Machine Learning

Week One

What is ML?

- Arthur Samuel described it as: "the field of study that gives computers the ability to learn without being explicitly programmed." This is an older, informal definition.
- Tom Mitchell provides a more modern definition: "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."
    - Example: playing checkers.
    - E = the experience of playing many games of checkers
    - T = the task of playing checkers.
    - P = the probability that the program will win the next game.
- ML Algorithms: supervised learning, unsupervised learning, reinforcement learning, recommender systems.

Supervised Learning (right answers given)

- In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.
- (a) Regression - Given a picture of a person, we have to predict their age on the basis of the given picture. Continuous output
- (b) Classification - Given a patient with a tumor, we have to predict whether the tumor is malignant or benign. Discrete output

Unsupervised Learning

- Unsupervised learning allows us to approach problems with little or no idea what our results should look like.

- With unsupervised learning there is no feedback based on the prediction results.

- Clustering: Take a collection of 1,000,000 different genes, and find a way to automatically group these genes into groups that are somehow similar or related by different variables, such as lifespan, location, roles, and so on.

- Non-clustering: The "Cocktail Party Algorithm", allows you to find structure in a chaotic environment. (i.e. identifying individual voices and music from a mesh of sounds at a cocktail party). `[W,s,v]=svd((repmat(sum(x.*x,1),size(x,1),1).*x)*x');`

Model Representation

- x^i denote as “input” variable and y^i denote as the output variable. (x^i, y^i) is called a training example. When y is continuous, regression; if y is discrete, classification.

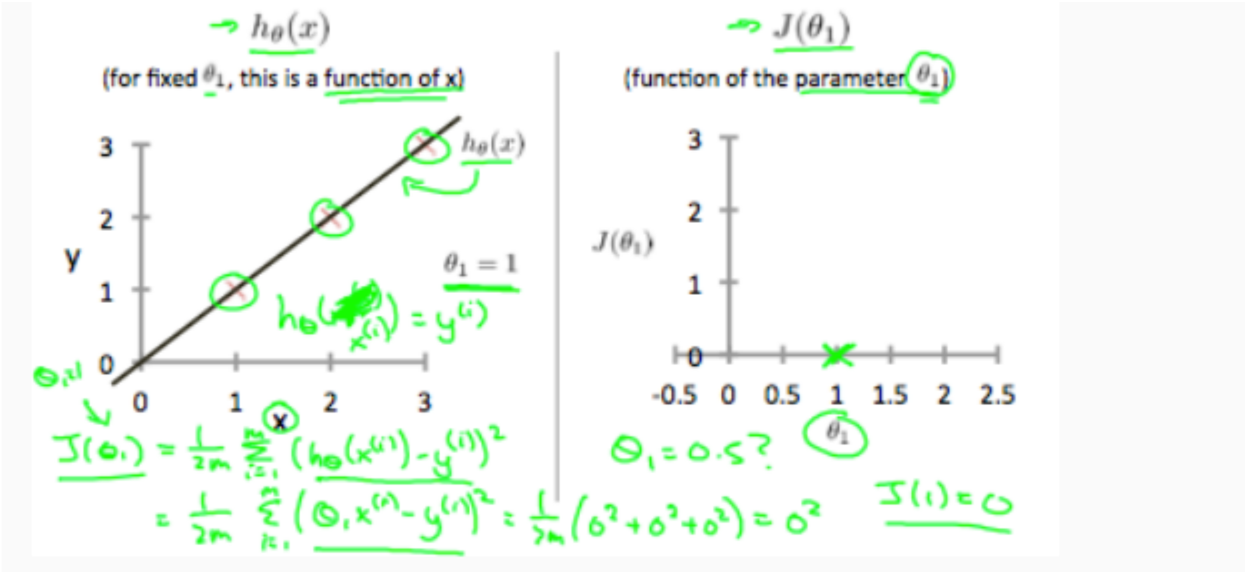
Cost Function

Hypothesis:

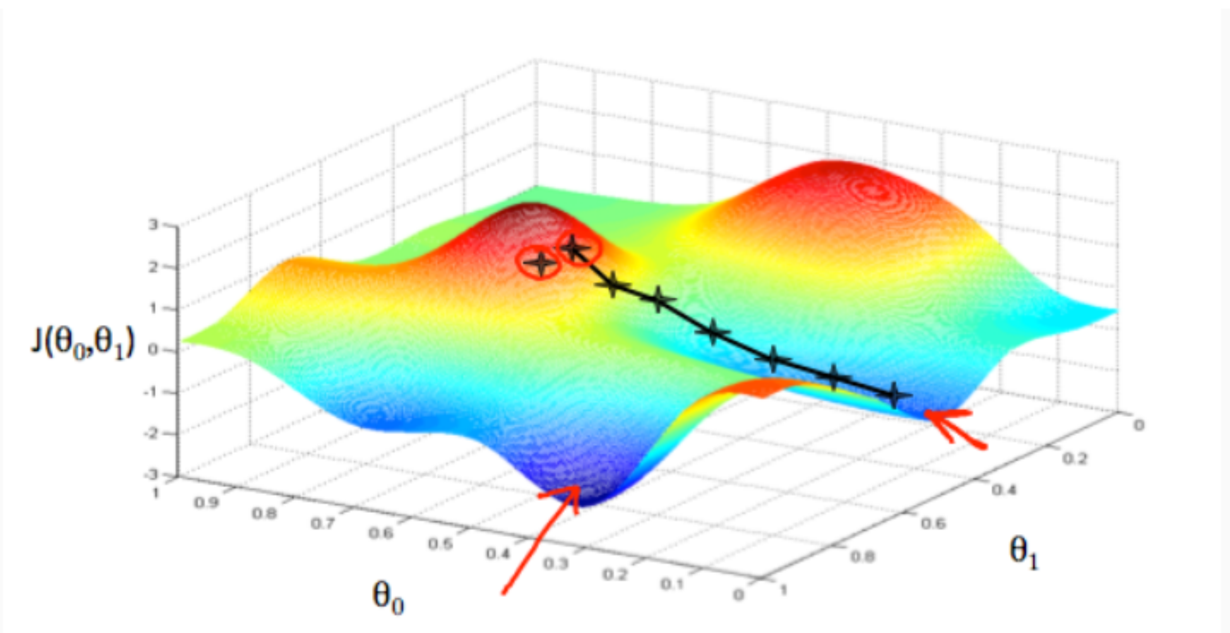
$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}_i - y_i\right)^2 = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x_i) - y_i\right)^2$$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}} \; J(\theta_0, \theta_1)$

