Machine Learning

Stanford | Andrew Ng

Summer 2017

Week 1: 5/10/17

1. Introduction

Machine learning algorithms are used dozens of times per day without us knowing it. A few popular applications are:

- *Google:* Web search ranking pages,
- Facebook or Apple photo tagging tool,
- E-mail spam blocker.

So why is machine learning so prevalent today? It turns out that machine learning is a field that had grown out of the field of AI, or artificial intelligence. In the early days, handling simple task was relatively simple. However, for the most part, we just did not know how to write AI programs to do the more interesting things such as web search or photo tagging or email anti-spam. There was a realization that the only way to do these things was to have a machine learn to do it by itself. So, machine learning was developed as a new capability for computers and today it touches many segments of industry and basic science.

One example of a very useful application is Handwriting recognition. As it turns out, one of the reasons that the Postal system is so inexpensive today is due to a learning algorithm that has learned how to read handwriting. Consequently, most mail is automatically routed to its destination, rather than relying on a human interpreter. In this way, it costs merely a few cents to send a piece of mail thousands of miles.

This example is easily extended to the fields of *Natural Language Processing* (NLP) or *Computer Vision* (CV), which pertain to understanding languages or images. Unfortunately, for these fields to expand and adapt to every user's needs, some customization is required. And, in general, there is no way to write a million different programs for millions of users. The only way to approach this problem is to have the machine learn by itself.

Here are a few more examples:

- Database mining.
  - Large databases from growth of automation & web.
  - Web data, medical records, biology, & engineering.
- Applications that cannot be programmed by hand.
  - Autonomous helicopter, handwriting recognition, Natural Language Processing (NLP), computer vision.
- Self-Customizing programs
  - Eg., Amazon, Netflix, recommendations
- Understanding human learning

1.1. What is Machine Learning?

Currently, there isn't a well-accepted definition of what is and what isn't machine learning. However, two leading definitions are:

Definition 1. ***Arthur Samuel (1959)*** *Field of study that gives computers the ability to learn without being explicitly programmed.*

Definition 2. ***Tom Mitchell (1998):*** *Well-posed Learning Problem: A computer program is said to learn from experience, E, with respect to some task, T, and some performance measure, P, if its performance on, T, as measured by, P, improves with experience, E.*

There are two main types of Machine Learning algorithms:

- **Supervised learning**
- **Unsupervised learning**

These two algorithms are by far the most popular. However, *reinforcement learning* and *recommender systems* are also rather popular.

1.2. Supervised Learning.

Supervised learning is generally the most popular form of learning. Here, 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.

Supervised learning problems are often divided into two categories of problems:

- ***Regression:*** we are trying to *predict* results within a continuous output, meaning that we are trying to map input variables to some continuous function
- ***Classification:*** we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.

Example 1.1 : *Housing Market*

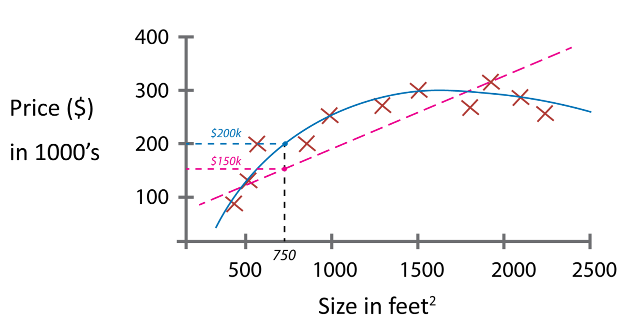
Given data about the size of houses on the real estate market (Fig. 1a), try to predict their price. Price as a function of size is a continuous output, so this is a regression problem. In this example we could use a number of regression algorithms, but this will be the task of the ML algorithm at a later point.

We could turn this example into a classification problem by instead making our output about whether the house *"sells for more or less than the asking price."* Here we are classifying the houses based on price into two discrete categories.

