theta}_2 = 0.34$. The estimation changes as the simulated data change.

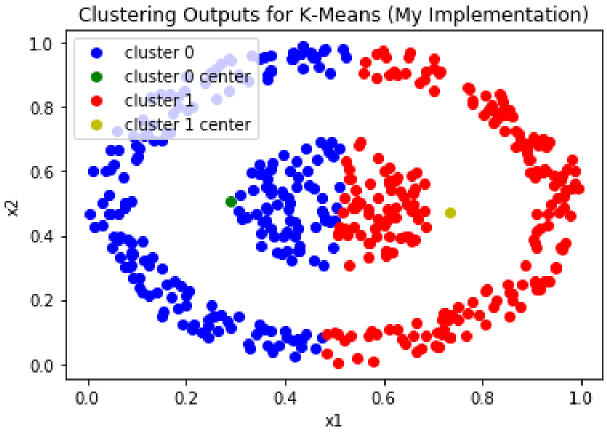
3 Clustering

(a) See the implementation in `q3.ipynb`.

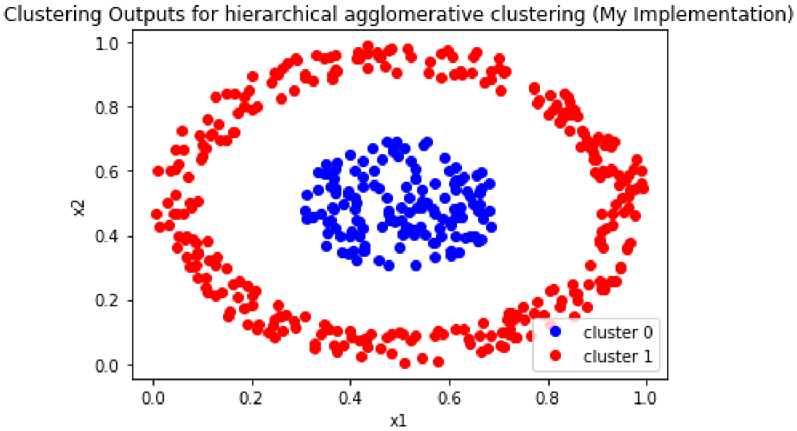
(b) See the implementation in `q3.ipynb`.

(c) The empirical clustering results on the dataset with both algorithms are as following:

K-Means:



Hierarchical Agglomerative Clustering:



From the visualization, we can tell the hierarchical agglomerative clustering algorithm performs better.

On this specific dataset, the possible reason for the discrepancy is that K-Means assumes the variance of each X variable is spherical. However, the given data doesn't satisfy this assumption.

(d) Possible preprocessing on data to make K-Means perform better: move the data to center at $[0, 0]$, and turn the data represented by cartesian coordinates into polar coordinates ($[\rho, \theta]$). Then run K-Means on the ρ values of all data. Because points in the same cluster all have similar ρ values.