Simple Linear Regression

Regression Week 1 Notes v1

Simple Regression: Linear Regression with a Single Input

The value **x is the input**.
The value **y is the output**.
We assume they are related by some simple linear function, y = f(x).
The function is used to predict a value of y given a value for x.
Since it is a prediction, we say that y ~ mx + b and that there is some error.

So our regression model is:

$$y_i = f(x_i) + \epsilon_i$$

y_i is the **observed value**
$f(x_i)$ is the **predicted value**
ϵ_i is the **error** – the difference between the predicted and observed values.

The error is treated as a 'random quantity'. More precisely, the **Expected Value (E) of the error is zero**;

$$E[\epsilon_i] = 0$$

It is saying the weighted average of all (values that the error could take times the probability that it will take the value). Basically, it is saying there is no systematic error.

It also means that the error is just as likely to be positive, as it is to be negative. In other words, the observed value y_i is just as likely to be above predicted value $f(x_i)$ as it is to be below it.

Estimating the function from the data

First we must decide the kind of function; is it constant, like $y = 5$, it is linear like $y = mx + b$, is it quadratic like $y = ax^2 + bx + c$, or is it some higher order polynomial.

Once we have chosen the kind of function, then we estimate a specific fit for that function to the data. So if we have data and we decide we want to use a quadratic function, then we would estimate a specific quadratic function that best fits the data; this is the **estimated fit**, $\hat{f}(x)$ (f-hat of x).

y is the actual observed output value taken from the training data.
x is a feature of the data obtained by extracting it from the training data (and associated with the actual value y).
$\hat{y}$ is the predicted output value after applying our regression model to x.

Given y and $\hat{y}$, the **quality metric** is used to calculate the error between them.

Simple Linear Regression

Simple linear regression fits the data to a line with one input, a slope and an intercept:

$$f(x) = w_0 + w_1 x$$

w_0 is the intercept.
w_1 is the slope.

The regression model is:

$$y_i = w_0 + w_1 x_i + \epsilon_i$$

y_i	is the observed value
ϵ_i	is the difference between the observed value and the predicted value (the value o the line give by $w_0 + w_1 x_i$). So if the observed value y_i is above the line, the error is positive and negative if the observed value is below the line.
w_0 and w_1	are called the **regression coefficients** or the **regression parameters**.

Fitting the line to data: the Quality Metric

The estimated function is:

$$\hat{f}(x) = \hat{w}_0 + \hat{w}_1 x$$

In this setting $\hat{w}_0 + \hat{w}_1$ are the estimated regression coefficients, also just called $\hat{w}$ to generalize the notion. So in our discussion, we can talk about $\hat{w}$ rather than $\hat{f}$ because the coefficients fully describe the function.

We can calculate the 'cost' of using a given function (in this case a line) using **Residual Sum of Squares** (RSS). **A residual is the difference between a predicted value and an observed value**. The RSS for a set of regression coefficients is calculated by using the regression coefficients to calculate a predicted value, then finding the difference between the predicted value and the observed value and then squaring that difference. We do this for all training data and sum the squared differences. So this could be more descriptively called the Sum of the Squared Residuals (but don't do that). So this can be written:

$$RSS(w_0, w_1) = \sum_{i=1}^{n} (y_i - [w_0 + w_1 x_i])^2$$

Now given this function, **we can calculate the cost of any estimated line $\widehat{w}$** (the line with estimated coefficients $\widehat{w}_0$ and $\widehat{w}_1$). The idea then, is to find the line with the lowest cost and use that as our model for predicting values.

Model vs Fitted Line

Our regression model is given by

$$y_i = w_0 + w_1 x_i + \epsilon_i$$

In this context we will call w_o and w_1 the model parameters. We are trying to calculate these - they are unknowns. In trying to calculate the model parameters, we generated estimated parameters, $\widehat{w}_0$ and $\widehat{w}_1$, which are known values (they are known because we generated them). We use these estimated parameters to produce a specific fitted line:

$$\hat{f}(x) = \widehat{w}_0 + \widehat{w}_1 x$$

This fitted line produces a specific predicted value $\hat{y}$ given an specific observed value x;

$$\hat{y} = \hat{f}(x)$$

Note that we can use this fitted line to make reverse predictions as well. If we have a specific y, we can calculate an estimated $\hat{x}$.

Interpreting the Coefficients

The slope $\widehat{w}_1$ of the fitted line has useful meaning on its own. It represents the predicted change in the output per unit change in the input.