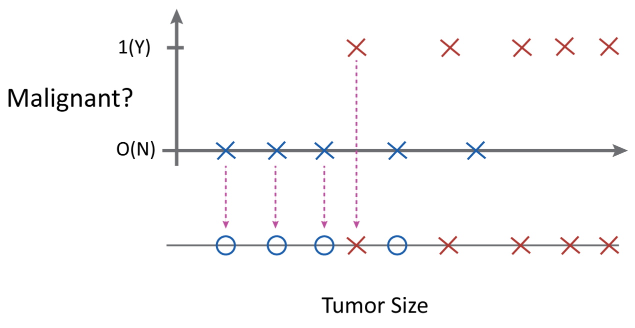
Example 1.2 : *Breast Cancer Identification*

Now, we are looking at medical records and determining if a tumor is malignant or harmless (Fig. 1b).

The ML question to ask is: *What is the chance the tumor is malignant or benign.* This is a classification problem since we are trying to predict a discrete valued output. In this case our output is a value of 0 or 1. However, this could be multiple discrete states: $0, 1, 2, \ldots$.



(a)



(b)

Figure 1. Supervised learning examples: (a) *Regression* and (b) *Classification*

Now, let's say we know the patient's age and tumor size. In this case, we now have an example with '2 features.'

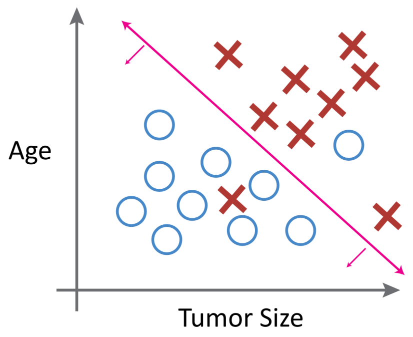


Figure 2. Classification example with 2 features.

In general, we would prefer to use as much information as we have available. Hence, we can include more information, such as:

- Clump thickness
- Uniformity of cell size
- Uniformity of cell shape
- ...

For some learning problems, we can use an *infinite* number of features. *How can we handle this?*

- This can be handled by Support Vector Machines (SVM).

Example 1.3 : *Additional Examples*

(1) ***Regression*** - Given a picture of Male/Female, We have to predict his/her age on the basis of given picture.

(2) ***Classification*** (1) - Given a picture of Male/Female, we predict whether He/She is of High school, College, or Graduate age.

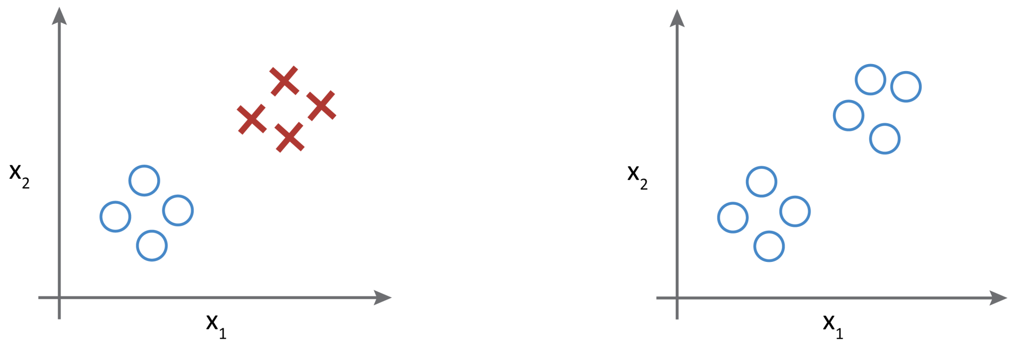
(3) ***Classification*** (2) - Banks must decide whether or not to give a loan to someone on the basis of their credit history.

