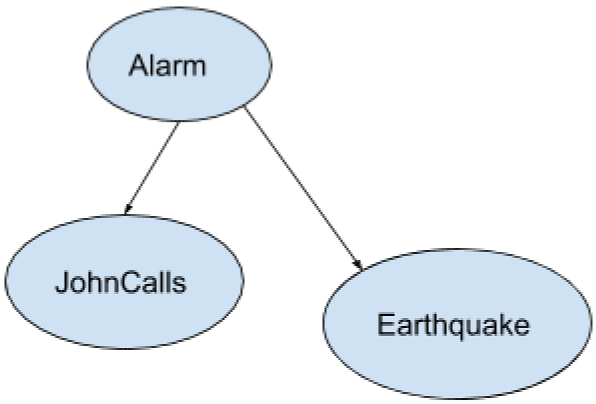
1. P(~JohnCalls, MaryCalls, Alarm,~Earthquake, Burglary)
   = P(~J, M, A, ~E, B) //simplify variable names
   = P(~J|Parent(J))P(M|Parent(M))P(A|Parent(A))P(~E)P(B)
   = P(~J|A)P(M|A)P(A|~E^B)P(~E)P(B)
   = 0.1 * 0.7 * 0.94 * 0.998 * 0.001
   = 0.0000656684
2.

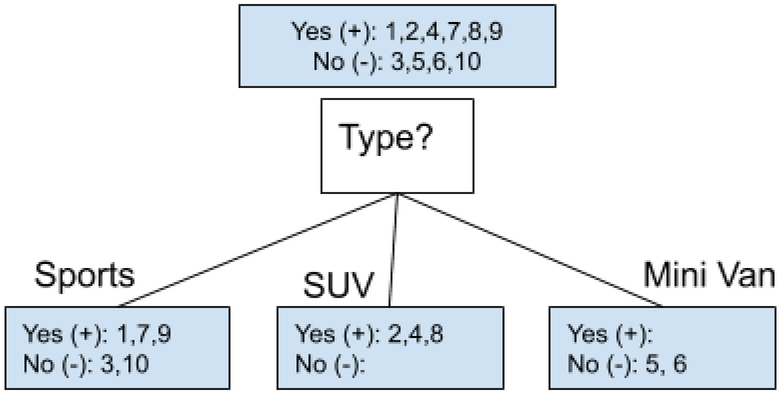


   Node ordering is Alarm, JohnCalls, Earthquake

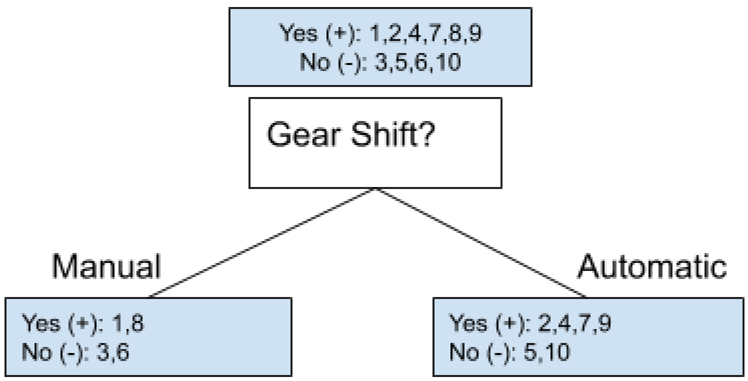
Edges	Explanation
Alarm→JohnCalls	P(JohnCalls|Alarm) != P(JohnCalls) In the original belief network, it is clear that the Alarm has an effect on whether or not JohnCalls. Therefore, this edge is required to satisfy conditional independence.
Alarm→Earthquake	Conditional Independence: P(Earthquake|Alarm, JohnCalls) = P(Earthquake|Alarm) Based on the original belief network, whether or not john calls is based on the alarm going off. So whether or not JohnCalls does not affect P(Earthquake|Alarm). That is also why there is not an edge from JohnCalls to Earthquake.

3. P(Burglary|JohnCalls) > P(Burglary|JohnCalls, Earthquake)
   => P(B|J) > P(B|J,E) //simplify variable names
   => B is a cause, J is an effect, and E is another cause
   => This is Intercausal inferences (explaining away)
   Therefore, the inequality holds because the “cause is already found”
4. Depth-one decision trees

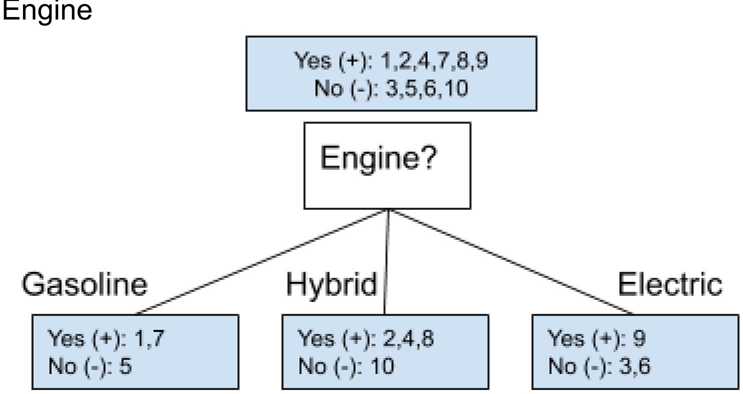
a. Type



b. Gear Shift



c. Engine

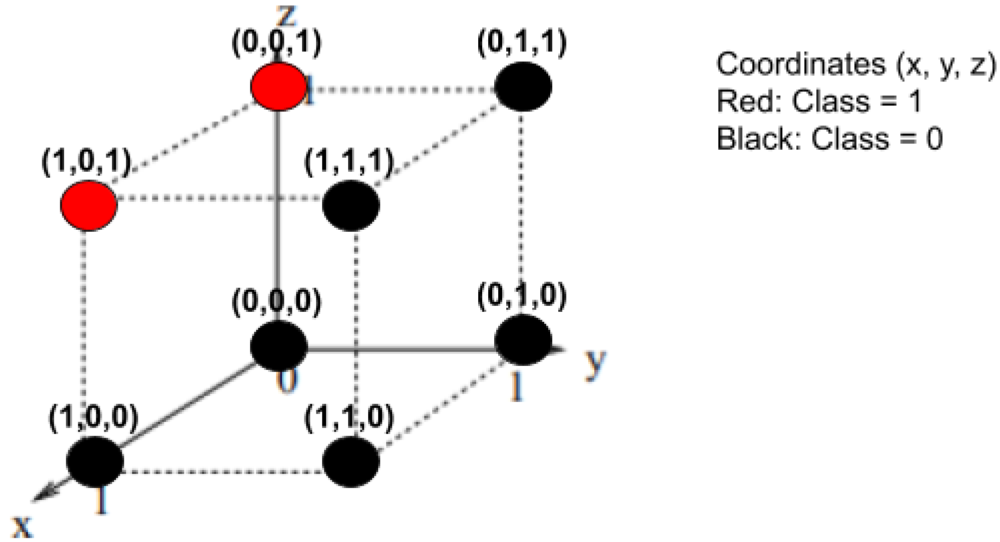


5. Information gain: Entropy(parent) - [average entropy(children)]
   Entropy(parent) = sum for i in C [-Pi*log2(Pi)]
   = -Pyes*log2(Pyes) + -Pno*log2(Pno)
   = -0.6*log2(0.6) + -0.4*log2(0.4) = 0.971
   a. Gain(E, Type?) = 0.971 - sum(|Ev| * Entropy(Ev)/|E|)
      = 0.971 -
      (Esports*Entropy(Esports)+Esuv*Entropy(Esuv)+Evan*Entropy(Esuv))/|E|
      = 0.971 - (5*0.971 + 3*0 + 2*0)/10
      = **0.486**