The magnitude of the change depends on the units of the coefficients. Of course, coefficients of the fitted line are in the same units as the training data and so our inputs to the fitted line must be in these units. The y-axis and the y-intercept are in the units of our observed values. The slope is in units of observed values / units of feature values (rise over run). So if we are predicting the prices of houses in dollars using the houses' floor space in square feet, the then y-intercept in in dollars and the slope is in dollars/square feet.

Defining the Least Squares Optimization Objective

We wish to minimize the Residual Sum of Square RSS($w_{0,}w_1$) for all possible coefficients. This is goal is given by;

$$\min_{w_0,w_1} \sum_{i=1}^{n} (y_i - [w_0 + w_1 x_i])^2$$

We can see the sum represents RSS(w_0, w_1). The notation indicates that we want to minimize the results of a function (in this case a function over two variables, w_0, w_1.

Concave and Convex functions

Concave functions are those that, for any two points on the curve, the points of the line connecting those two points will always be below (less in the y direction) than the corresponding points of the curve.

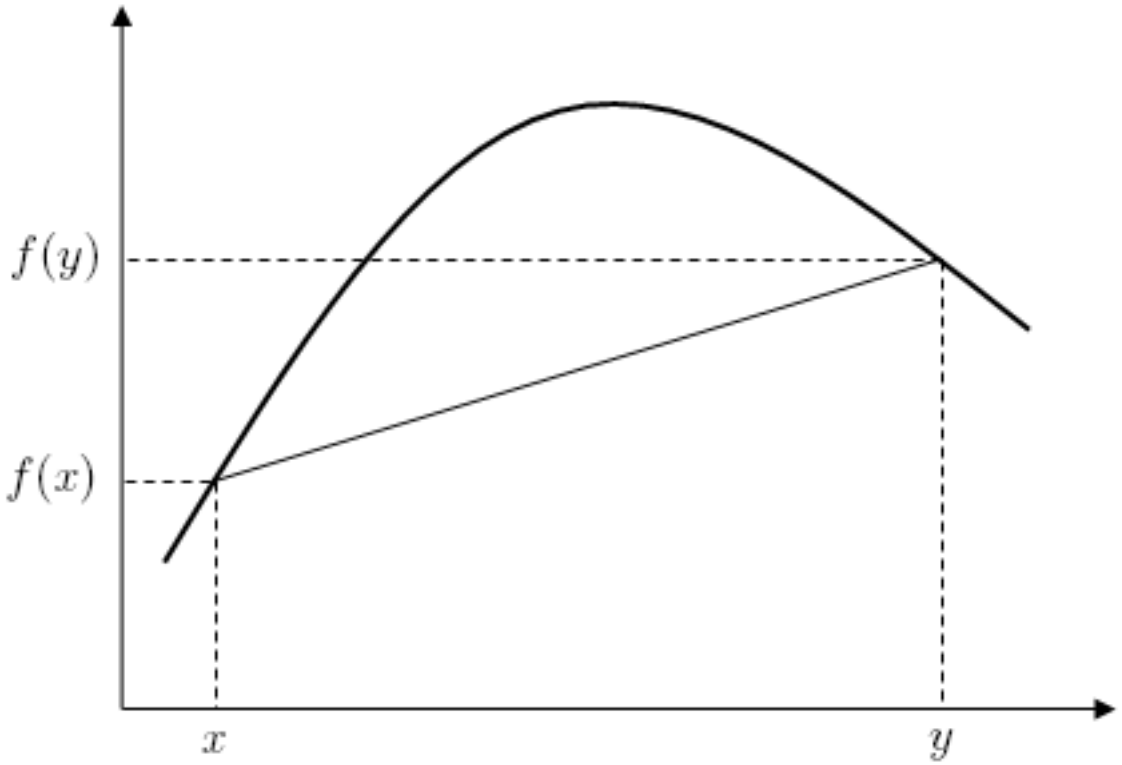
Convex functions are those that, for any two points on the curve, the points of the line connecting those two points will always be above (greater in the y direction) than the corresponding points of the curve.

