NN coursework 1: Brian Formento: A0196190Y

Q1

1)

ξ is the boundary threshold where output $y = \varphi(v) > \xi$ belongs to C1 and if $\leq$ it belongs to C2.

Hence we can say that the decision boundary is defined as $\varphi(v) = \xi$

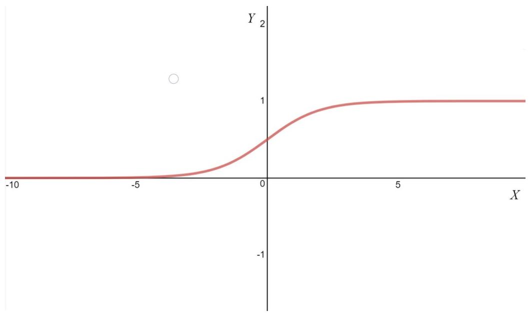
Therefore:

$$\xi = \frac{1}{1+e^{-v}} \; ; \; \xi + \xi e^{-v} = 1 \; ; \; e^{-v} = \frac{1-\xi}{\xi} \; ; \; -v = ln(\frac{1-\xi}{\xi}) \; ; \; v = ln(\frac{\xi}{1-\xi})$$

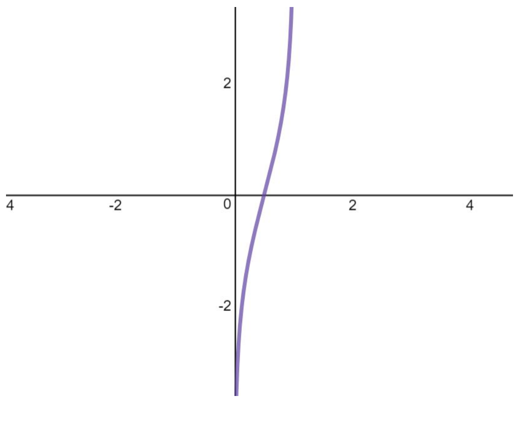
V is the receptive field of the perceptron, also defined as $v = \sum_{i=i}^{m} x_i w_i + b$ where b is the bias. What we are interested in is the term $\sum_{i=i}^{m} x_i w_i$ as this holds the equation for the perceptron boundary. If we rearrange for this term and substitute v we will get an equation, if such equation has only one constant solution for any value of ξ then the boundary is a hyperplane, otherwise it is not.

$\sum_{i=i}^{m} x_i w_i = ln(\frac{\xi}{1-\xi}) - b$ this clearly has a constant solution for $0 < \xi < 1$ and therefore:

$$\varphi(v) = \frac{1}{1+e^{-v}} \text{ is a hyperplane}$$



Plotting y = $ln(\frac{\xi}{1-\xi})$



2)
By using the same arguments

$$\varphi(v) = e^{\frac{(v-m)^2}{2}}$$

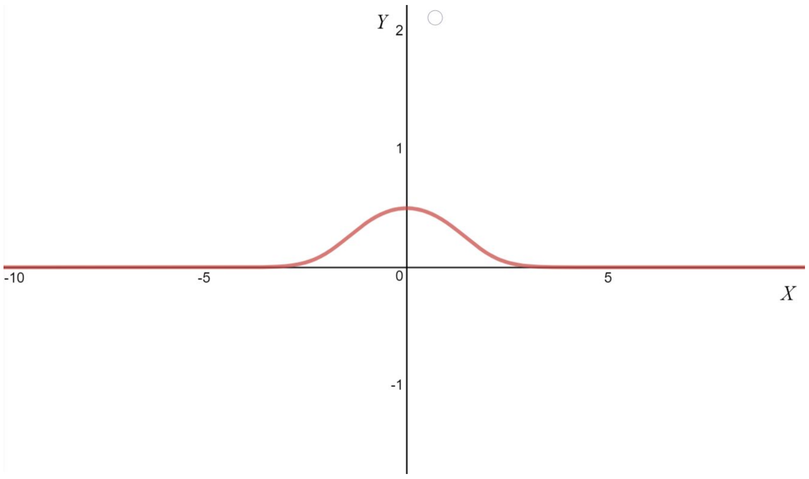
Hence

$$\xi = e^{\frac{(v-m)^2}{2}} \; ; \; ln(\xi) = \frac{(v-m)^2}{2} \; ; \; \pm\sqrt{2ln(\xi)} = v - m \; ; \; v = \pm\sqrt{2ln(\xi)} + m$$

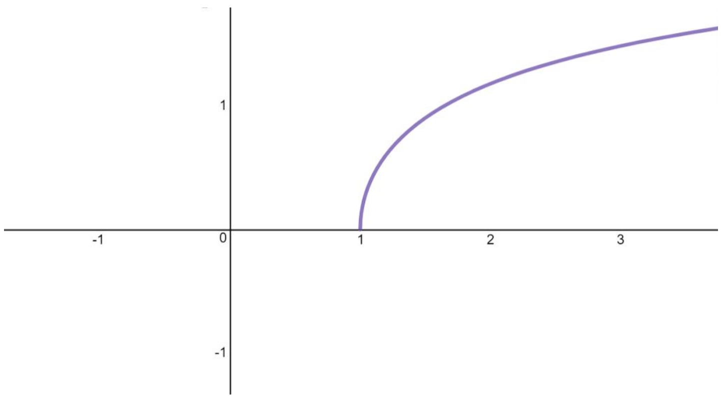
Where m defines how the gaussian is shifted along the x axis.

