Batch Gradient Descent Algorithm:

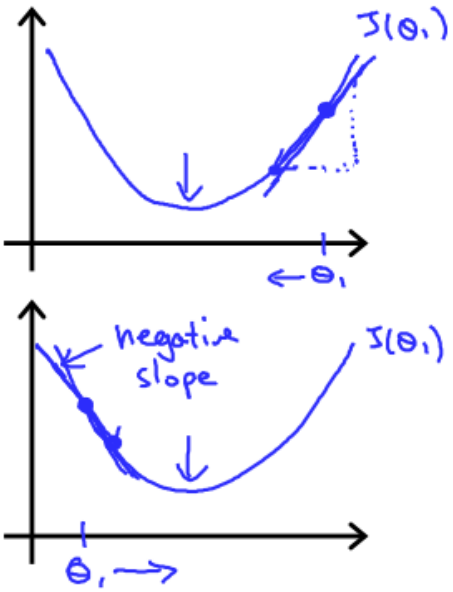
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

Dove:

- Batch perchè ad ogni iterazione tiene conto di tutte le j-esime entry del training set
- j: indice del' esempio;
- α: learning rate. E' un numero sempre positivo che definisce l' ampiezza ampiezza passo che facciamo durante la discesa del gradiente nella ricerca del minimo locale;
- J(θ0,θ1) = la funzione di costo **(Errore quadratico medio).** Per ottenere una funzione che sia vicina ai dati dobbiamo minimizzare l' errore (h0(x) - y)^2. Questo però dobbiamo farlo per tutti gli elementi del training set, ovvero:

$$\frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

- $\frac{\partial}{\partial \Theta j}$: la derivata servi a capire in che direzione muoversi. Se è positiva (siamo in fase crescente) allora θ verrà decrementata (perché noi vogliamo minimizzarne il valore). Se è invece è negativa, significa che siamo in una fase di discesa e allora dobbiamo procedere in quella direzione e θ verrà incrementata (il negativo della derivata si combinerà con il segno negativo della formula e diventerà positivo).



Riscrivendo la formula, senza derivate otteniamo:

Gradient descent algorithm

repeat until convergence {

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

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

}

Attenzione: eseguire l' algoritmo simultaneamente, ovvero non aggiornare θ0 e poi usare il nuovo θ0 nell' equazione di θ1. Utilizzare delle variabili temporanee per allocare i "Nuovi θ".

Correct: Simultaneous update

$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$
$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$
$\theta_0 := \text{temp0}$
$\theta_1 := \text{temp1}$

Incorrect:

$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$
$\theta_0 := \text{temp0}$
$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$
$\theta_1 := \text{temp1}$

Versione Con più di una features

L'algoritmo finora descritto, si riferisce ad un sistema in cui è presente una sola feature. Nei problemi reali, saranno rese disponibili più features da calcolare. Adattiamo quindi la formula in modo tale che possa elaborare più features:

New algorithm $(n \geq 1)$:

Repeat {

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

(simultaneously update θ_j for $j = 0, \ldots, n$)

}

Esempio di implementazione in Matlab:

Discesa del Gradiente:

```
errors = (X * theta -y); %Ci calcoliamo a parte il risultato del' errore
quadratico medio

tempThera = theta; %ci salviamo theta per effettuare l' update
simultaneo

tempTheta = theta - (alpha * 1/m * sum(errors .* X))'; %aggiorniamo
tutti i valori di theta secondo la formula sopra descritta (qui si
capisce meglio il perché ci siamo salvati il valore in tempTheta, ovvero
per non sporcare il contenuto di theta che usiamo all' interno della
formula)

theta = tempTheta; %aggiorniamo tutti i valori di theta
```

Calcolo funzione costo:

```
predictions = X * theta; %ovvero Hθ o equivalentemente ŷ
sqrErrors = (predictions-y).^2;

J = 1/(2*m) * sum(sqrErrors)
```

Versione ottimizzata:

```
J = 1/(2*m) * (predictions-y)' * (predictions-y);
```

Trick di ottimizzazione:
1) Feature scaling & Mean Normalization: Invece di lavorare con numeri spropositati, si cerca di “normalizzare” il valore delle varie features. Il range di valori per ogni feature dev rimanere in un intorno di valori vicini al range da -1 a 1.

- Va bene anche tipo da -3 a +3 ma non -1000 a +1000
- Non deve neanche essere troppo piccolo però, valori come 0.000001 rendono comunque complessi i calcoli.