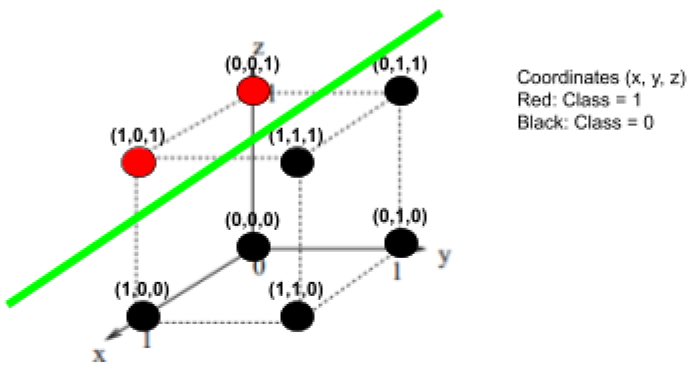
    b. Gain(E, Gear Shift?) = 0.971 - sum(|Ev| * Entropy(Ev)/|E|)
    = 0.971 - (4*1 + 6*0.918)/10
    = **0.020**
    c. Gain(E, Engine?) = 0.971 - sum(|Ev| * Entropy(Ev)/|E|)
    = 0.971 - (3*0.918 + 4*0.811 + 3*0.918)/10
    = **0.0958**
6. I would choose the "Type" attribute first because it has the maximal information gain out of all attributes as calculated above. In terms of a decision tree, we want to make as few tests before reaching a decision, so by maximizing information gain we decrease entropy the most with each test and ideally decrease the number of tests before reach a decision.
7. (1) Program output for "Info" was 0.485475, "Gear Shift" was 0.0199731, and "Engine" was 0.0954618. This confirms my calculations.
   (2) Program output for attributes table slide06, page 9

Attribute	Info Gain
Alt	0.0000
Bar	0.0000
Fri	0.0207208
Hun	0.19571
Pat	0.540852
Price	0.19571
Rain	0.0000
Res	0.0207208
Type	0.0000
Est	0.207519

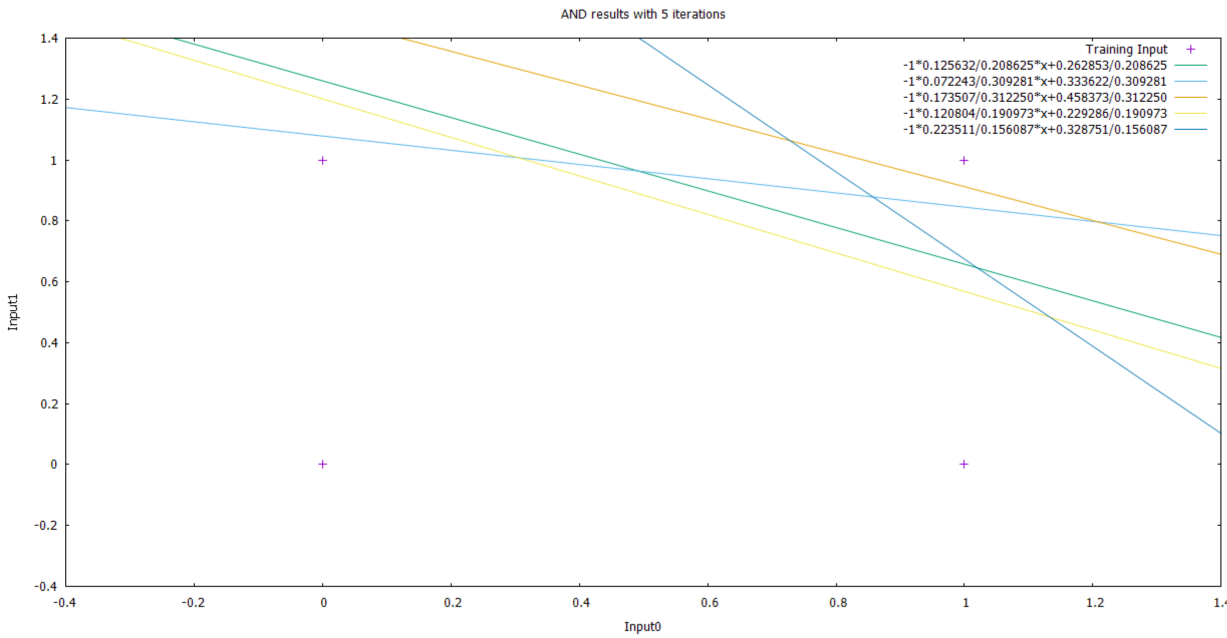
8. (1) Yes
   (2)



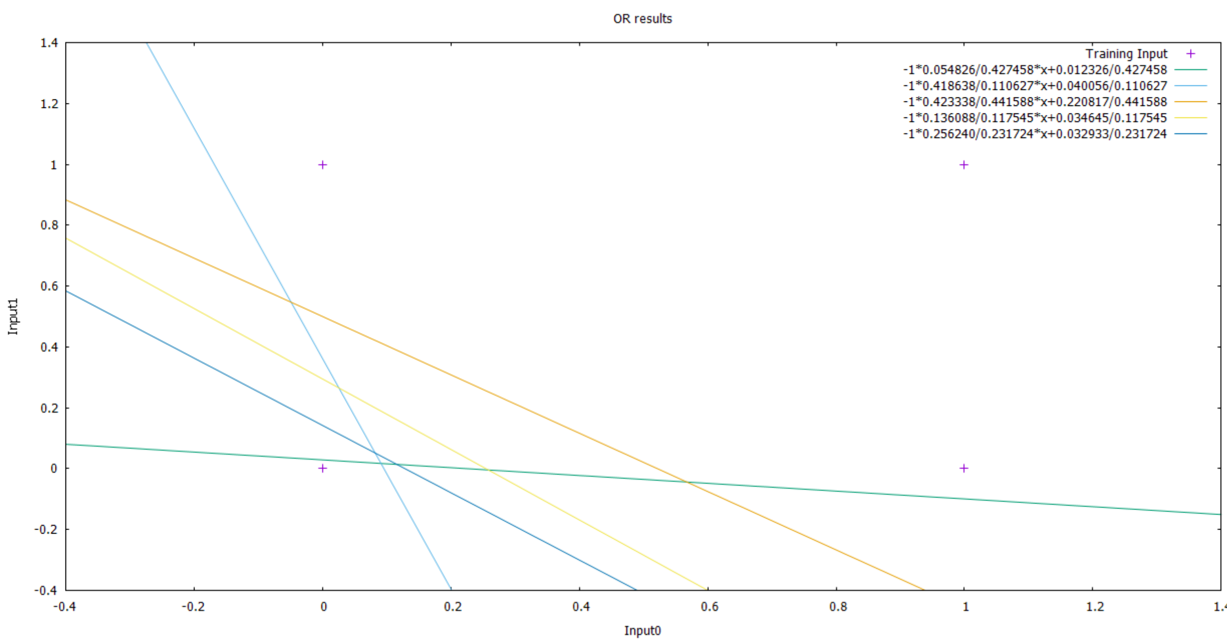
(3) A single perceptron unit can solve this classification problem because the points are linearly-separable. We can draw a flat plane to separate red and black dots. The plane would be parallel to the line created by the red dots, and at an angle. This drawing illustrates this example:



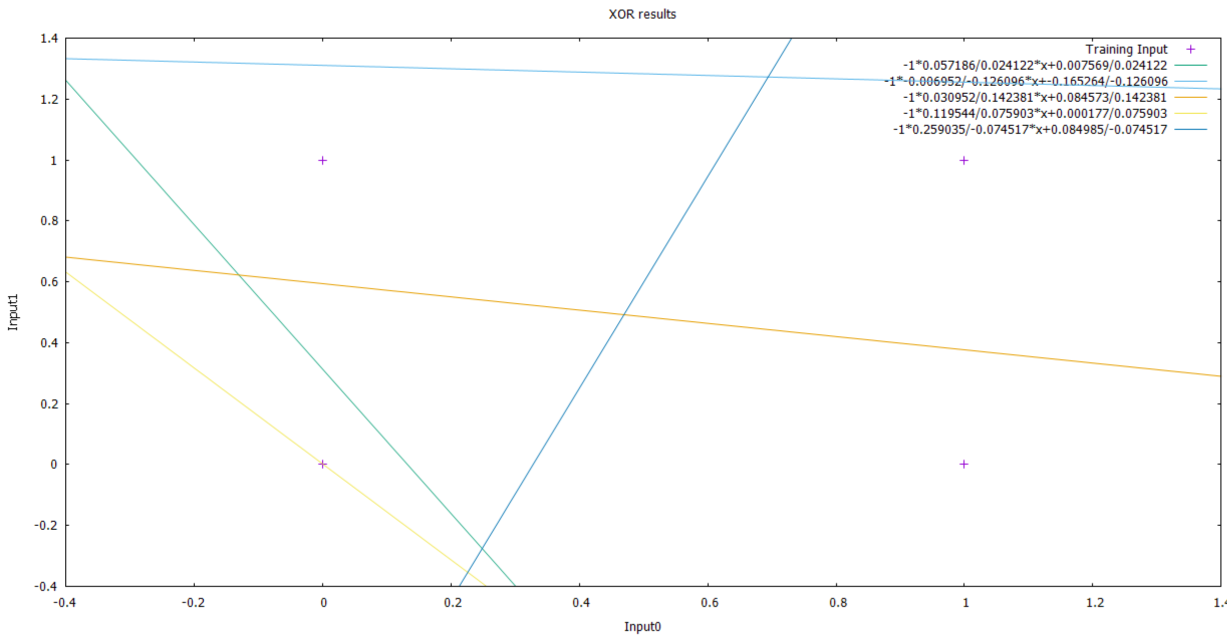
9. (2) You can also view these pictures under the project director “question9/results”.
   - note that plotting was done with Gnuplot and on Windows machine. In order to run the C++ code you must have Gnuplot installed and using Windows OS.
   AND Results



OR results:



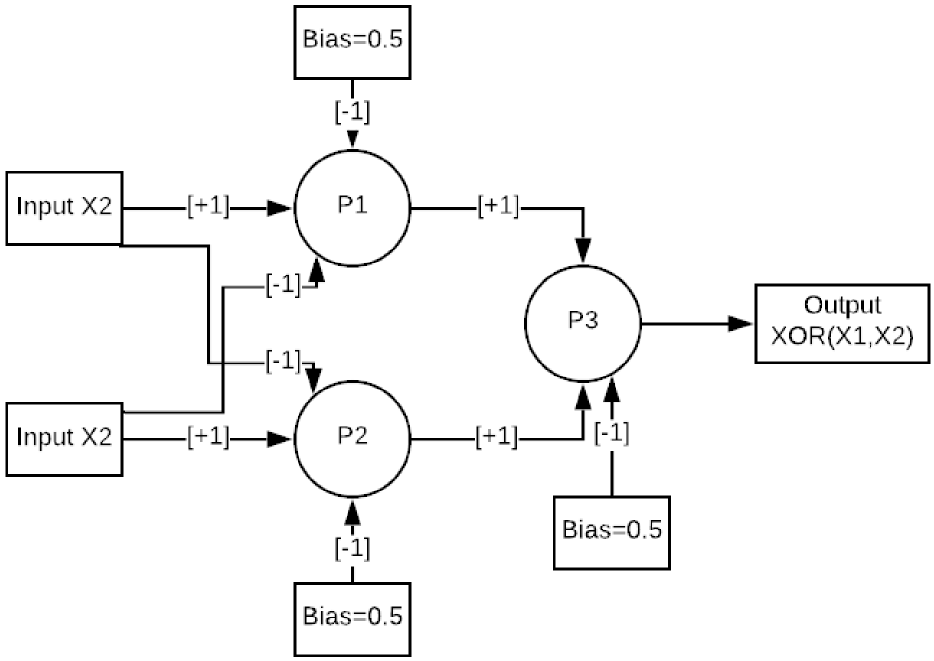
XOR Results (100 iterations max)



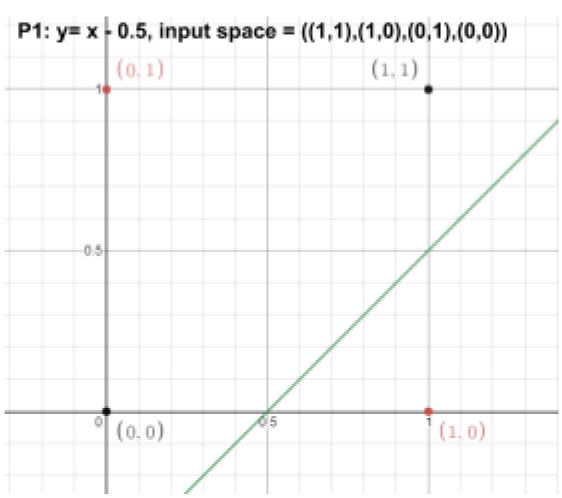
(3) For AND and OR, it is possible. The weights and bias are initialized randomly between -0.5 and 0.5. This means that the weights for AND can be randomly intialized as bias = -0.5 and weight1 = 0.3 and weight2 = 0.3, which will result in no errors. Similarly, if the weights and bias for OR are randomly initialized as bias = -0.3 and weight1 = 0.4 and weight2 = 0.4, which will also result in no errors.