In these two cases, we are talking about a **secant line**. From Wikipedia;

*In geometry, a **secant line** of a curve is a **line** that (locally) intersects two points on the curve. A chord is an interval of a **secant line**, the portion of the **line** that lies within the curve.*

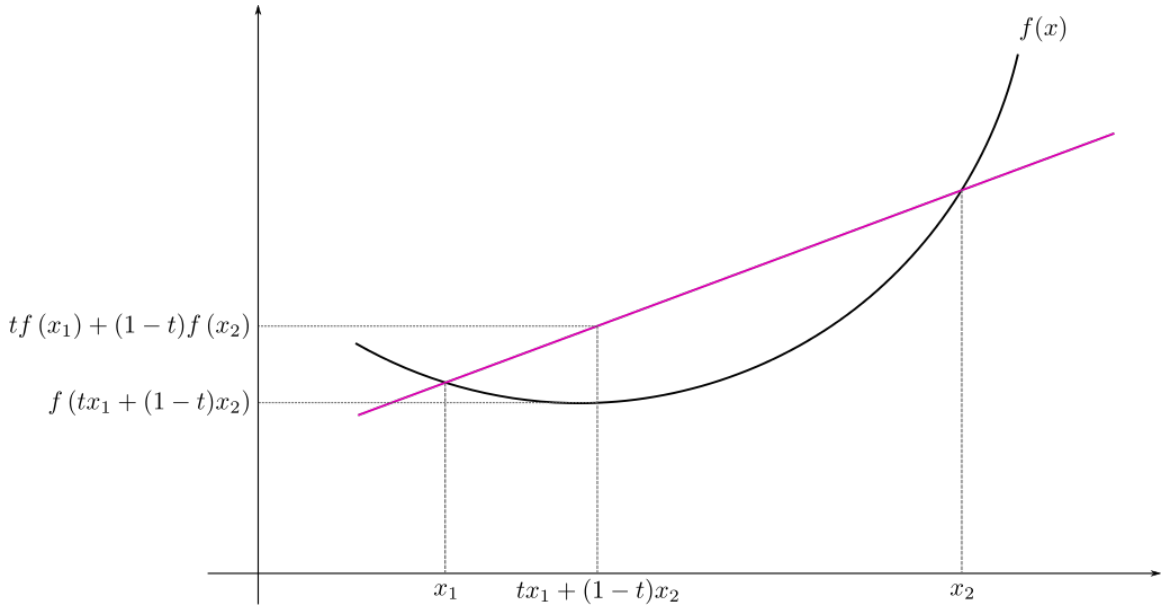
So we are saying, for a concave curve, the chord is always below the curve and for a convex curve, the chord is always above the curve.

Here is an example of a concave curve from Wikipedia;



This shows one possible secant line on the concave curve.

Here is an example of a convex curve from Wikipedia;



The pink line is one possible secant line for this convex curve.

Some functions are neither concave nor convex. The line connecting two points on the curve may actually cross the function (so it intersects at more than two places and so is not a chord), indicating that section of the curve is both concave and convex. Or it may be a line, which has no curvature.

Curvature & Instantaneous Rate of Change

The **first derivative** is the **instantaneous rate of change** at a point on the curve. It is the **slope of the** line that is **tangent** to the curve at that point. From Wikipedia;

*In geometry, the **tangent line** (or simply tangent) to a plane curve at a given point is the straight line that "just touches" the curve at that point. Leibniz defined it as the line through a pair of infinitely close points on the curve.*

NOTE: We use two points on the curve to create a secant line. If we begin choosing points that are closer together, as the points get closer and closer together, the distance between them approaches zero (they become infinitely close), and we approach the single point which is the tangent line. That is the geometric interpretation of the first derivative for the curve at a given point.

In the case of convex and concave functions, at the place where the curve changes slope, there is point where the instantaneous rate of change of the slope is zero. **That is, there is a place where the tangent line is exactly horizontal. That place, where the first derivative is zero, is the maximum in the case of concave curves or the minimum in the case of convex functions**.

Also worth noting, any place on a curve where the slope changes 'sign', either going from positive slope to negative slope or visa versa, is called an inflection point. So the maximum of a concave curve is an inflection point. To the 'left', where x values are less, it has positive slope. On the 'right' side of the maximum, where x values are greater, it has negative slope. At the maximum, the inflection point has zero slope.

NOTE: here is a cool animation from Wikipedia that shows how the tangent changes as we move along a 'curvy' curve that has many local maxima and minima. You can see that the top of local maximum the slope of the tangent is zero (the tangent is exactly horizontal) and the same at the bottom of local minima. This also hints at potential problems with the hill-climbing algorithm – with non-concave functions, we may only find a local maximum.

Hill Climbing – finding the maximum of a concave function

Let's assume our function is a concave function. The function is given by g(w) and the maximum is given by max w g(w). Then at max w g(w), the first derivative is zero; that is the point we are looking for.