Generalmente si assegna al valore di una feature in questo modo:
Feature scaling: si divide ogni valore per “max value - min value”
Mean Normalization: sottrarre ad ogni valore la media di tutti i valori

Combinando insieme le due tecniche otteniamo la seguente formula di update:

$$x_i = \frac{x_i - \mathbf{AVG}(x)}{x_{max} - x_{min}}$$

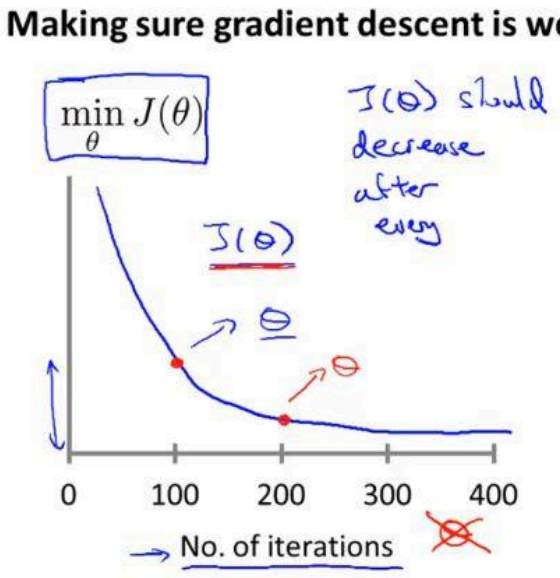
Ci sono più modi ottenere lo scalign, ad esempio utilizzando il valore min invece di AVG, oppure utilizzando la deviazione standard invece di max-min.

Esempio di implementazione in Matlab:

```
mu = mean(X); %calcolo AVG una sola volta e mi tengo da parte il risultato
sigma = std(X); %calcolo la dev.std (denominatore) e la tengo da parte
X_norm = (X - mu)./sigma; %per normalizzare applico la formula sopradescritta
```

2) Debugging:

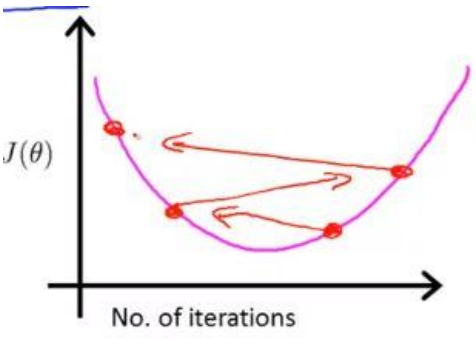
Per assicurarci che la discesa del gradiente stia procedendo correttamente possiamo crearci un grafico che metta in relazione il numero di iterazioni sull' asse delle x, ed il valore di J(Θ) sull' asse delle Y. Ci aspettiamo una continua e costante discesa, se cosi non è, significa che c'è un problema.



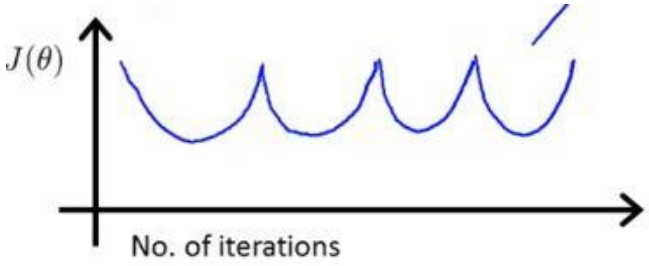
Inoltre da questo esempio di grafo capiamo che non c'è più molto margine di miglioramento dopo le 400 iterazioni, quindi possiamo interrompere li l' apprendimento.

3)Scegliere *α*:

Un caso particolare è quello in cui il valore di J(Θ) aumenta. Questo può essere causato da una scelta errata del learning rate (***α***), che è troppo alto e causa il "rimbalzare" in un intorno del minimo locale, senza mai riuscire a centrarlo a causa appunto di passi troppo ampi.



Stesso discorso con un andamento non costante, tipo:



In generale conviene partire da piccoli valori, fare prove e pian piano modificarli. Non per forza devono essere potenze di 10.

Un buon set di ***α*** possono essere: 0.003, 0.03, 0.3, 1,3

Normal Equation calcola direttamente il valore esatto attraverso l' uso di matrici,come nella seguente formula:

$$\theta = (X^T X)^{-1} X^T y$$

Esempio di implementazione in Matlab:

```
pinv(X'*X) * X'*y;
```

Gradient Descend	VS	**Normal Equation**
Necessita di un parametro α		Non necessita di α
Richiede più iterazioni dell' algoritmo		Calcola il risultato in una sola iterazione
Richiede il "features scaling"		Non richiede il "features scaling"
Costa $O(kn^2)$		Costa $O(n^3)$(Invertire le matrici costa)

In definitiva la Normal Equation conviene fino ad un certo limite di features nel dataset. Indicativamente con numeri maggiori di 10000 conviene usare la discesa del gradiente.

Thanks to: Andrew Ng [Coursera]