1.3. Unsupervised Learning.

Unsupervised learning, the other major type of ML algorithms, allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables. In supervised learning we are 'given the right answer.' That is, each example was labeled either as a positive or negative example, whether it was a benign or a malignant tumor (Fig. 3a). In Unsupervised learning, we are not given information on labels. We derive this structure by clustering the data based on relationships among the variables in the data. With unsupervised learning there is no feedback based on the prediction results, i.e., there is no teacher to correct you (Fig. 3b).

The ML question we can ask is: *Can we find some structure in the dataset?*. (Fig. 3b) is an example of a *clustering* algorithm - which is widely used by Google news as well as various DNA identifications. In unsupervised learning, there are two general categories of interest:

- ***Clustering:*** Take a collection of 1000 essays written on the US Economy, and find a way to automatically group these essays into a small number that are somehow similar or related by different variables, such as word frequency, sentence length, page count, and so on.



(a) Supervised **(b)** Unupervised

Figure 3. Classification examples (a) *Supervised* and (b) *Unsupervised*

- ***Non-clustering:*** The *"Cocktail Party Algorithm"*, which can find structure in messy data (such as the identification of individual voices and music from a mesh of sounds at a cocktail party

Other examples include:

- Computing cluster organization
- Social network analysis
- Market segmentation
- Astronomical data analysis

Example 1.4 : *Cocktail Party Problem*

Imagine there's a party room full of people, all sitting around, all talking at the same time and there are all these overlapping voices because everyone is talking at the same time, and it is almost hard to hear the person in front of you. So maybe at a cocktail party with two people.

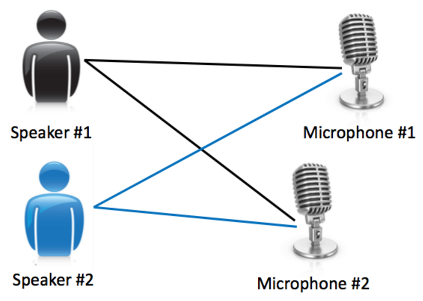


Figure 4. Unsupervised learning example: Cocktail party problem.

Separating out the voices is an example of an unsupervised learning algorithm. This can be done by the following Matlab command:

```
        [W,s,v] = svd((repmat(sum(x.*x,1),size(x,1),1).*x)*x');
```

1.4. Summary.

In this first lecture, we looked at two types of Machine Learning (ML): *Supervised* and *Unsupervised* learning. Fig. 5 below summarizes the two types.

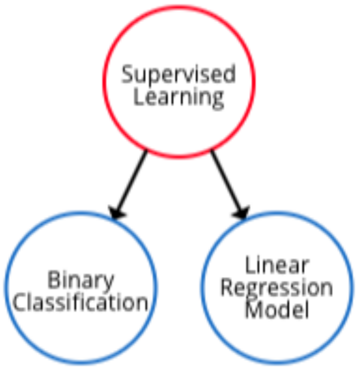
Supervised Learning: Given dataset, already know what output should look like; are 'given' the answers.

- Regression: Given picture, guess age.

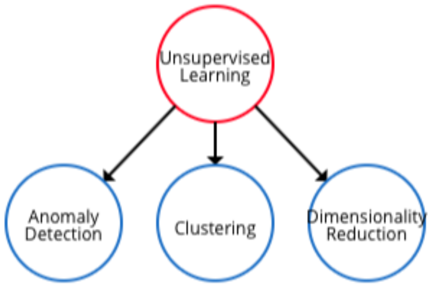
- Classification: predict whether tumor is benign or malignant.

Unsupervised Learning: No idea of what results should be. Derive structure from the data.

- Clustering: Group genes of a person together.
- Non-clustering: Extract voices from music.