To find the maximum of a concave curve, we can use a hill-climbing algorithm. For the concave curve given by the function g(w):

1. Choose an initial value for w. We can think of this as an initial 'point' on the curve *[w, g(w)].*
2. Calculate the slope of the curve at that point.
3. If the slope is positive, choose a larger value for *w* and repeat.
4. If the slope is negative, choose a smaller value for *w* and repeat.
5. If the slope is zero (to within our chosen tolerance), we are done.

There are several aspects to pay attention to;

- We have to choose an initial value of *w* to start. So we have to decide how to do that.
- We have to calculate the slope at point *w*. If we know the function *g(w)*, then this can be directly calculated using a first derivative. Of course, we are actually trying to figure out our function, so we will have to use some other way to calculate the slope. HINT: that is where the secant line comes in.
- After we figure out the slope, we want to increase or decrease *w* and try again, so we have to figure out how much to increase or decrease *w*.

This can be summarized for iteration **t**;

$$w^{(t+1)} \leftarrow w^{(t)} + \eta \frac{dg(w)}{dg}$$

$w^{(t)}$ is the value of *w* that we chose for iteration t of the algorithm.

$w^{(t+1)}$ is the value of w in the next iteration (the value we are calculating)

η (the greek letter Eta) is the step size. This is some value that we choose.

$\frac{dg(w)}{dg}$ is the first derivative of *g(w)* at our chosen $w^{(t)}$. This is the slope at the point *[w, g(w)]* on the curve.

$\eta \frac{dg(w)}{dg}$ The step size times the slope at *w* determines how much we change *w* for the next iteration. So we can see the actual amount we move AND the direction we move will depend on the slope at g(w). In places the slope has a large magnitude, the curve is steep, we will move far. As the slope gets close to zero, we will not move far. This is good, because it means as we get close, the algorithm automatically get's more 'sensitive' in searching for zero slope and when we are not close, it moves quickly. Nice. Also, if the slope is positive, we will move to the 'right' as we wanted and if the slope is negative, we will move to the left as desired. Nice. Also, if we jump over the actual maximum, we will see a change in slope. This is nice. Also, because the slope has changed, we will 'reverse direction' and head back towards the maximum, so the algorithm seems to naturally converge. Very nice. Finally, if the derivative becomes zero, we will not move at all and we know we are done.

Hill Descent - Finding the minimum of a convex function

In this case, we want choose the value for w in the next iteration, $w^{(t+1)}$, we want to head in the opposite direction from what we did for a concave function. When the slope is positive, we want to move 'left" or we want to decrease w. When the slope is negative, we want to increase w. So instead of adding $\eta \frac{dg(w)}{dg}$ to $w^{(t)}$, we want to subtract it. So Hill Descent looks like this;

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{dg(w^{(t)})}{dg}$$

Choosing the step size

In the simplest case, we can choose a constant value for the step size. For strongly convex or concave curves, this will converge naturally.

We can also vary the step size dynamically. For instance, as the algorithm runs, we expect to get closer to the answer. So, if we take this into account, we can start with a larger step size and then decrease the step size with each iteration. We have to be careful to not decrease the step size too rapidly, because that may cause us to converge too slowly (offsetting any gain we got by starting with a large step size).

Choosing when to stop

We have said that when the slope becomes zero, we are done. However, in practical terms, because floating point hardware in computers is not infinitely accurate, we may not reach zero. Also, we may be happy with getting close enough, so we can stop early and make our algorithm faster. So in practice, we choose a tolerance, ϵ. If the absolute value of the slope is within the tolerance, we stop.

$$\left|\frac{dg(w)}{dg}\right| < \epsilon$$

We choose ϵ carefully, based on our data.

Moving to multiple dimensions – Gradient Descent

Our prior discussion is of a function in one variable *g(w)* where the derivative is with respect to the one variable; $\frac{dg(w)}{dg}$. In fact, even in our simple linear equation, we have two variables, w_0, w_1. So we need to generalize the notion of derivative to the notion of gradient to support more than one variable.

$$\nabla g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_0} \\ \cdots \\ \frac{\partial g}{\partial w_p} \end{bmatrix}$$

∇ the Greek letter nabla is the symbol for gradient.

$g(w)$ is a function of the vector *w*. (don't let the g confuse with you, it could be any letter – it is not specific to the gradient).

w is a vector of length p + 1. Each element is one coefficient, named from w_0 to w_p.

[] the result is a vector of length p + 1, where each element is the partial derivative in one element on w, from w_0 to w_p, respectively. In a partial derivative, say for w_0, we take the derivative with respect to w_0 and treat all other elements of w as constants.