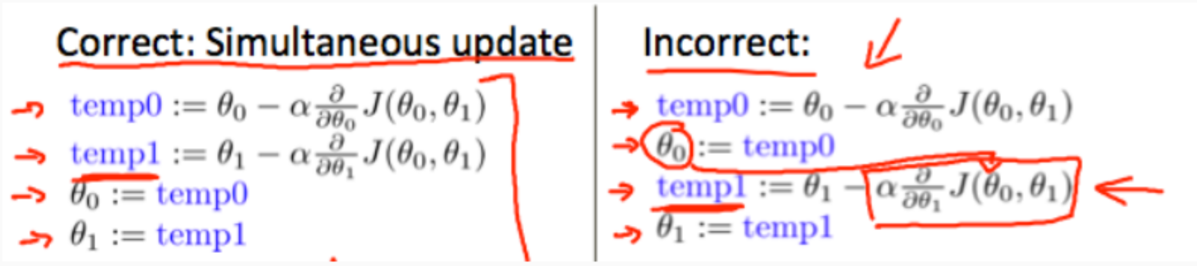


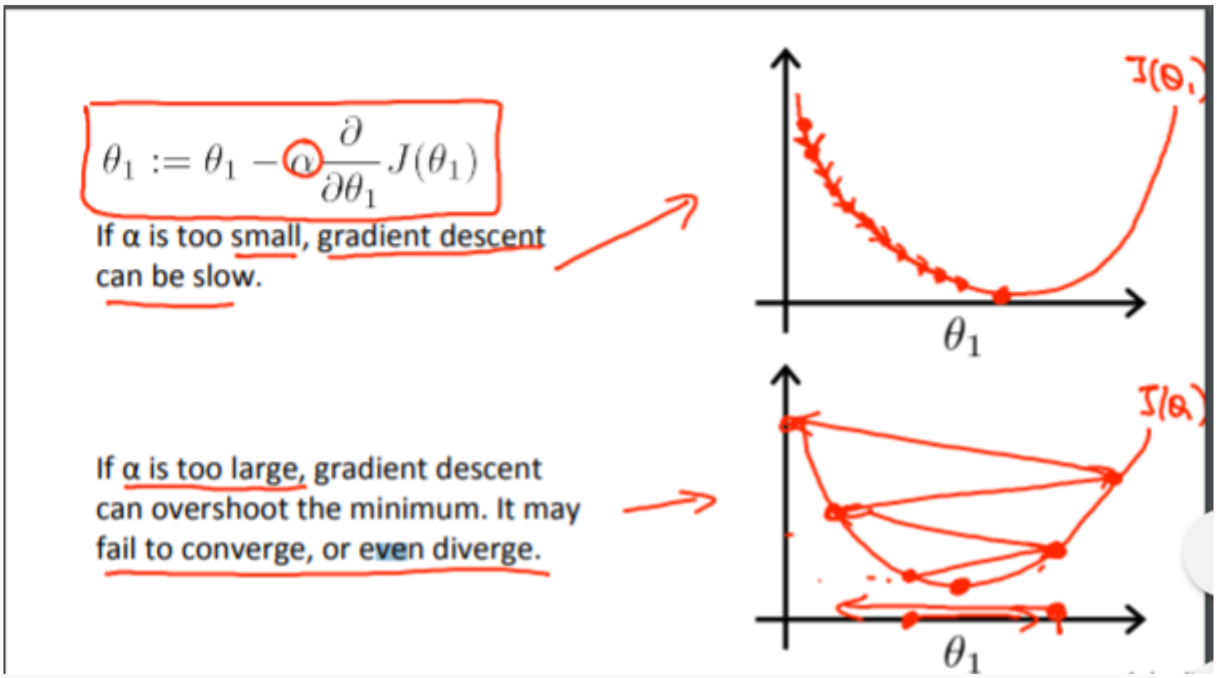
Gradient Descent

- 

-

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\qquad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad \text{(simultaneously update } j = 0 \text{ and } j = 1) \\
&\}
\end{aligned}
$$

- 



- **batch gradient descent**

repeat until convergence: {

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

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(x_i) - y_i) x_i)$$

}

- Suppose that for some linear regression problem (say, predicting housing prices as in the lecture), we have some training set, and for our training set we managed to find some θ_0, θ_1 such that $J(\theta_0,\theta_1)=0$.

False	Gradient descent is likely to get stuck at a local minimum and fail to find the global minimum.

- Let f be some function so that

  $f(\theta_0, \theta_1)$ outputs a number. For this problem,

  f is some arbitrary/unknown smooth function (not necessarily the

  cost function of linear regression, so f may have local optima).

  Suppose we use gradient descent to try to minimize $f(\theta_0, \theta_1)$

  as a function of θ_0 and θ_1. Which of the

False	If θ_0 and θ_1 are initialized so that $\theta_0=\theta_1$, then by symmetry (because we do simultaneous updates to the two parameters), after one iteration of gradient descent, we will still have $\theta_0=\theta_1$.

- Consider a hypothetical cost function which is J=0.5 θ_1^2 and the gradient descent is θ_j=θ_j - a*d/dj (J) ==> θ_0= θ_0 and θ_1 = θ_1 - a*θ_1.

Week Two

Multivariate Linear Regression

- Multiple Features

Multiple features (variables).

Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

$m = 47$

Notation:

- n = number of features $\quad n = 4$
- $x^{(i)}$ = input (features) of i^{th} training example.
- $x_j^{(i)}$ = value of feature j in i^{th} training example.

$$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$$

$$x_3^{(2)} = 2$$

- $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n$

$$h_\theta(x) = \begin{bmatrix} \theta_0 & \theta_1 & \ldots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

Gradient Descent

$$
\begin{aligned}
&\text{repeat until convergence: } \{ \\
&\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)} \\
&\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)} \\
&\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_2^{(i)} \\
&\cdots \\
\}
\end{aligned}
$$

$$
\begin{aligned}
&\text{repeat until convergence: } \{ \\
&\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad \text{for j := 0...n} \\
\}
\end{aligned}
$$

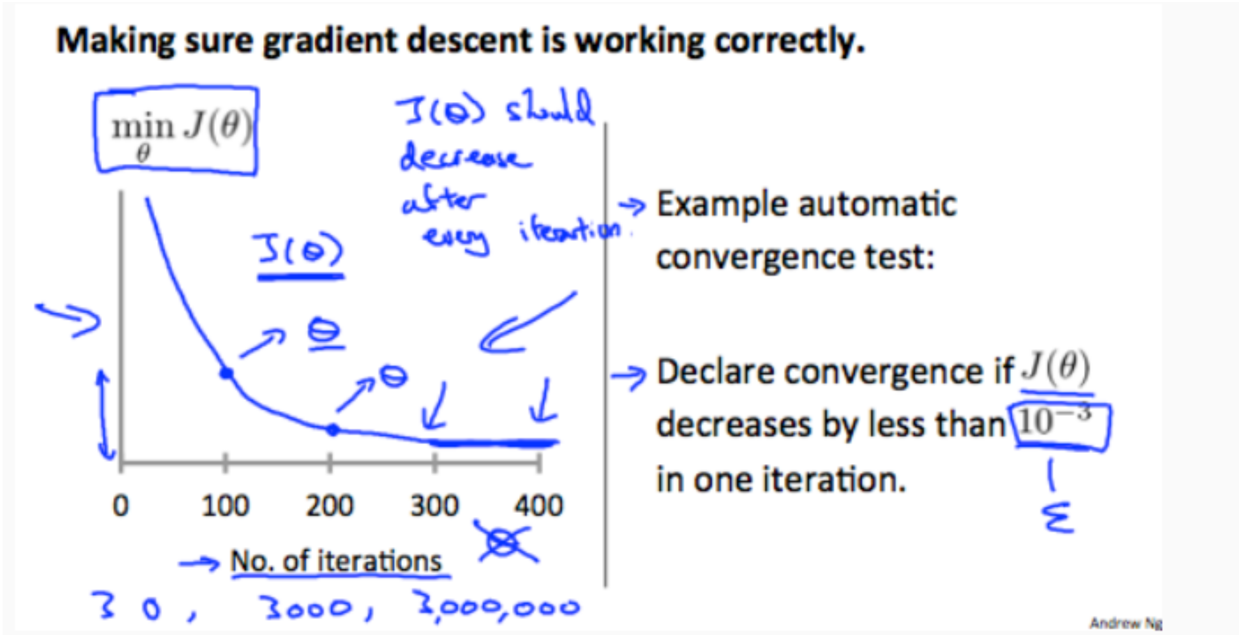
Feature Scaling & Mean Normalization

$$x_i := \frac{x_i - \mu_i}{s_i}$$

,

Where μ_i is the **average** of all the values for feature (i) and s_i is the range of values (max - min), or s_i is the standard deviation.

Learning Rate



If α is too small: slow convergence.

If α is too large: may not decrease on every iteration and thus may not converge.

Polynomial Regression

Computing Parameters Analytically

Normal Equation

Examples: $m = 4$.

	Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
x_0	x_1	x_2	x_3	x_4	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

m × (n+1)

m-dimensional vect

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

θ is an (n+1)-vector. And this θ will minimize J

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^TX
Works well when n is large(10^6)	Slow if n is very large

There is **no need** to do feature scaling with the normal equation.

The cost function J(θ) for linear regression has no local optima.

Matlab tutorial

Vectorization example.

$$h_\theta(x) = \sum_{j=\theta}^{n} \theta_j x_j$$

$$= \theta^T x$$

Unvectorized implementation

```
prediction = 0.0;
for j = 1:n+1,
  prediction = prediction +
               theta(j) * x(y)
end;
```

Vectorized implementation

```
prediction = theta' * x;
```