$\sum_{i=i}^{m} x_i w_i = \pm\sqrt{2ln(\xi)} + m - b$ is not a constant unless $\xi \leq 0$ or $\xi = 1$. Therefore:

$\varphi(v) = e^{\frac{(v-m)^2}{2}}$ is not a hyperplane.



Plotting $y = \sqrt{2ln(\xi)}$



3)

$$\varphi(v) = \frac{v}{1+|v|}$$

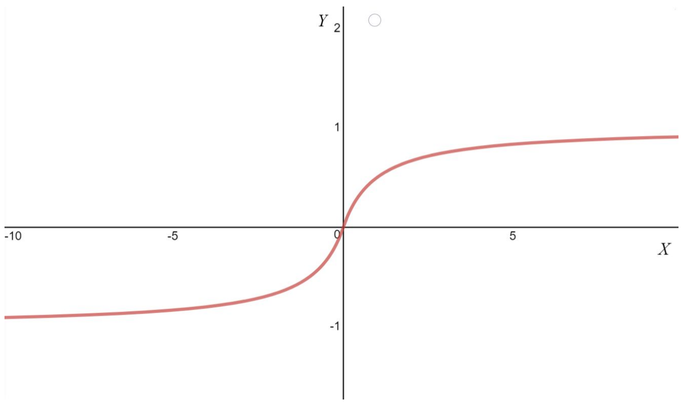
$$\xi = \frac{v}{1+|v|} \;;\; \xi(1+|v|) = v \;;\; \xi + \xi|v| = v \;;\; \xi = v - \xi|v|$$

At this point, since $|v|$ is the absolute value of v we have 2 potential paths, one where $v = -v$ and another when $v = +v$. hence with some arranging we have 2 equations:

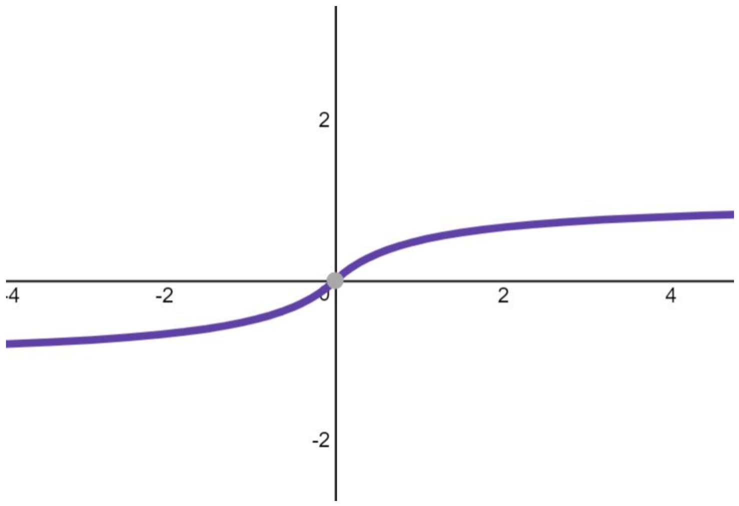
$$v = \frac{\xi}{1\pm\xi}$$

$\sum_{i=i}^{m} x_i w_i = \frac{\xi}{1\pm\xi} - b = \frac{\xi}{1+|\xi|} - b$ Where we have a constant for every value of ξ therefore:

$\varphi(v) = \frac{v}{1+|v|}$ is a hyperplane.

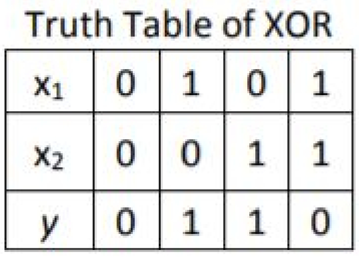


Plotting $y = \frac{\xi}{1+|\xi|}$



Q2

By looking at the truth table:



Truth Table of XOR

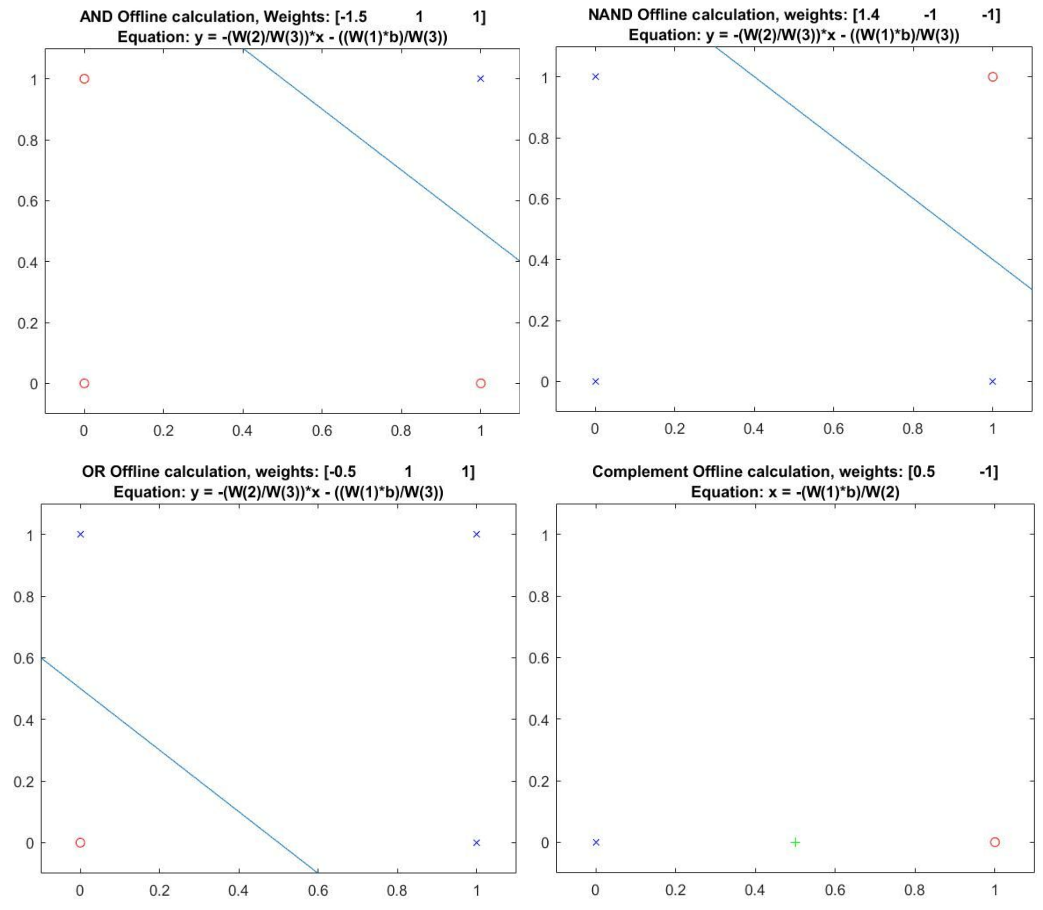
x_1	0	1	0	1
x_2	0	0	1	1
y	0	1	1	0

We can come up with 4 equations:

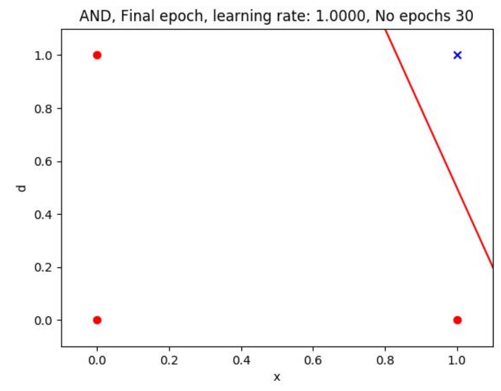
1) $0*x1\ +0*x2\ +\ b\ \le 0\ ;\ \ b\ \le 0 \qquad\qquad ;\ b\ \le 0$
2) $1*x1\ +0*x2\ +\ b\ > 1\ ;\ \ x1\ +\ b\ > 0 \qquad\qquad ;\ b\ >\ -x1$
3) $0*x1\ +1*x2\ +\ b\ > 1\ ;\ \ x2\ +\ b\ > 0 \qquad\qquad ;\ b\ >\ -x2$
4) $1*x1\ +1*x2\ +\ b\ \le 0\ ;\ \ x1\ +x2\ +\ b\ \le 0 \qquad\qquad ;\ b\ \le\ -x1\ -x2$

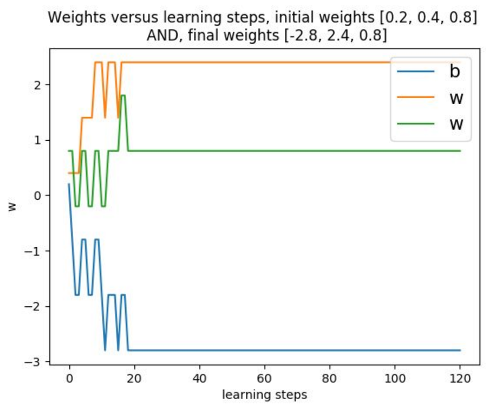
By looking at the 4 set of equations, more specifically 2 and 3 are in contradiction with each other and equation 4. These set of equations can not be solved!