So, in effect, **the gradient is a set of derivatives, each with respect to one coefficient in the function**. The resulting gradient has the same dimension as the original function.

So our hill descent algorithm becomes a **gradient descent algorithm** by using the gradient rather than the derivative;

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \nabla g(w^{(t)})$$

Written out more fully;

$$\begin{bmatrix} w_0^{(t+1)} \\ \cdots \\ w_p^{(t+1)} \end{bmatrix} \leftarrow \begin{bmatrix} w_0^{(t)} \\ \cdots \\ w_p^{(t)} \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial g}{\partial w_0^{(t)}} \\ \cdots \\ \frac{\partial g}{\partial w_p^{(t)}} \end{bmatrix}$$

Convergence in this case is when the magnitude of the gradient goes to zero or in practical terms, when the magnitude of the gradient is within some tolerance ϵ to zero.

$$\|\nabla g(w)\| < \epsilon$$

The magnitude of a vector is the square root of the sum of the squares of its elements. So for the vector *w* with *p + 1* elements the magnitude is given by;

$$\|w\| \leftarrow \sqrt{{w_o}^2 + {w_1}^2 + \cdots {w_p}^2}$$

Computing the Gradient of RSS

It can be shown the RSS function is convex, so that there is a single unique solution to the minimization problem AND the gradient descent algorithm will converge to the minimum.

$$\min_{w_0,w_1} \sum_{i=1}^{n} (y_i - [w_0 + w_1 x_i])^2$$

Remember our RSS function is the sum of the squares of differences between our predicted and observed values;

$$RSS(w_0, w_1) = \sum_{i=1}^{n} (y_i - [w_0 + w_1 x_i])^2$$

ASIDE: When doing this, it is helpful to remember that the derivative of a sum is the same as the sum of the derivatives of each of the components of the sum;

$$\frac{d}{dw}\sum_{i=1}^{n} g_i(w) = \frac{d}{dw}\big(g_1(w) + g_2(w) + \cdots g_n(w)\big)$$

$$= \frac{d}{dw} g_1(w) + \frac{d}{dw} g_2(w) + \cdots \frac{d}{dw} g_n(w) = \sum_{i=1}^{n} \frac{d}{dw} g_i(w)$$

or simply;

$$\frac{d}{dw}\sum_{i=1}^{n} g_i(w) = \sum_{i=1}^{n} \frac{d}{dw} g_i(w)$$

In our case of the RSS function, $g_i(w)$ *is*;

$$g_i(w) = (y_i - [w_0 + w_1 x_i])^2$$

So to calculate the gradient, we take the partial derivative of each element of our coefficient vector $\begin{bmatrix} w_o \\ w_1 \end{bmatrix}$. For the first element w_o;

$$\frac{\partial RSS(w)}{\partial w_0} = \sum_{i=1}^{n} \frac{\partial}{\partial w_o} (y_i - [w_0 + w_1 x_i])^2$$

and similarly the partial derivative of RSS(w) with respect to w_1 is given by;

$$\frac{\partial RSS(w)}{\partial w_1} = \sum_{i=1}^{n} \frac{\partial}{\partial w_1} (y_i - [w_0 + w_1 x_i])^2$$

If we do the work to complete these partial derivatives, then we find that the gradient for our RSS function is given by a two-element vector;

$$\nabla RSS(w_0, w_1) = \begin{bmatrix} -2 \sum_{i=1}^{n} y_i - (w_0 + w_1 x_i) \\ -2 \sum_{i=1}^{n} y_i - (w_0 + w_1 x_i)\, x_i \end{bmatrix}$$

Approach 1 – Closed-form Solution

Closed-form solution to gradient – **set gradient = 0 and solve for w_o and w_1.**

Simple Linear Regression Closed Form Solution

$$\widehat{w}_0 = \frac{\sum_{i=1}^{n} y_i}{n} - \widehat{w}_1 \frac{\sum_{i-1}^{n} x_i}{n}$$

$$\widehat{w}_1 = \frac{\sum_{i=1}^{n} y_i\, x_i - \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i}{n}}{\sum_{i=1}^{n} {x_i}^2 - \frac{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i}{n}}$$

There are some things to note;

- $\widehat{w}_0$ is in terms of $\widehat{w}_1$ so we want to calculate $\widehat{w}_1$ first.
- There are only really 4 summations we need to calculate, which we can them combine to get our values;
  - $\sum_{i=1}^{n} y_i$ is the sum of y values
  - $\sum_{i-1}^{n} x_i$ is the sum of x values
  - $\sum_{i=1}^{n} y_i\, x_i$ this is the sum of x*y
  - $\sum_{i=1}^{n} {x_i}^2$ this is the sum of x squared
- $\frac{\sum_{i=1}^{n} y_i}{n}$ is the mean of our observations. In our house example, the average house price of the houses in the training set.