(a) Supervised



(b) Unupervised

Figure 5. Types of Machine Learning (ML): (a) *Supervised* and (b) *Unsupervised*

2. Linear Regression with One Variable

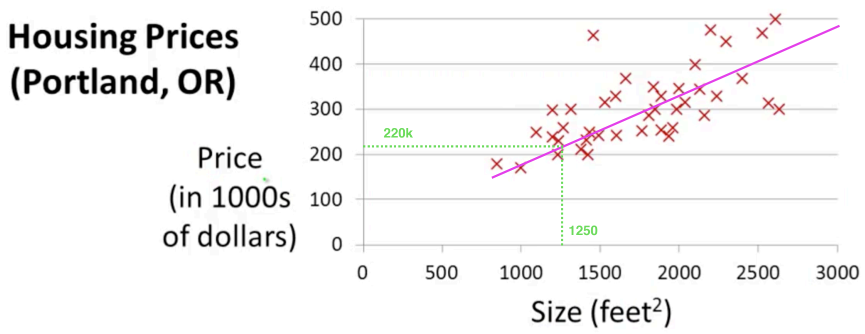
The first learning algorithm that we be a simple *linear regression.* This is one of the most simple, and widely used algorithms, and will be referred to frequently. In *univariate* linear regression, our goal is to learn a mapping from one real-valued space to another. More specifically, we use this model to predict a single output value y from a single input value x. We're interested in supervised learning here, so that means we already have an idea about what the input/output cause and effect should be.

2.1. Model Representation.

To begin, we start off with a motivating example:

Example 2.1 : *Housing Market*

- *What price should your friend sell their house at?*



- This is an example of ***supervised learning*** - since we already had the 'right answers' for each example in the data.

- This is also an example of a ***regression problem*** - which is used to predict a real-valued output.

More formally, in supervised learning, we use a ***training set*** to learn how to predict the housing prices:

Definition 3. ***Training Set*** *A pair $(x^{(i)}, y^{(i)})$ is called a **training example**, and the dataset that we'll be using to learn – a list of m training examples – is called a **training set***

Table 1. Training data example, assume $m = 42$

Size (ft^2) (x)	Price ($\times$1000)(y)
2104	460
1416	232
1534	315
852	178
...	...

Notation:

$$\begin{aligned} m &= \text{Number of training examples} \\ x &= \text{`Input' variable / } \textit{features} \\ y &= \text{`Output' variable / } \textit{target} \text{ variable} \end{aligned}$$

Where:

$$\begin{aligned} (x, y) &= \text{single training example} \\ (x^{(i)}, y^{(i)}) &= i^{th}\text{training example} \end{aligned}$$

Now, once we have this training set, we send it on to the ***Learning Algorithm***; the function of this algorithm is to produce a ***hypothesis function***, denoted by h.

More formally, our goal is, given a training set, to learn a function $h : \mathcal{X} \mapsto \mathcal{Y}$ so that $h(x)$ is a "good" predictor for the corresponding value of y. For historical reasons, this function h is called a hypothesis. Seen pictorially, the process is therefore like this:

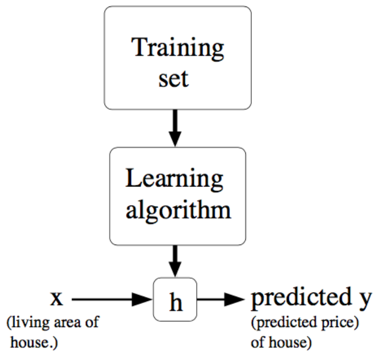


Figure 6. Representation

When the target variable that we're trying to predict is continuous, such as in our housing example, we call the learning problem a *regression problem.* When y can take on only a small number of discrete values (such as if, given the living area, we wanted to predict if a dwelling is a house or an apartment, say), we call it a *classification problem.*

Now, how do we represent h, using linear regression? For the moment, our hypothesis function will be represented by:

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

Which simply says that y is a *linear* function of x.