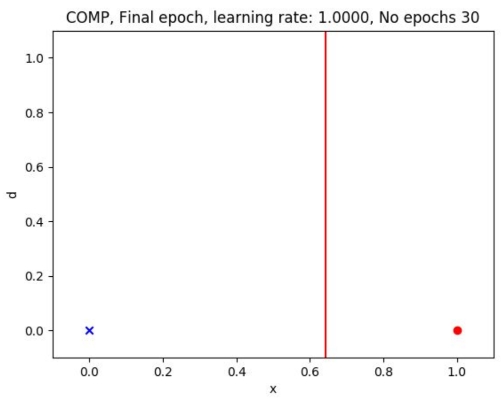
Q3 a) Done with matlab

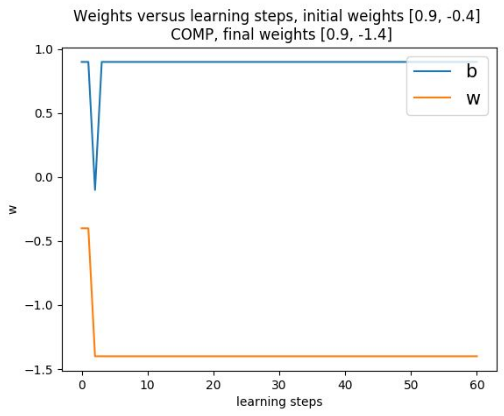


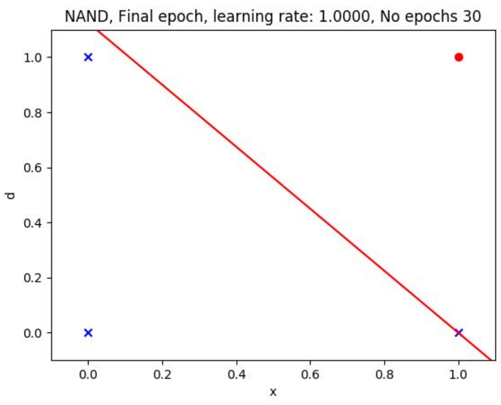
b) Changed to python

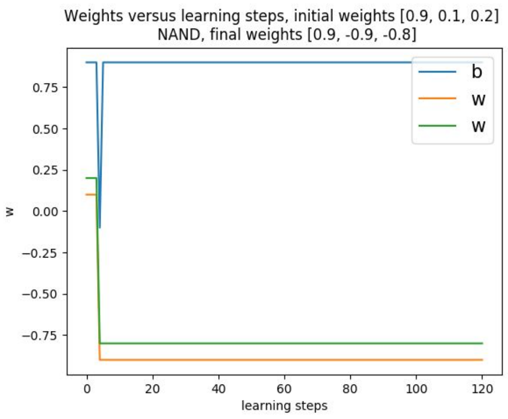


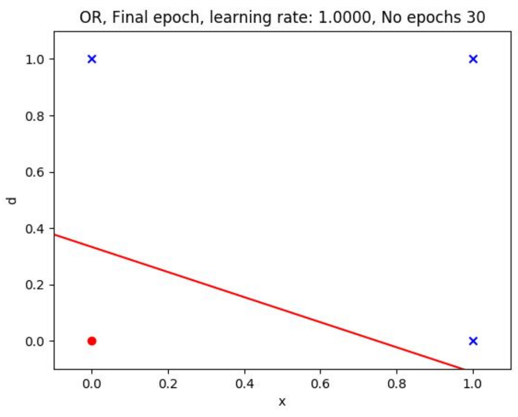