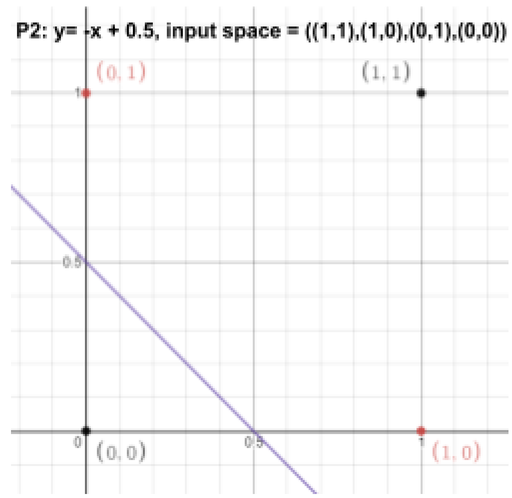
This is heavily implementation-dependent. Note that if the weights and bias were only initialized randomly to a positive range like [0, 1], then it is not possible because all and biases weights will be greater than or equal to 0, which will always result in an output of 1!

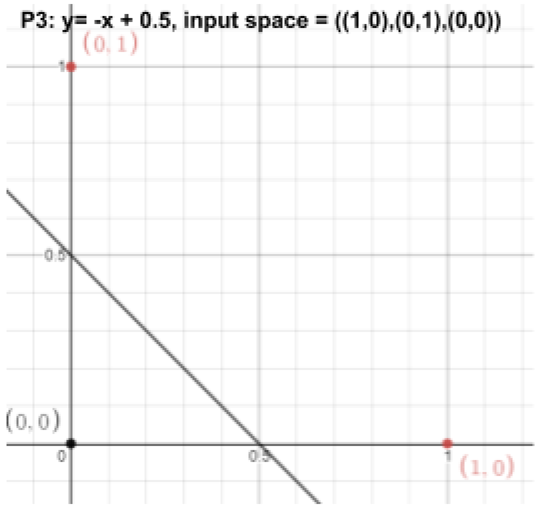
10. (1) Neural Network Topology



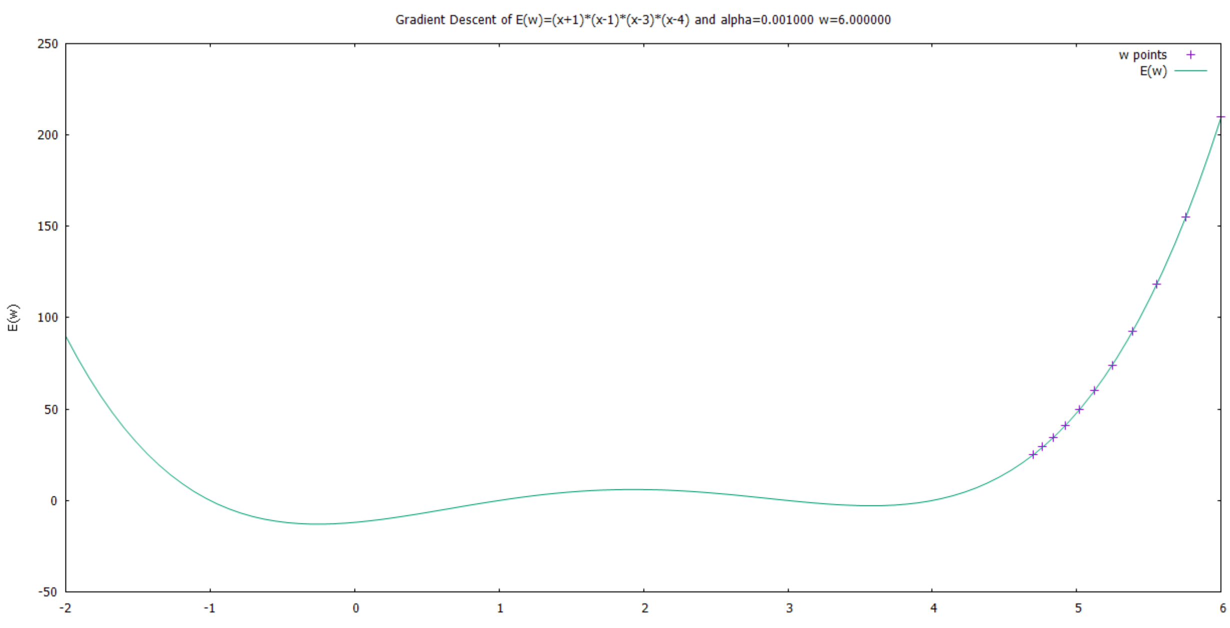
(2) Decision boundary



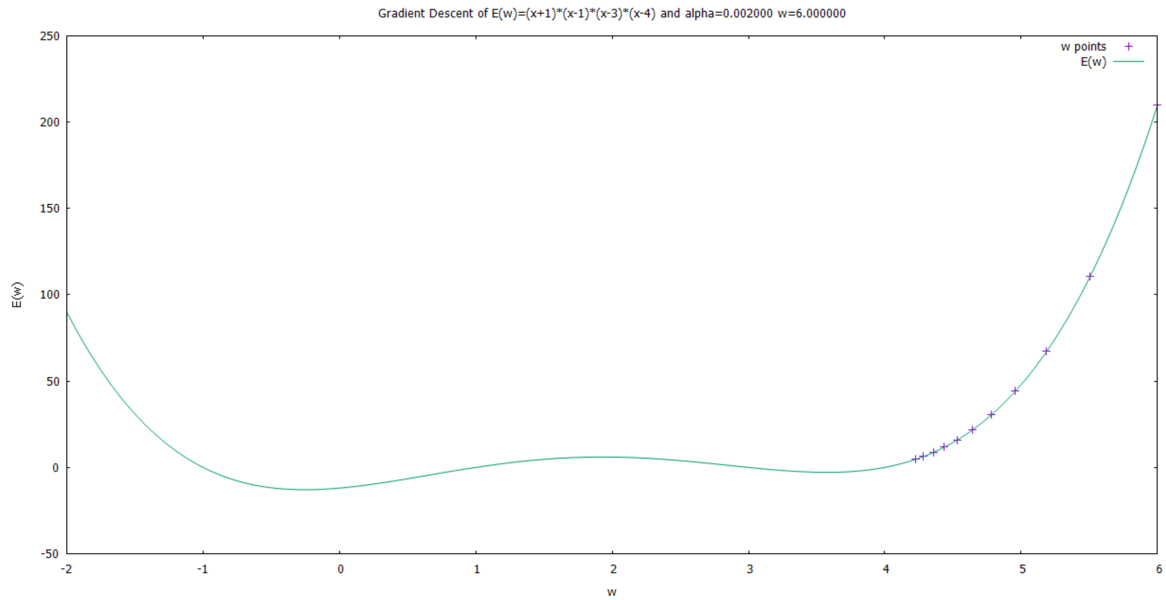




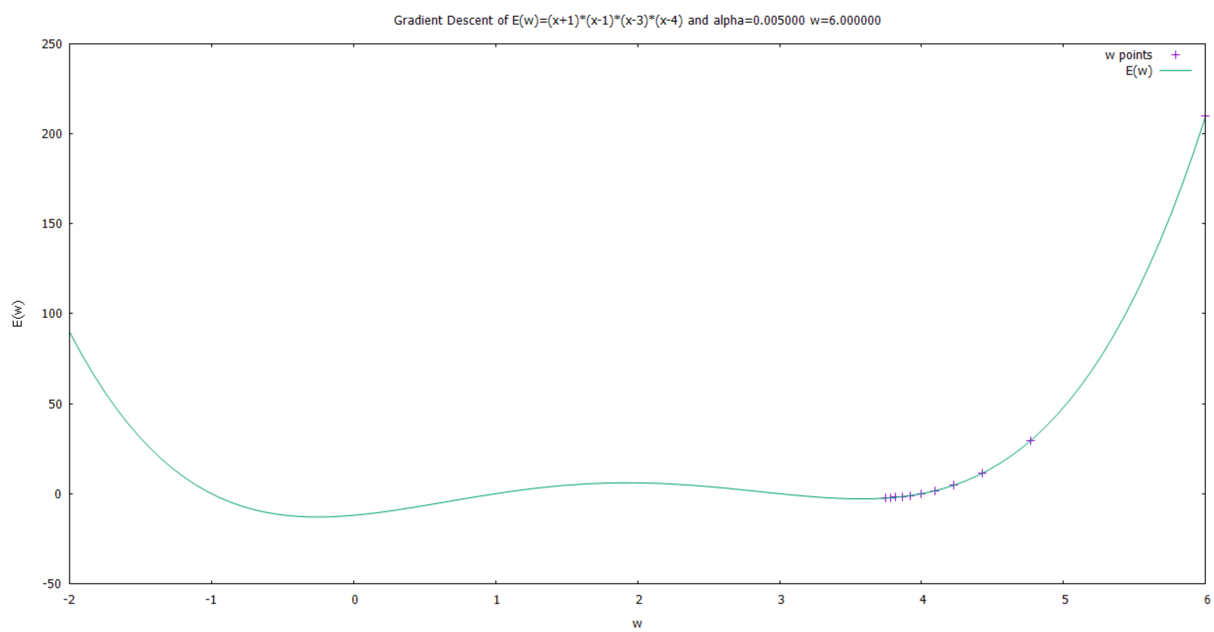
11. Given error function E(w) = (w+1)(w-1)(w-3)(w-4)
    => E(w) = w^4 - 7 w^3 + 11 w^2 + 7 w - 12 // simplify
    => dE/dw = 4w^3 - 21w^2 + 22w + 7 // deriviative
    => **∆w = −α * (4w^3 - 21w^2 + 22w + 7)** // plug into ∆w = −α*dE/dw
12. Plots:
    The pictures can also be found under "question12/results"
    w = 6.0 // this is a fixed value
    Experimented with different learning rates = [0.001, 0.002, 0.005, 0.01]
    Graph 1: alpha = 0.001