- $\widehat{w}_1 \frac{\sum_{i-1}^{n} x_i}{n}$ is the slope times the mean of our features. In our house example, the mean of the features is the average square footage of the houses in the training set.

So the slope is given by numerator/denominator;
numerator = (sum of X*Y) - (1/N)*((sum of X) * (sum of Y))
denominator = (sum of X^2) - (1/N)*((sum of X) * (sum of X))

We can divide both the numerator and denominator by N (essentially multiplying the ratio by 1, which does not change anything) to reformulate in terms of means (sums/N);
numerator = (mean of X * Y) - (mean of X)*(mean of Y)
denominator = (mean of X^2) - (mean of X)*(mean of X)

Using the computed slope, the intercept can be calculated
intercept = (mean of Y) - slope * (mean of X)

Approach 2: Gradient Descent of RSS(w)

Remember the gradient for our residual sum of squares function is given by;

$$\nabla RSS(w_0, w_1) = \begin{bmatrix} -2\sum_{i=1}^{n} y_i - (w_0 + w_1 x_i) \\ -2\sum_{i=1}^{n} y_i - (w_0 + w_1 x_i)\, x_i \end{bmatrix}$$

y_i is our observed value. In our house example, it is the actual price of the house i.

$(w_0 + w_1 x_i)$ is the function of the line that produces a prediction $\hat{y}_i$ given a value for x_i. In the case of our house example, x_i would be the square footage of the house i.

We can make this notation more succinct by substituting the line equation with the notation for the predicted value for y;

$$\hat{y}_i = \hat{y}_i(w_0, w_1) = (w_0 + w_1 x_i)$$

or for the t-th iteration of the gradient descent the prediction is;

$$\hat{y}_i^{(t)} = \hat{y}_i\left(w_0^{(t)}, w_1^{(t)}\right) = \left(w_0^{(t)} + w_1^{(t)} x_i\right)$$

So re-writing the gradient for the RSS function;

$$\nabla RSS(w_0, w_1) = \begin{bmatrix} -2\sum_{i=1}^{n} y_i - \hat{y}_i^{(t)} \\ -2\sum_{i=1}^{n} y_i - \hat{y}_i^{(t)}\, x_i \end{bmatrix}$$

So the gradient descent algorithm is

While not converged:

$$\begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \end{bmatrix} \leftarrow \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \end{bmatrix} - \eta \begin{bmatrix} -2\sum_{i=1}^{n} y_i - \hat{y}_i^{(t)} \\ -2\sum_{i=1}^{n} y_i - \hat{y}_i^{(t)}\, x_i \end{bmatrix}$$

We can move the constant -2 out and get final formulation for the **gradient descent** calculation

```
Simple Linear Regression Gradient Descent

While not converged:
```

$$\begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \end{bmatrix} \leftarrow \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \end{bmatrix} + 2\eta \begin{bmatrix} \sum_{i=1}^{n} y_i - \hat{y}_i^{(t)} \\ \sum_{i=1}^{n} y_i - \hat{y}_i^{(t)}\, x_i \end{bmatrix}$$

Closed-form vs Gradient Descent

- In many cases, there is no closed form for the gradient, so Gradient Descent is required.
- In some cases, we have many, many coefficients, so calculating the the closed form is difficult even if there is a closed form.
- The advantage of closed form is that it is not iterative. We do not need to choose an initial value for the iteration and we do not need to choose a step size. We simply calculated it directly.

Influence of High Leverage points

A **high leverage point** is point along the x-axis that is an outlier. This may or may not affect the fit. It has the potential, but only influences the fit if it is outside the trend of the other data.
An **influential observation** is one where the removal of the point significantly changes the fit. So a high leverage point may be an influential observation and we can test that by taking it out of the dataset and recalculating.

It is important when fitting data, to look at the data (make visual plots) to see if there are influential observations that you may want to remove to get a better fit for predictive purposes.

Asymmetric Cost Functions
What if the cost of over-estimating if different than the cost of under-estimating? In this case we can use something other that RSS(w) to calculate the cost of a fit.