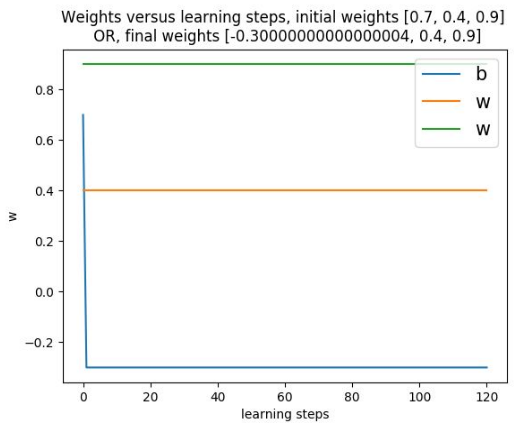




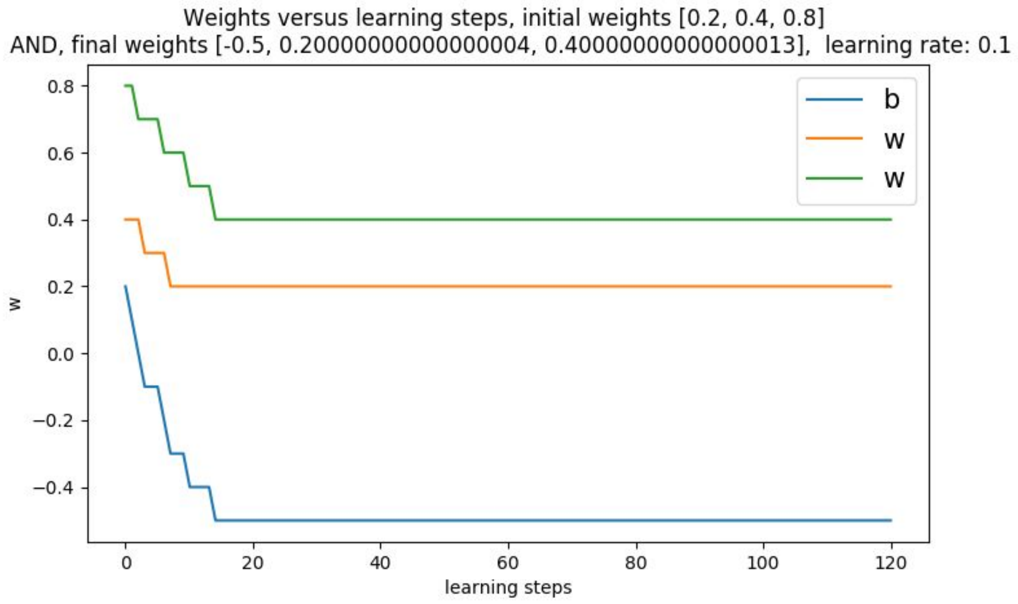


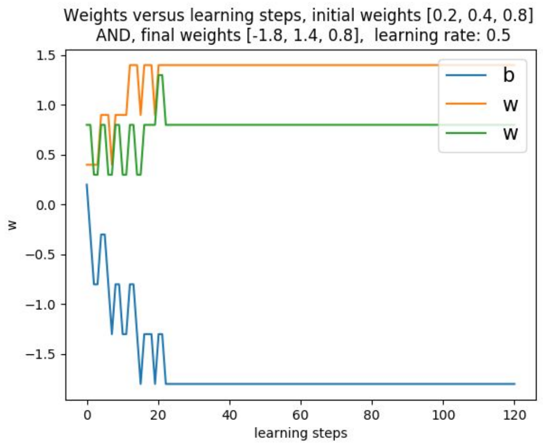


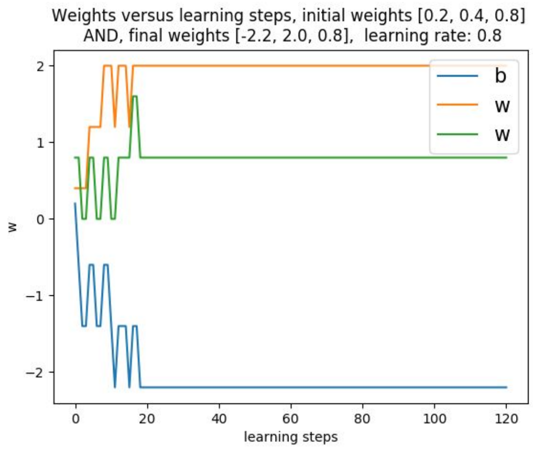




When the learning rate is lower it creates a smoother weights response, e.g the AND:

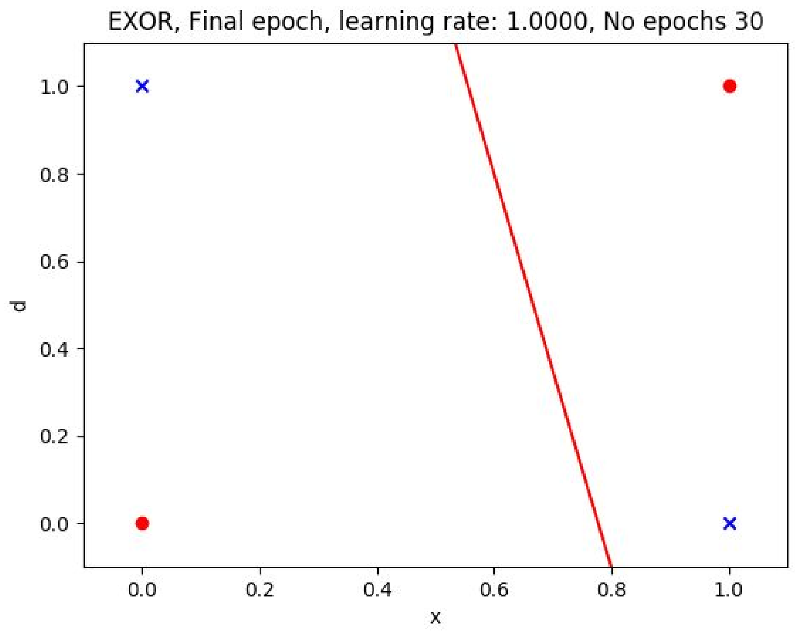


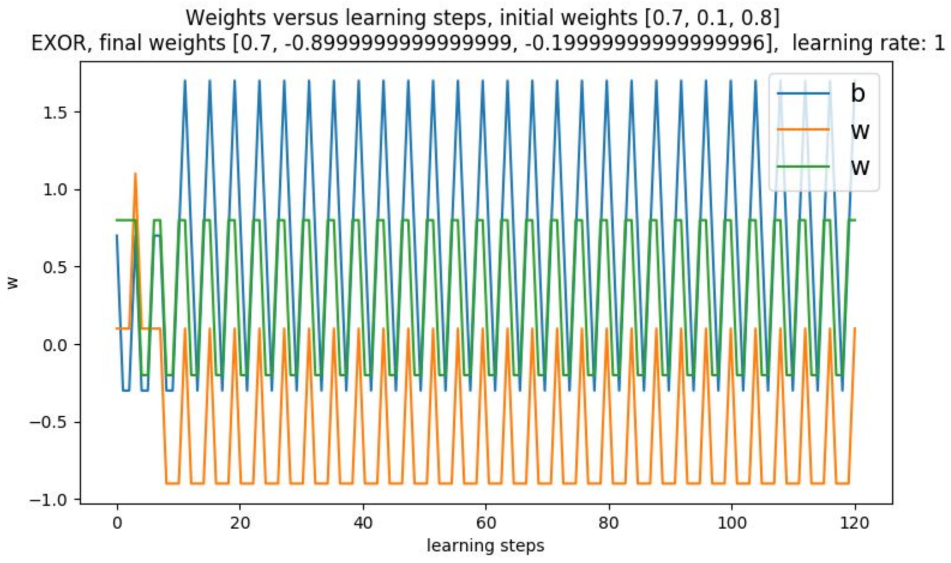




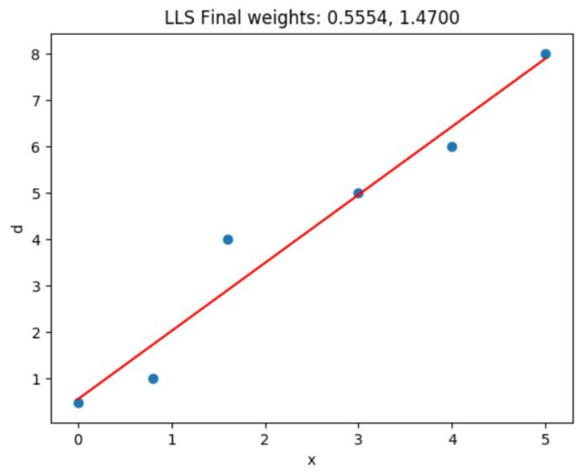
However it can happen that the boundary is set exactly on some of the training points, if this happens it could be considered a form of overfitting. It can also be seen how some learning rates allow some weights to reach the optimum faster. However this is highly dependant on the starting condition and learning rate. Therefore in a operational environment it’s not always easy to achieve such behaviour.

c)
With XOR no solution could be obtained, this is because as shown form the weights response there is an oscillatory behaviour where no optimum can be found. This is because, as explained in question 2 the XOR problem is non linearly separable, where two lines are needed to create a fit boundary.


EXOR, Final epoch, learning rate: 1.0000, No epochs 30
d
x

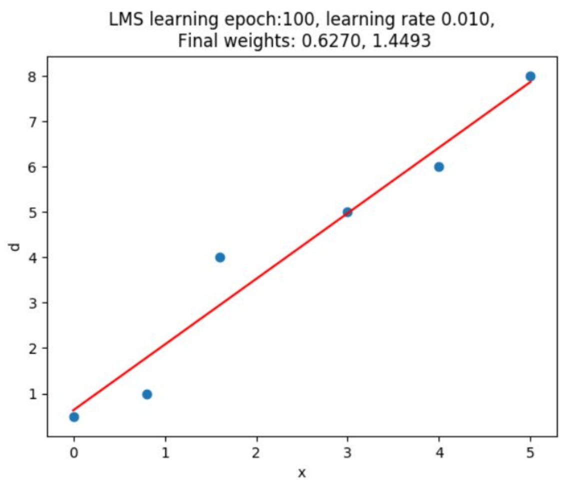

Weights versus learning steps, initial weights [0.7, 0.1, 0.8]
EXOR, final weights [0.7, -0.8999999999999999, -0.19999999999999996], learning rate: 1
b
w
w
w
learning steps