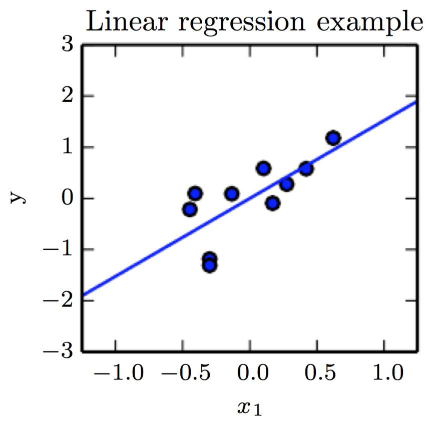


Figure 7. Linear Regression

2.2. Cost Function.

Next, we need to figure out how to solve for the parameters θ_0, θ_1. This will be done using a *Cost Function.*

Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for each training example (x, y). This naturally leads us to the following problem statement:

Find the values of θ_0, θ_1 that causes $h_\theta(x^{(i)})$ to be minimized.

Or, as a minimization problem:

Minimization Problem: Minimize over θ_0, θ_1:

$$\begin{aligned} &\min_{\theta_0,\theta_1} : \Big(h_\theta(x) - y\Big)^2 \\ &= \frac{1}{2m}\sum_{i=1}^{m}\Big(h_\theta(x^{(i)} - y^{(i)}\Big)^2 \end{aligned}$$

where,

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

Which minimizes the "Mean squared error." More formally, this leads to the standard cost function description:

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

which will also be written as:

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

In which we:

$$\underset{\theta_0,\theta_1}{\text{minimize}} : J\Big(h_\theta(x) - y\Big)$$

Here, we effectively measure the accuracy of our hypothesis function. This takes an average difference between the result of the hypothesis with inputs x and actual output y:

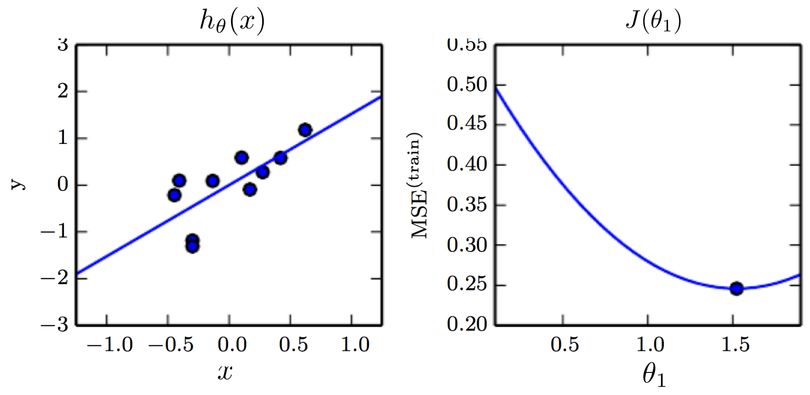
To break it apart, we have:

- $1/2\ \bar{x}$, where $\bar{x}$ is the mean of the squares of $\big(h_\theta(x_i) - y_i\big)$, or the difference between the predicted value and the actual value.
- This function is otherwise called the *"Squared error function"*, or *"Mean squared error."* The mean is halved $(1/2m)$ as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the $1/2$ term.

Now we are able to concretely measure the accuracy of our predictor function against the correct results we have so that we can predict new results we don't have.

Example 2.2 : *Example*

Now, if we track the *hypothesis* function, h, and the cost function, J, we have something like this:



This is very similar in 2D;

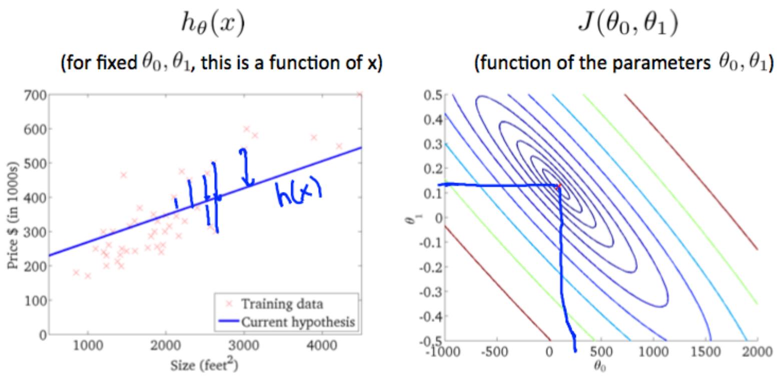


Figure 8. Example

2.3. Gradient Descent.

So we have our hypothesis function and we have a way of measuring how well it fits into the data. Now we need to estimate the parameters in the hypothesis function. That's where gradient descent comes in.