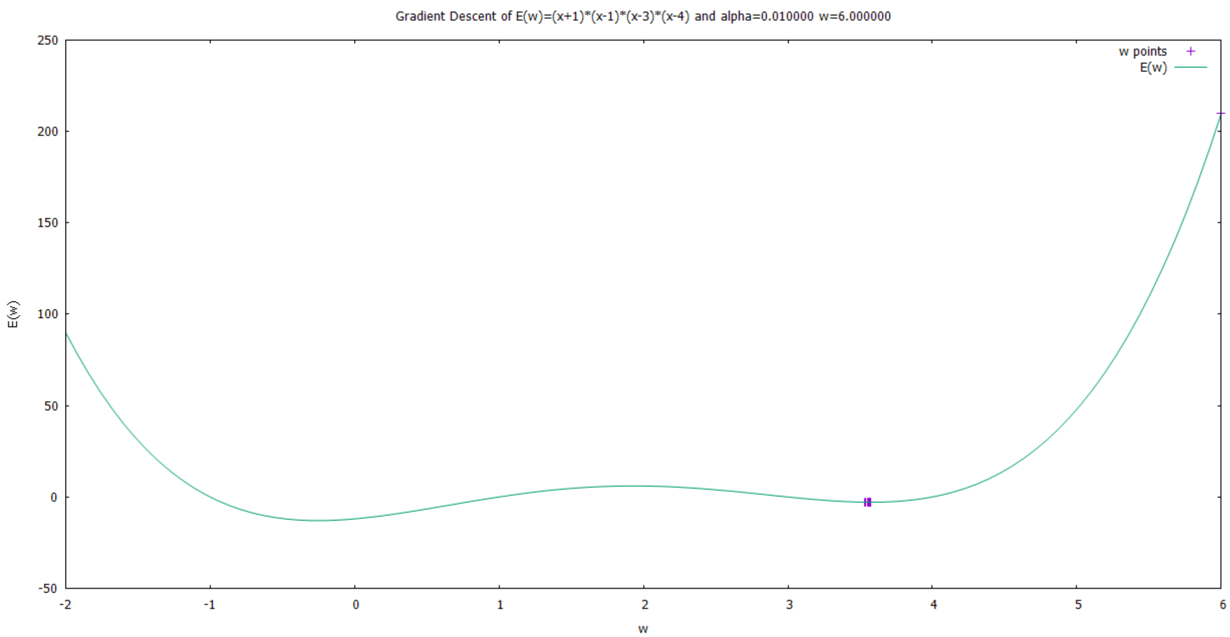
Graph 2: alpha = 0.002



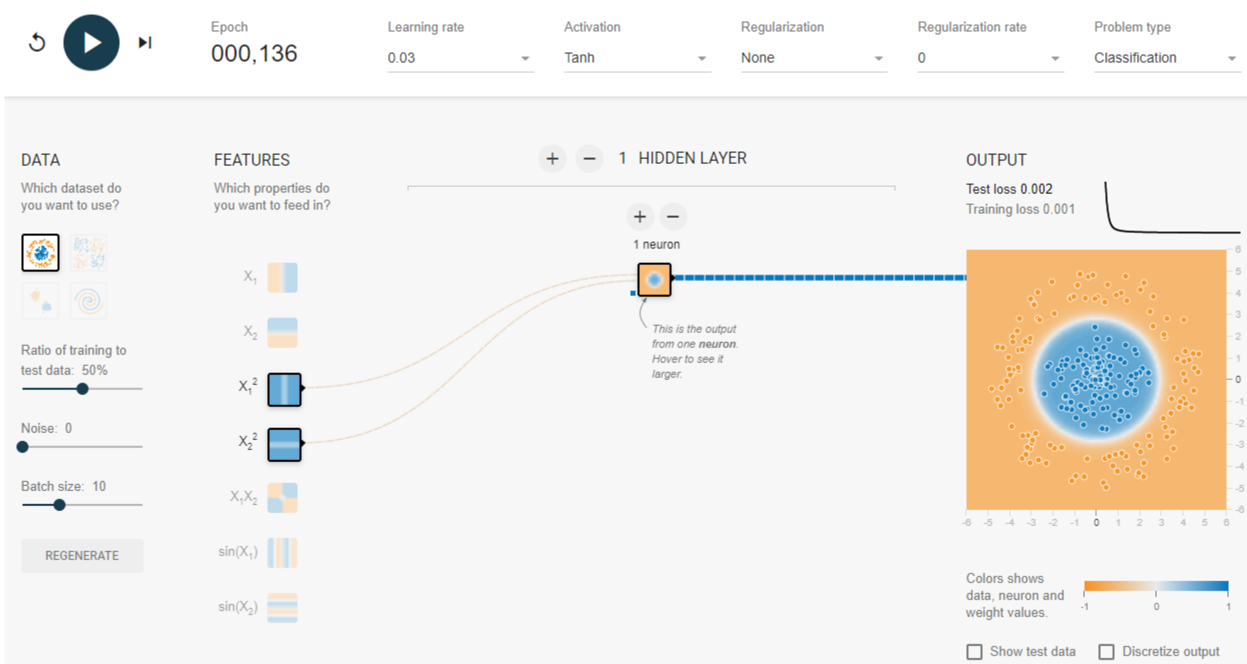
Graph 3: alpha = 0.005



