1 Week 2 - Gradient Descent

We want to find θ_0, θ_1 that will minimise our cost function

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

We will use gradient descent to do that. Here is the idea: Imagine that you are in a hilly landscape blindfolded, and you want to get to the lowest elevation around. What would you do? You'd probably tap around your immediate area, and then move to the lowest elevation that you found. Then you'd repeat until you tapped around your immediate area and found that the ground around you to all be higher or on the same level. That is what gradient descent does.

Our plan:

- Pick some θ_0, θ_1 (Perhaps $\theta_0 = 1, \theta_1 = 1$, but it doesn't really matter).

- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$, until we have found a minimum

More formally:

Data: A set of points to which we are fitting a line
$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})$
Result: θ_0, θ_1, parameters for the line that fits the input data
set some initial values for θ_0, θ_1
while *we have not reached convergence* **do**

$$\theta_0 := \theta_0 - \alpha \frac{\delta}{\delta \theta_0} J(\theta_0, \theta_1)$$

$$\theta_1 := \theta_1 - \alpha \frac{\delta}{\delta \theta_1} J(\theta_0, \theta_1)$$

end
return θ_0, θ_1
       Algorithm 1: algorithm for **gradient descent**

We will update θ_0, θ_1 simultaneously.

α is the learning rate, and it represents the size of the step you take in each iteration. More on that later. $\frac{\delta}{\delta \theta_0} J(\theta_0, \theta_1)$ and $\frac{\delta}{\delta \theta_1} J(\theta_0, \theta_1)$ are partial

derivatives for the cost function, and they represent the direction we need to go if we want $J(\theta_0, \theta_1)$ to decrease in the next iteration. Here is the algorithm with the partial derivatives for our cost function subbed in:

Data: A set of points to which we are fitting a line
$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})$
Result: θ_0, θ_1, parameters for the line that fits the input data
set some initial values for θ_0, θ_1
while *we have not reached convergence* **do**

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \times x^{(i)}$$

end
return θ_0, θ_1
        Algorithm 2: algorithm for **gradient descent**

1.1 Gradient Descent - how do we detect convergence?

The number of iterations we need will vary, but note that:

- For sufficiently small α, $J(\theta)$ should decrease on every iteration

- But if α is too small, gradient descent is *slow*. It can also encourage getting stuck in a local minima.

The best way to know when we can stop iterating is to keep track of $J(\theta)$ on each iteration, and then plot it like the following:
We can tell gradient descent is working if it decreases on each iteration, and at the point where the cost seems to level off is when we can stop iterating.
You can also use these plots to spot if something has gone wrong.
This usually means that our α is too big. We are probably overshooting the minimum. To go back to the analogy of getting to the lowest elevation, imagine that your steps are so large that it's like your landscape is actually a puddle, and you just keep stepping over it.
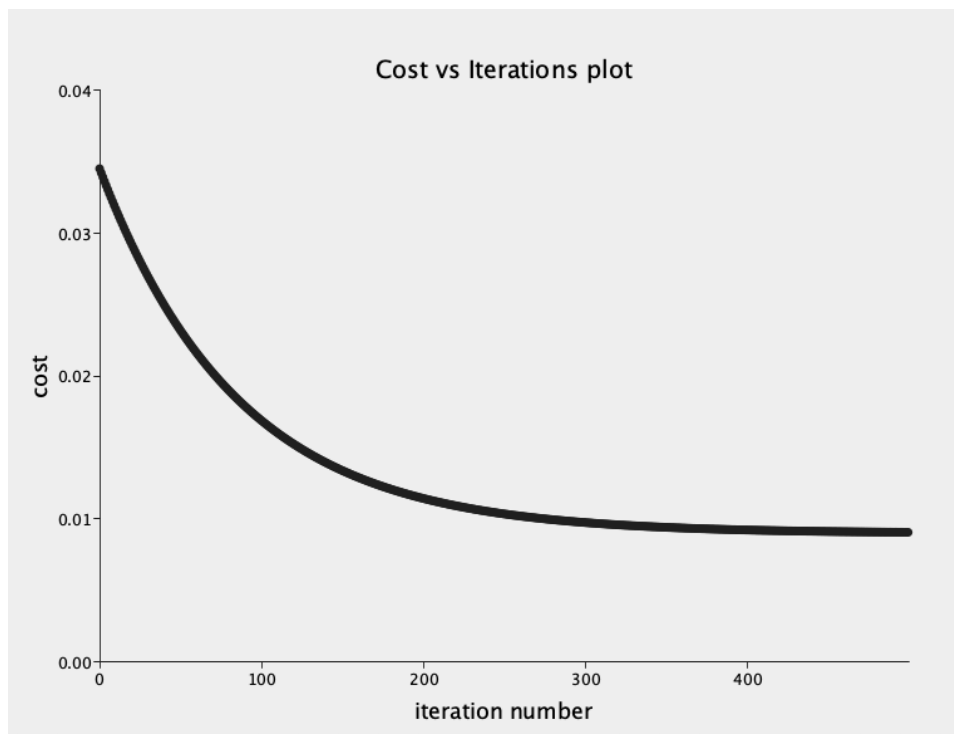
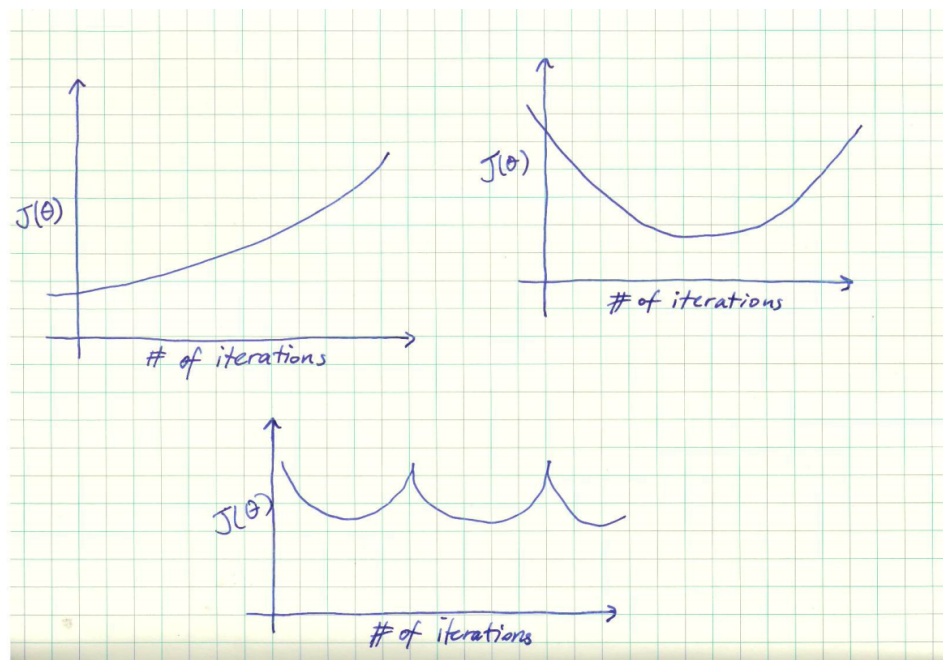Figure 1: A cost vs iterations plot where gradient descent has converged.

