**Closed form solution math**

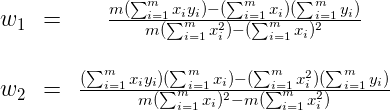
Closed form solution math can be used instead of gradient descent or the tricks to minimize the mean squared error.

It gives 2 equations and 2 unknowns for 2 variables. Thus, it will give n equations and n unknowns for n variables which will require high computational power and is not feasible. To avoid this we use gradient descent which doesn’t give exact solution but that fits pretty well.

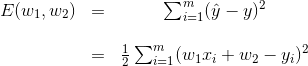
We'll develop the math of the closed form solution, which we introduced in the last video. First, we'll do it for the 2-dimensional case, and then for the general case.

**2-Dimensional solution**

Our data will be the values x1,x2,…,xm,*x*1​,*x*2​,…,*xm*​, and our labels will be the values y1,y2,…,yn.*y*1​,*y*2​,…,*yn*​. Let's call our weights w1,*w*1​, and w2.*w*2​. Therefore, our predictions are yi^=w1xi+w2.*yi*​^​=*w*1​*xi*​+*w*2​. The mean squared error is

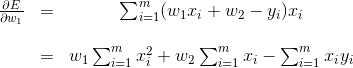
[[](https://classroom.udacity.com/me)](https://classroom.udacity.com/me)

We need to minimize this error function. Therefore, the factor of 1m*m*1​ can be ignored. Now, replacing the value of y^,*y*^​, we get

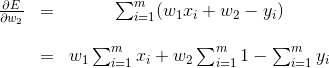
[[](https://classroom.udacity.com/me)](https://classroom.udacity.com/me)

Now, in order to minimize this error function, we need to take the derivatives with respect to w1*w*1​ and w2*w*2​ and set them equal to 0.0.

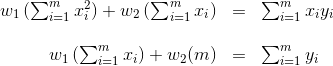
Using the chain rule, we get

[[](https://classroom.udacity.com/me)](https://classroom.udacity.com/me)

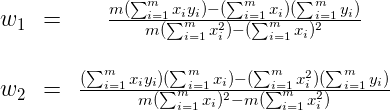
and

[[](https://classroom.udacity.com/me)](https://classroom.udacity.com/me)

Setting the two equations to zero gives us the following system of two equations and two variables (where the variables are w1*w*1​ and w2*w*2​).

[[](https://classroom.udacity.com/me)](https://classroom.udacity.com/me)

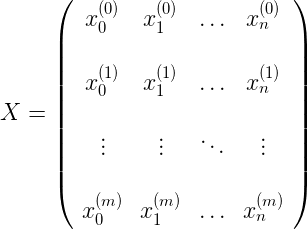
We can use any method to solve 2 equations and 2 variables. For example, if we multiply the first equation by ∑i=1mxi∑*i*=1*m*​*xi*​, the second one by m*m*, subtract them to obtain a value for w1*w*1​, and then replace this value in the first equation, we get the following:

[[](https://classroom.udacity.com/me)](https://classroom.udacity.com/me)

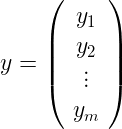
That's our desired solution.

**n-Dimensional solution**

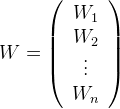
Now, let's do this when our data has n dimensions, instead of 2. In order to do this, we'll introduce the following notation. Our matrix X*X* containing the data is the following, where each row is one of our datapoints, and x0(i)=1*x*0(*i*)​=1 to represent the bias.

[[](https://classroom.udacity.com/me)](https://classroom.udacity.com/me)

Our labels are the vector

[[](https://classroom.udacity.com/me)](https://classroom.udacity.com/me)

and our weight matrix is the following:

[[](https://classroom.udacity.com/me)](https://classroom.udacity.com/me)

And so the equation for the mean square error can be written as the following matrix product:

E(W)=1m((XW)−yT)(XW−y)*E*(*W*)=*m*1​((*XW*)*T*−*yT*)(*XW*−*y*).

Again, since we need to minimize it, we can forget about the factor of 1m*m*1​, so expanding, we get

E(W)=WTXTXW−(XW)y−yT(XW)+yTy*E*(*W*)=*WTXTXW*−(*XW*)*Ty*−*yT*(*XW*)+*yTy*.

Notice that in the sum above, the second and the third terms are the same, since it's the inner product of two vectors, which means it's the sum of the products of its coordinates. Therefore,

E(W)=WTXTXW−2(XW)y+yTy*E*(*W*)=*WTXTXW*−2(*XW*)*Ty*+*yTy*.

Now, to minimize this, we need to take the derivative with respect to all values in the matrix W*W*. Using the chain rule, as we used above, we get the following:

[[/var/folders/mq/yt6n35gd1j1bvp286kf41cwc0000gn/T/com.microsoft.Word/WebArchiveCopyPasteTempFiles/e.gif](https://classroom.udacity.com/me)](https://classroom.udacity.com/me)

And in order to set this equal to zero, we need

XTXW−XTy=0*XTXW*−*XTy*=0, or equivalently,

W=(XTX)XTy*W*=(*XTX*)−1*XTy*.

That's it, that's our closed form solution for W*W*!

As we stated in the video, this method will be expensive in real life, since finding the inverse of the matrix XTX*XTX* is hard, if n*n* is large. That's why we go through the pain of doing gradient descent many times. But if our data is sparse, namely, if most of the entries of the matrix X*X* are zero, there are some very interesting algorithms which will find this inverse quickly, and that'll make this method useful in real life.