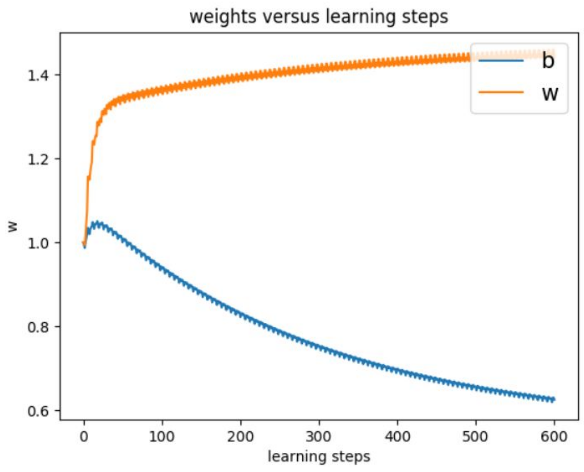
Q4 a) LLS


LLS Final weights: 0.5554, 1.4700
d
x

This is the mathematical optimal solution for linear regression.

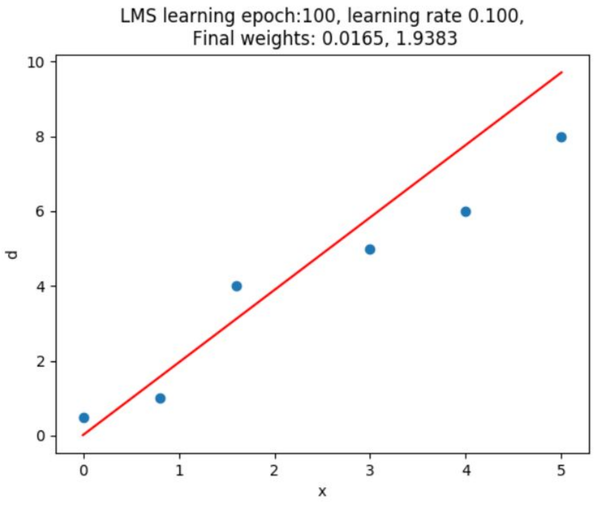
b) LMS

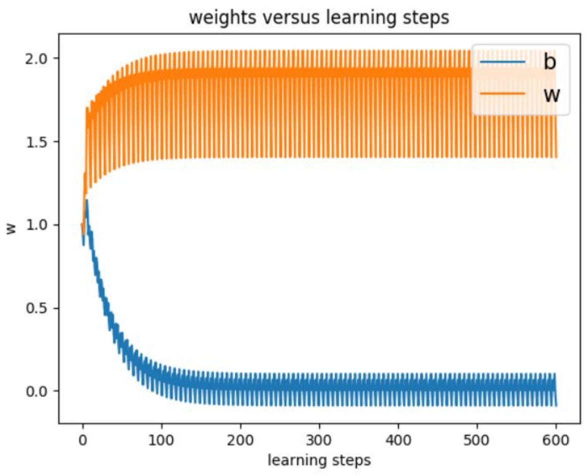




c) From the weights versus learning steps it looks like (given enough time) the weights will converge to an optimum. However, from looking only at the 100 epochs the found solution isn’t as perfect as the LLS method. The computation times also differ, LLS is computed almost instantly, while LMS takes some time. The problem with LLS is that with a large matrix X it will take a lot of memory space to do the whole calculations, hence making it memory inefficient compared to LMS.

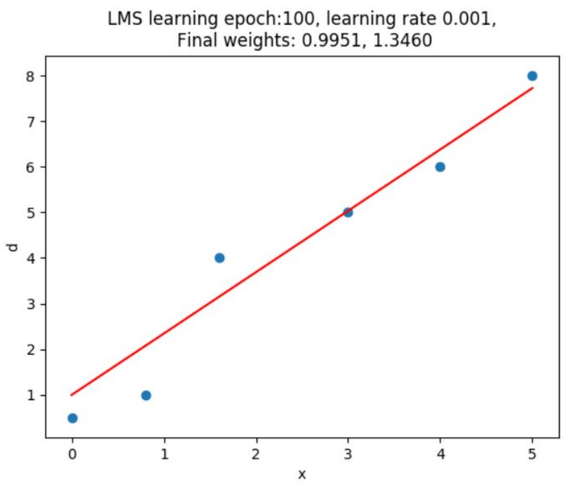
d) As expected with different learning rates we get different results.