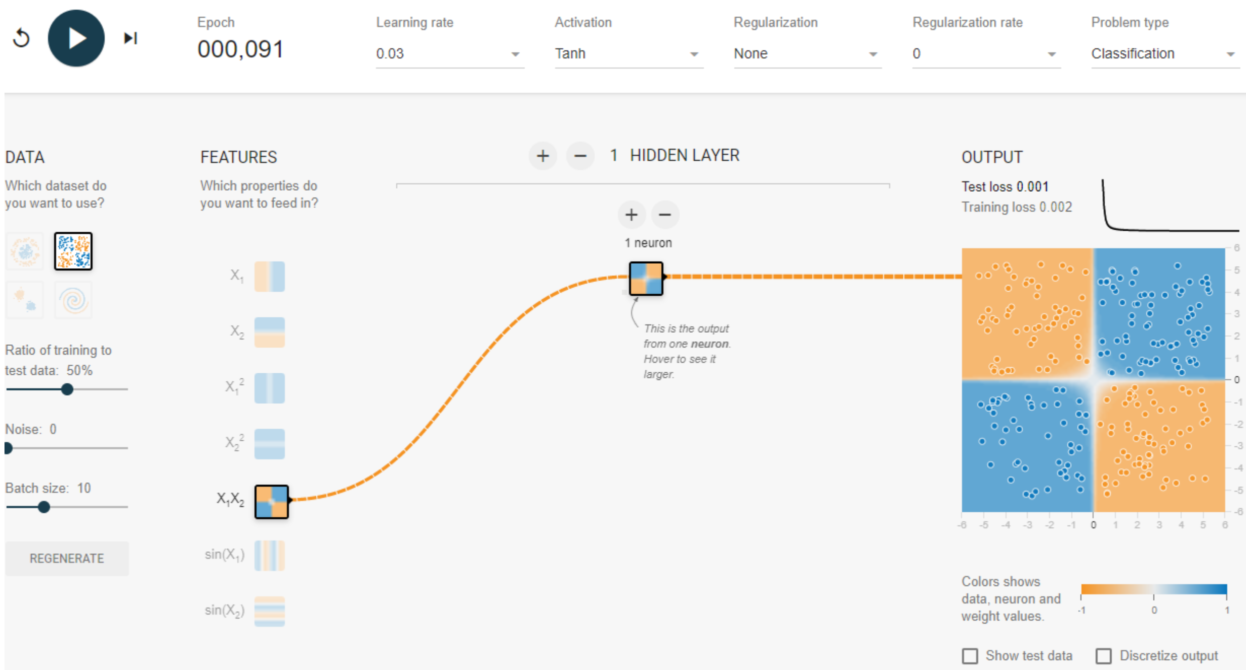
Graph 4: alpha = 0.01



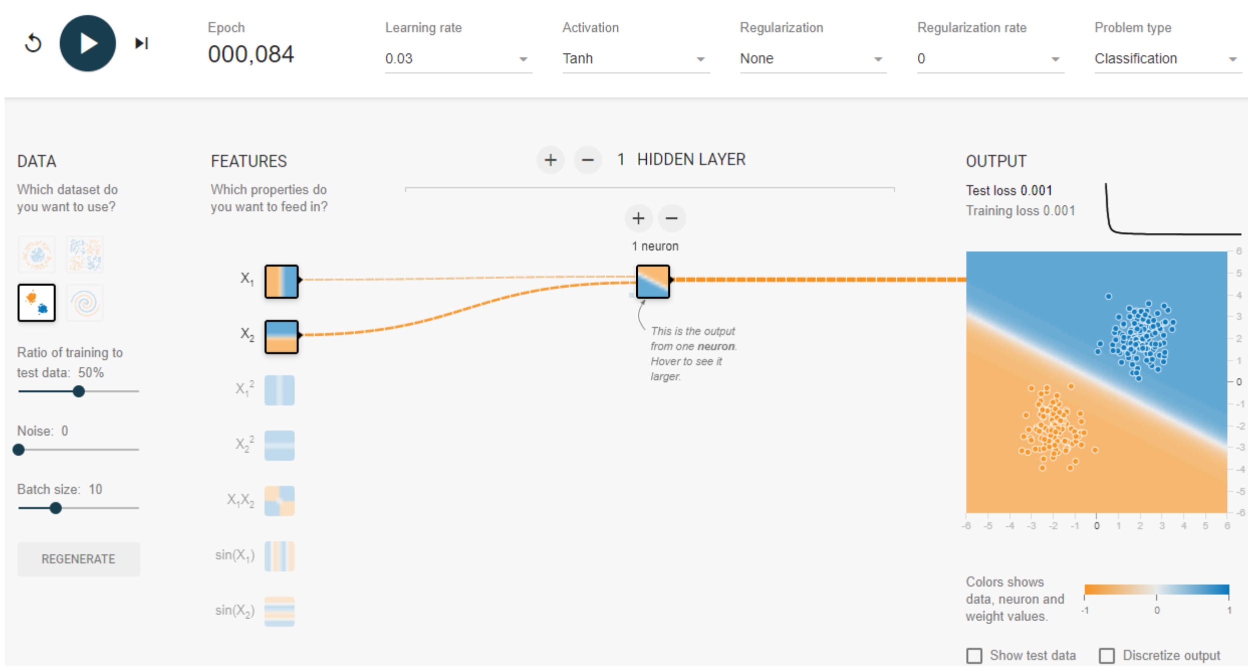
13. Dataset 1:



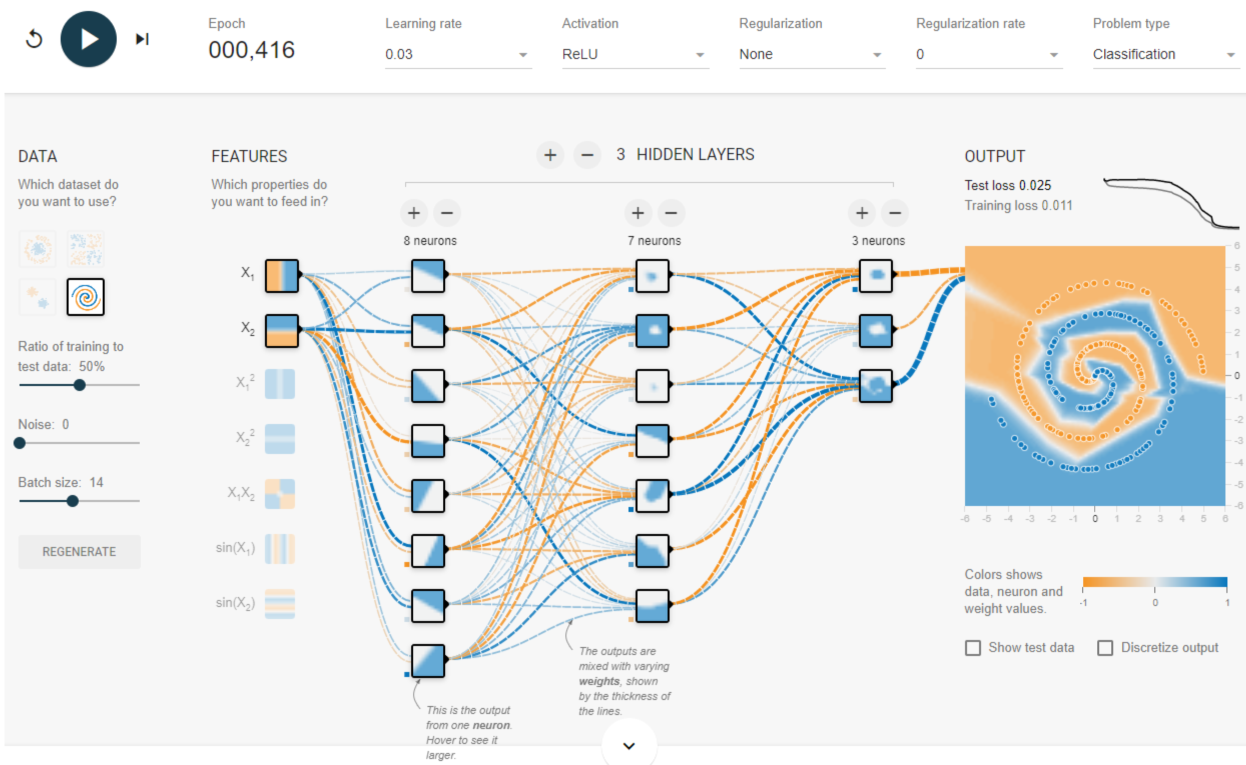
Dataset 2:



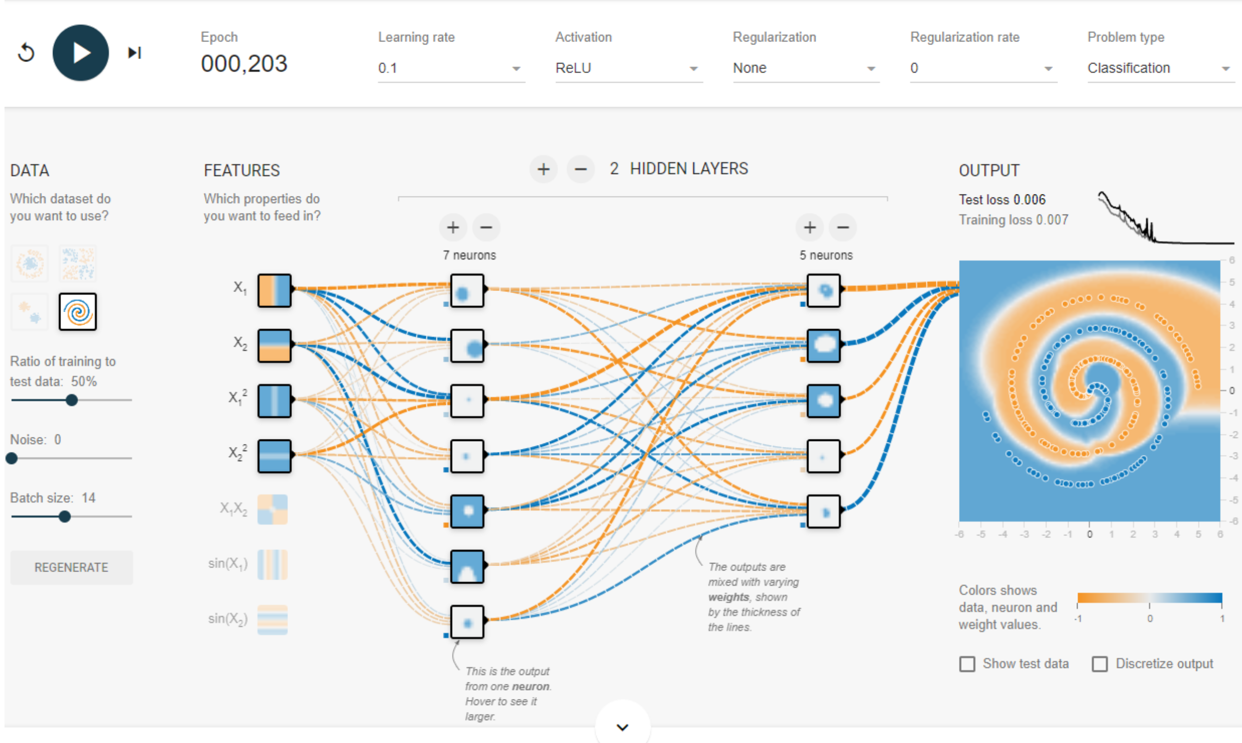
Dataset 3:



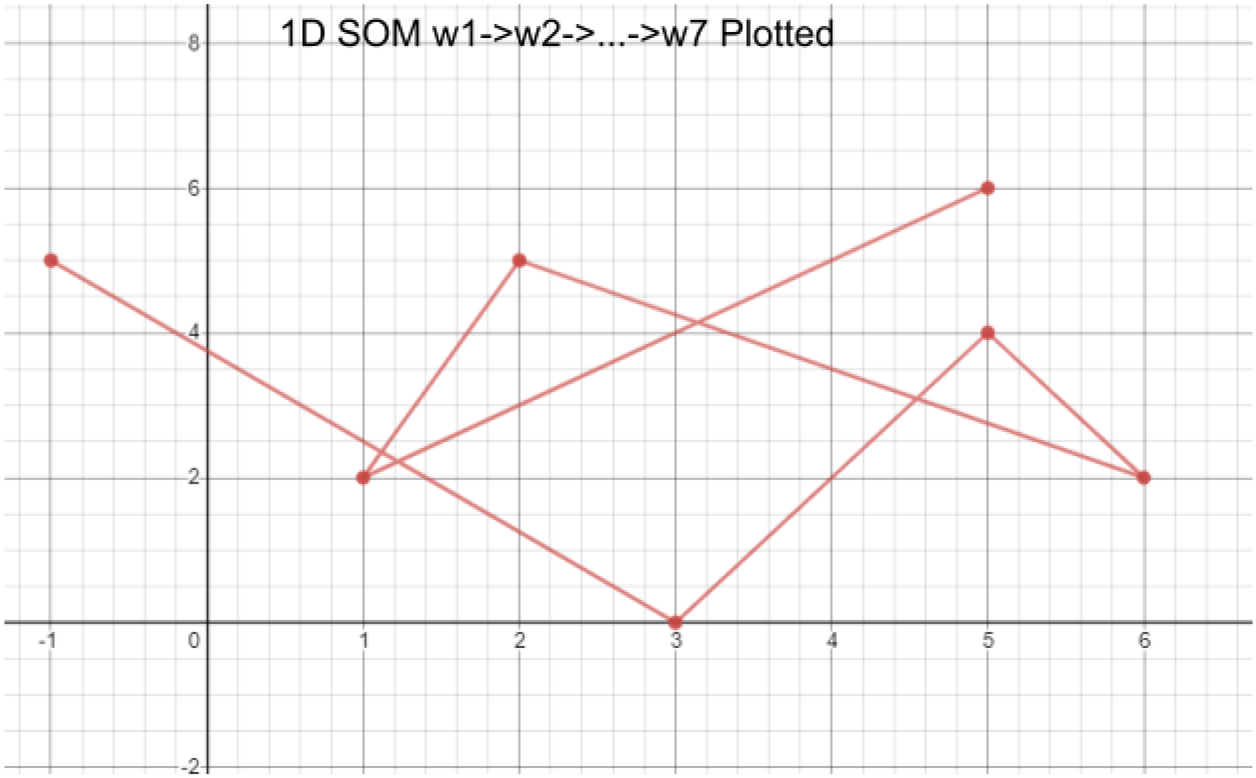
Dataset 4: Only x1, x2 used



Dataset 4: only x1,x2,x1^2,x2^2 used

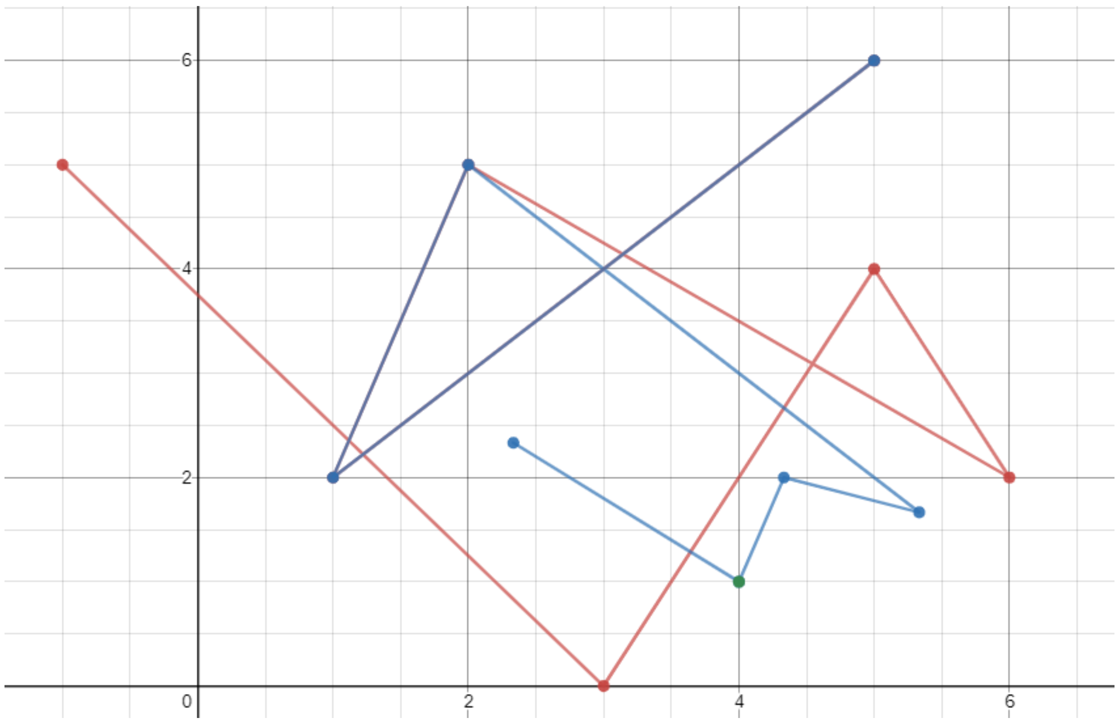


14. (3):



(4): input vector x = (4,1)
=> best matching is w6 (3,0)
    w6new = (3,0) + 1 * (1, 1) = (4, 1)
=> immidiate neighbors are w(6-1) and w(6+1) => w5, w7
    w5 = (5, 4) + ⅔ * (-1, -3) = (4.333, 2)
    w7 = (-1, 5) + ⅔ * (5, -4) = (2.333, 2.333)
=> second order neighbor is w(6-2) and w(6+2) => w4, w8
    w4 = (6, 2) + ⅓ * (-2, -1) = (5.333, 1.666)
    w8 doesn't exist
=> because rest h() value is 0, those weight vectors don't move!

The graph below shows the red line (before one iteration of training) and the blue line (after one iteration of training). The green dot is the input vector added.



6
4
2
0
2
4
6