Machine Learning algorithms

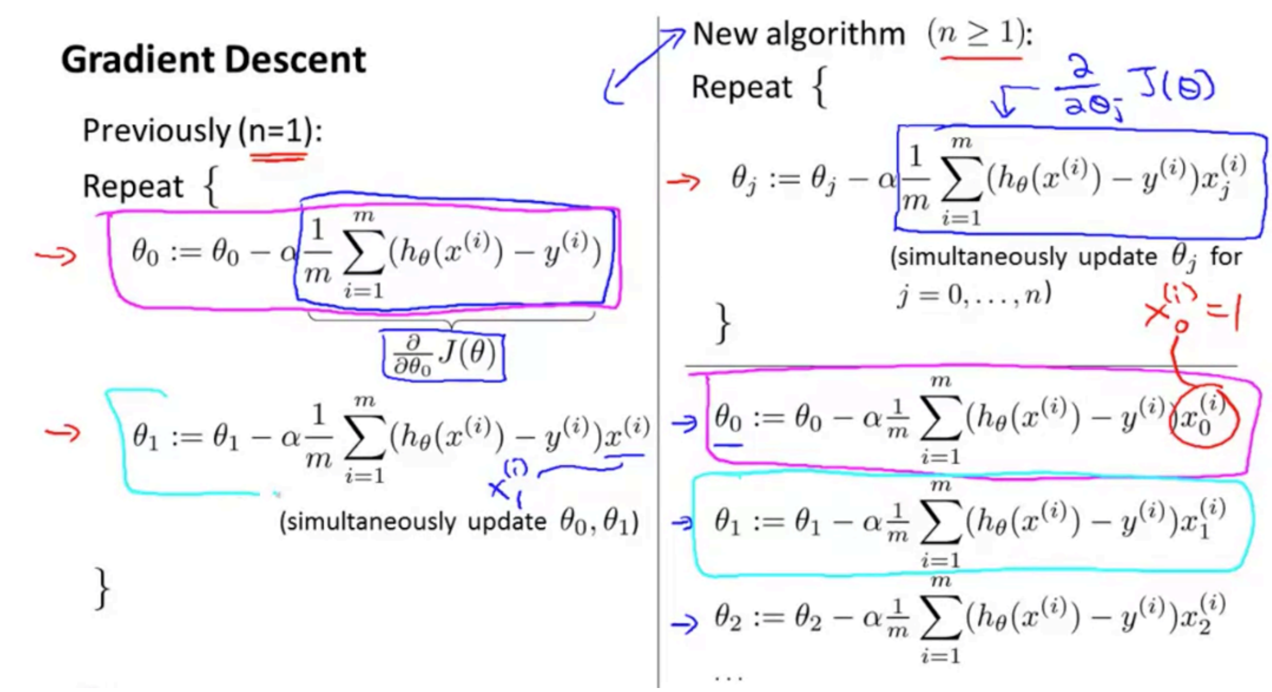
Machine learning:

“The field of study that gives computers the ability to learn without being explicitly programmed.” by Arthur Samuel

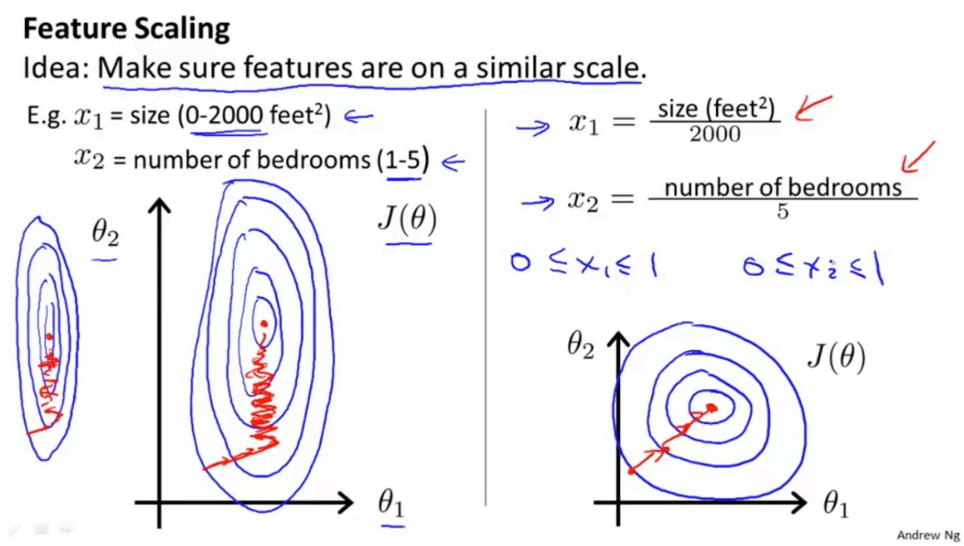
“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E”