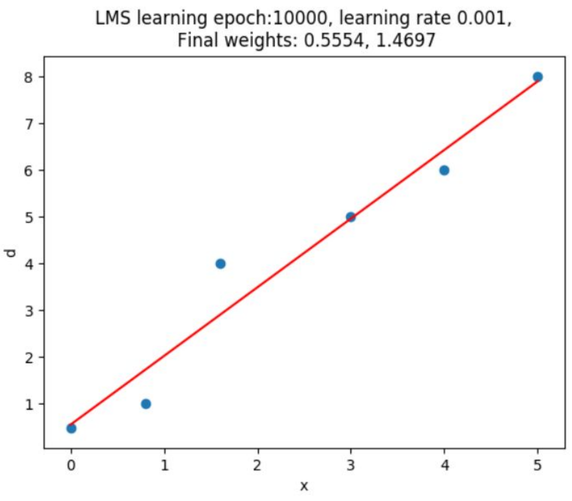


With a larger learning rate we have the worst fit, this is due to the optimizer not finding the absolute minimum in the valley. In fact, it overshoots (as shown from the noise in the weights versus learning steps graph) when the derivative of the energy function approaches zero.

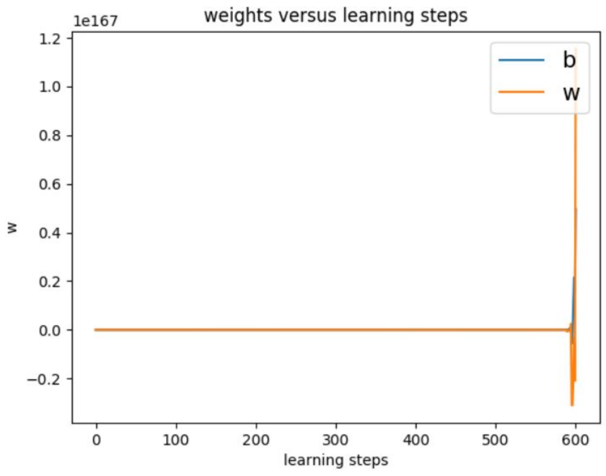
While when the learning rate is too low, if the epoch cut off is the same there is a chance that it does not converge in time. Hence, again it will give the worst result.

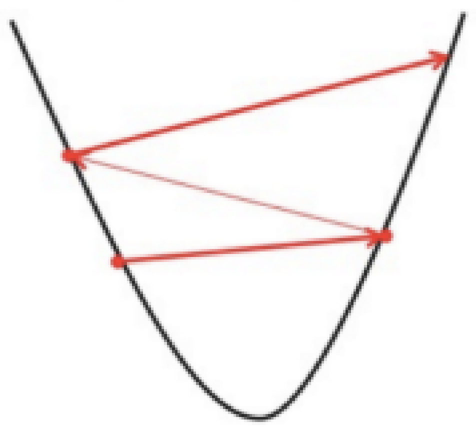


However, if given enough time, it will converge to even a better optimum as shown from the final weights being almost equal to the LLS method.



When the learning rate is too large e.g 0.5 the learning process will diverge. This is when the overshoot is so large that it escapes the minima because of a learning rate that is too large.





https://blog.algorithmia.com/introduction-to-optimizers/

Q5

The cost function is defined as $J(w) \ = \ \frac{1}{2}\sum_{i-1}^{n} r(i)^2 e(i)^2 \ + \frac{1}{2}\lambda w^T w$

The necessary condition for optimality is where $\Delta(J(w^*)) = 0$ however this is not trivial to get. However we can solve this iteratively, by starting with a random initial w(0) and updating it by knowing the gradient.

To minimise this we want to find the derivative J in respect to w, as by finding the gradient

$$\frac{dJ}{dw} \ = \ \frac{dJ}{de} * \frac{de}{dw} \ + \ \frac{d}{dw}(\lambda w^T(n)w(n))$$

Where $\frac{de}{dw} \ = \frac{d}{dw}(d(n) - x(n)^T w(n)) \ = \ -x^T(n)$

Then:

$$\frac{dJ}{de} = e^T(n)r^2(n)$$

Joining everything together after using the chain rule.

$$\frac{dJ}{dw} = \ -e^T(n)r^2(n)x^T(n) + \lambda w^T(n)$$

If we take an optimization procedure like steepest descent (gradient descend) where we have:

$w(n+1) = w(n) - \Delta w(n)$ and where $\Delta w(n) \ = \ \eta(\frac{dJ}{dw})^T$

Therefore:

$$w(n+1) = w(n) \ - \eta(\frac{dJ}{dw})^T$$

By substitution

$$w(n+1) = w(n) \ - \eta[- e(n)r^2(n)x(n) + \lambda w(n)\,]$$

The update rule can be described as:

$$w(n+1) = w(n) \ + \eta[e\,(n)r^2(n)x\,(n) - \lambda w(n)\,]$$

To fit a linear model we need the LMS algorithm, which is mathematically described as:

$$w(n+1) \ = \ w(n) \ + \eta e(n)x(n)$$

When $\lambda = 0$ we have an algorithm that can be used to approximate a linear model where $r^2(n)$ are the weighting factors for each output error e(n).

$$w(n+1) = w(n) \ + \eta e\,(n)r^2(n)x\,(n)$$