Figure 2: Some cost vs iterations plots where gradient descent has diverged.

1.1.1 Feature scaling

Imagine we are building a model where we and we have a feature x_1 which corresponds to age, and another x_2 which is annual income in pounds. In our dataset, the range of x_1 might be 18-65, and x_2 could be £10K-£500K. These features have very different ranges! If we tried to run gradient descent with this dataset as is, we'd end up trying to traverse a warped landscape. This is going to:

- Make our run of gradient descent *slow*

- Make it harder to choose *alpha* and the number of iterations needed

We can speed up gradient descent by ensuring that our data falls into similar ranges, while maintaining the shape of the data. In our case with one feature, LotArea, LotArea and selling price seem to have a linear relationship. After scaling LotArea, it will still have a linear relationship with selling price.

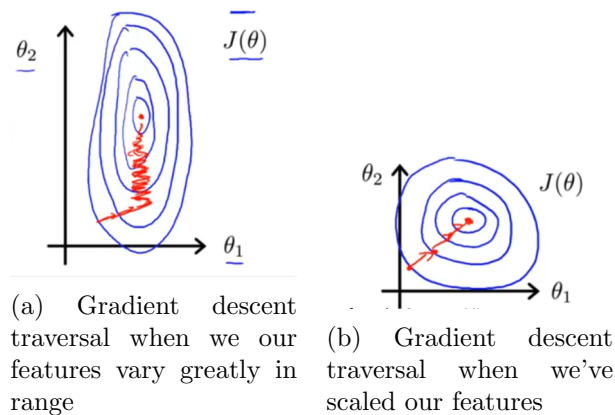One common technique is *mean normalisation*

(a) Gradient descent traversal when we our features vary greatly in range

(b) Gradient descent traversal when we've scaled our features

Figure 3: Feature scaling for gradient descent. Figures from Andrew Ng's ML course on Coursera.

For some feature x_i, we find the mean μ_i and the range s_i (max value - min value). Then for each point j in x_i:

$$x_i^{(j)} \leftarrow \frac{x_i^{(j)} - \mu_i^{(j)}}{s_i^{(j)}}$$

and then for each $x_i^{(j)}$, $-0.5 \leq x_i^{(j)} \leq 0.5$

Mean normalisation isn't the only approach to take for feature scaling, but it will work well enough for our purposes here. *Min-max scaling* and *unit vector scaling* are also common, but we won't cover them here.

//TODO: how feature scaling fits into the overall model of training set $\rightarrow$ h $\rightarrow$ predictions