Linear regression

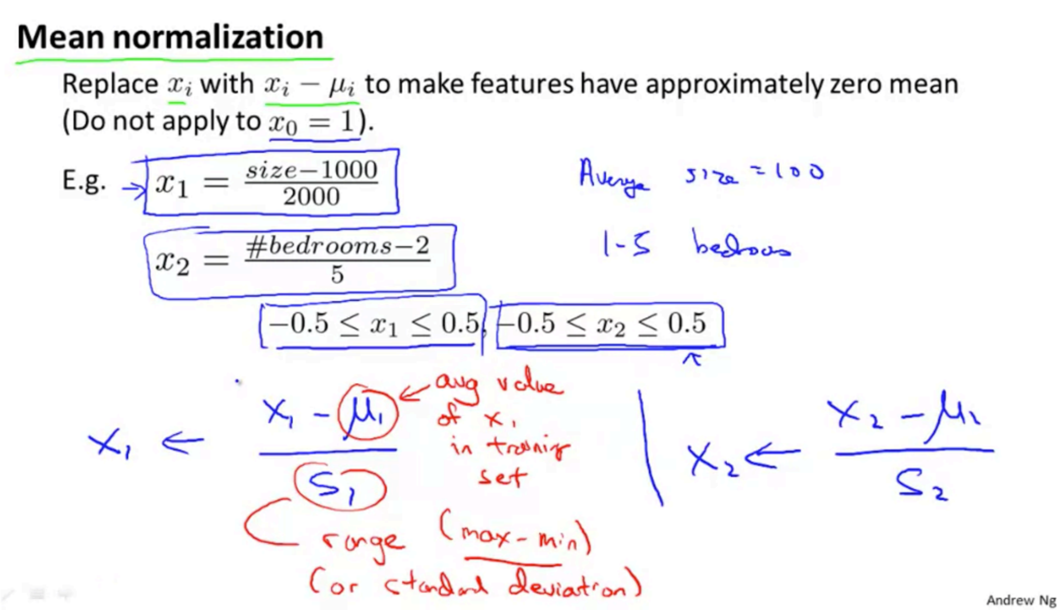
- **Linear regression**: predict a real-value output based on input value. For example, house price prediction
- **Multivariate linear regression**: predict a real-value output based on multiple input values.
- **Cost function**: mean square error (MSE)



- **Practical tricks for Gradient descent**:
    - **Feature scaling**: make sure features are on a similar scale.
        - Gradient descent has a hard time to find its way and take a much more convoluted path to the minimum if features are not in the same scale.
        - On the contrary, if the features are in the same scale, Gradient descent takes a much simpler and more direct path to the minimum



        - Get every feature into approximately a [-1 1] range, [-3 3], [-1/3 1/3] range is still fine, no worries if the features are not in the exact same range
        - Mean normalization: to make features with zero mean



- How to make sure **gradient descent is working correctly**?
  - Cost function J(θ) should decrease after every iteration (plot J(θ) vs no. of iterations)
    - If J(θ) is not decreasing, probably learning rate α is too large, should decrease the learning rate α
    - If α is small enough, J(θ) should decrease on every iteration
    - But if α is too small, then it will take a long time to converge
    - Try a number of α, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1…, plot J(θ) vs no. of iterations
  - Automatic convergence test: declare convergence if J(θ) decreases by less than a certain threshold after every iteration (the threshold is hard to choose most of the time)
- **Polynomial regression**: We can **change the behavior or curve** of our hypothesis function by making it a quadratic, cubic or square root function (or any other form).
  - **Feature engineering**:
    - create new features based on the problem
    - Feature scaling is important with respect to new features (new features might be in a much larger or smaller magnitude)
- **Normal equation method** is another alternative for gradient descent

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

Gradient Descent	**Normal Equation**
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
O (kn^2)	O (n^3), need to calculate inverse of $X^T X$
Works well when n is large	Slow if n is very large

- Noninvertible (singular)
  - Causes
    - redundant features (linearly dependent)
    - Too many features (No. Training samples less than No. of features)
  - Pseudo inverse will solve this problem
  - Inverse
- **Coding steps**

Simple linear regression

Hypothesis : $h_\theta(x) = \theta_0 + \theta_1 x$

parameters/weights : θ_0, θ_1

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

Find the parameters that'll minimize cost function $J(\theta_0, \theta_1)$:

☆ Gradient descent algorithm

△ starting with random θ_0, θ_1

△ Updates the weights until reach minimum of $J(\theta_0, \theta_1)$

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad \begin{array}{l} j=0, \text{ for } \theta_0 \\ j=1, \text{ for } \theta_1 \end{array}$$

Note: weights should be updated simultaneously.

e.g. $\theta_{0,next} = \theta_{0,current} - \alpha \frac{\partial}{\partial \theta_0} J(\theta_{0,current}, \theta_{1,current})$

$\theta_{1,next} = \theta_{1,current} - \alpha \frac{\partial}{\partial \theta_1} J(\theta_{0,current}, \theta_{1,current})$

☆ Normal equation method

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