## Feature scaler

$X = \begin{bmatrix} 1 & 1 & 1 & \\ 9 & 223 & 740 & \\ 6120 & 1147 & 9313 & \cdots \\ 2 & 3 & 3 & \\ 2 & 2 & 2 & \\ 0 & 1 & 1 & \\ 1774 & 1552 & 1728 & \end{bmatrix}$

→ Add in $x_0$ after you've scaled $x_1, \ldots, x_n$.

→ for m training examples

1) Create a vector of length n that contains the mean for each feature

2) Likewise for range (max − min)

3) For each feature j for each training example, subtract mean for feature j and divide by range for j.

every feature in every training example will be updated

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

with the following

Linear mean squared error: See Alex's post in Digital/Learning groups.

data/house-prices-train.csv NumberOfBathrooms
Id, LotArea, NumberOfBedrooms, NumberofKitchens, SquareFeet, SalePrice
9, 6120, 2, 2, 0, 1774, 129900
223, 11473 0, 3, 2, 1, 1552, 179900
740, 9313, 3, 2, 1, 1728, 190 000
⋮ 1168 training examples

**Data:** An ~~$m \times n$~~ $n \times m$ matrix with $n$ features and $m$ training examples, $y$, a $m \times 1$ matrix
**Result:** $\theta$ a $n \times 1$ matrix of parameters for the line that fits the input data
set some initial values for $\theta_0, \theta_1, ..., \theta_n$
**while** *we have not reached convergence* **do**
  **for** *each $j$ in range 0 up to $n$* **do**
    $$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \times x^{(i)}_j$$
  **end**
**end**
**return** $\theta_0, \theta_1, ..., \theta_n$

**Algorithm 1: gradient descent**

vectorised theta update (see: Vectorised Gradient Descent.scala)

$X_0$ is always 1

$$X = \begin{bmatrix} 1 & 1 & 1 & \\ 9 & 223 & 140 & \\ 6120 & 1147 & 9313 & \cdots \\ 2 & 3 & 3 & \\ 2 & 2 & 2 & \\ 0 & 1 & 1 & \\ 1774 & 1552 & 1728 & \end{bmatrix} \rightarrow \text{for 1168 training examples}$$

$$y = \begin{bmatrix} 129\,900 \\ 179\,900 \\ 190\,000 \\ \vdots \end{bmatrix}$$

Note: you will apply feature scaling to X ~~[illegible]~~ before running gradient descent

We are maintaining a $n \times 1$ matrix of our parameters. Before we begin iterating we set $\theta = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$

The input to vectorised theta update is
- X dimensions: $n \times m$
- y dimensions: $m \times 1$
- $\theta$ (our current values for our parameters) dimensions: $n \times 1$

The output to vectorised theta update is
- $\theta$ (our current values for our parameters; they are slightly better than in the last iteration) dimensions: $n \times 1$

**Data:** An $n \times m$ matrix with $n$ features and $m$ training examples, $y$, a $m \times 1$ matrix
**Result:** $\theta$ a $n \times 1$ matrix of parameters for the line that fits the input data
set some initial values for $\theta_0, \theta_1, ..., \theta_n$
**while** *we have not reached convergence* **do**
  **for** *each $j$ in range 0 up to $n$* **do**

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

  **end**
**end**
**return** $\theta_0, \theta_1, ..., \theta_n$

**Algorithm 1: gradient descent**

update parameters $\theta_0, \theta_1, \ldots \theta_n$ simultaneously

when you see this Sum, think "for each training example"

X: 6 x 1168 matrix
Θ: 6 x 1 matrix
y = 1168 x 1 matrix

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$$
$$= \theta^T x$$

$\theta^T x$ is a 1 x 1168 matrix representing predicted value for each training example

Then you want a new matrix of the same dimension that subtracts y from predicted value. Let's call it pd, prediction difference, pd is a 1 x 1168 matrix.

pd does not change based on which theta we're interested in calculating.

To get $\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$ ← The Sum we could extract $x_j$, a 1 x 1168 matrix. (then treating $x_j$ and pd as vectors) and take the dot product $x_j \cdot pd$.

Or, if we multiply $X_j$ a 6 x 1168 matrix with $pd^T$, a 1168 x 1 matrix, we'll have a 6 x 1 matrix that has The Sum for each parameter. Then we just need to multiply (scalar) by $\alpha \frac{1}{m}$, The Sum matrix.

That will give us a 6 x 1 matrix of theta updates.

Return Θ - {theta updates matrix}.