Imagine that we graph our hypothesis function based on its fields θ_0 and θ_1 (actually we are graphing the cost function as a function of the parameter estimates). We are not graphing x and y itself, but the parameter range of our hypothesis function and the cost resulting from selecting a particular set of parameters.

We put θ_0 on the x axis and θ_1 on the y axis, with the cost function on the vertical z axis. The points on our graph will be the result of the cost function using our hypothesis with those specific theta parameters. The graph below depicts such a setup.

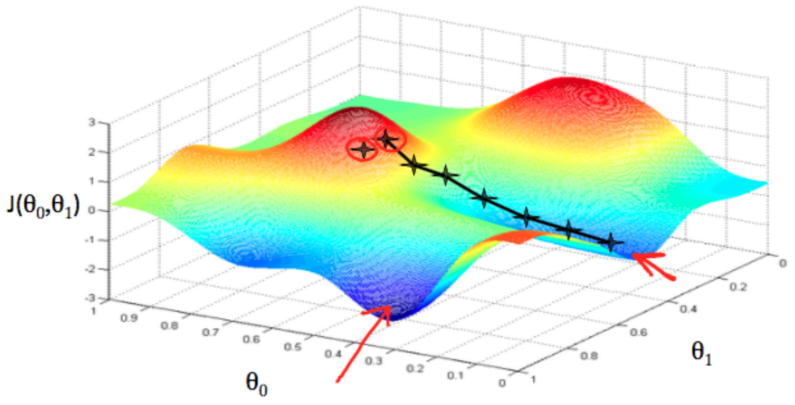


Figure 9. Gradient Descent

The way we do this is by taking the derivative (the tangential line to a function) of our cost function. The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down the cost function in the direction with the steepest descent. The size of each step is determined by the parameter α, which is called the learning rate.

Definition 4. ***Gradient descent*** is a first order iterative optimization algorithm for finding a function's minimum value. To find a local minimum of a function, gradient descent iteratively take steps in the direction of the gradient at the current point.

This method is used in a variety of optimization schemes.

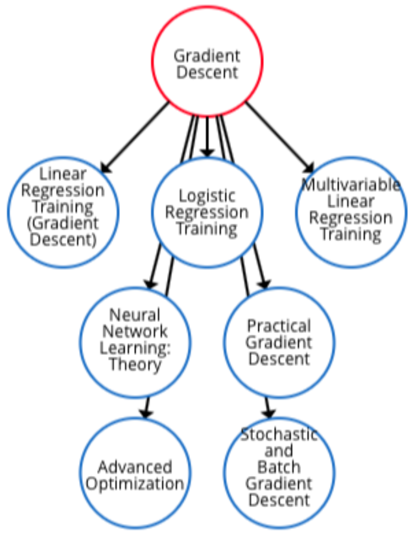

Gradient Descent
Linear Regression Training (Gradient Descent)
Logistic Regression Training
Multivariable Linear Regression Training
Neural Network Learning: Theory
Practical Gradient Descent
Advanced Optimization
Stochastic and Batch Gradient Descent

Figure 10. Gradient Descent

Assume we need to minimize some function:

$$\min_{\theta_0,\theta_1} J\big(\theta_0,\theta_1\big)$$

The gradient descent algorithm is:

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

in which α is defined as the *learning rate*, and $j = 0, 1$ represents the feature index number. More formally, this method may be implemented by the following algorithm:

Algorithm 1: Gradient Descent

```
1 $i = 1$
2 while $err < tol$ do
3     $tmp0 := \theta_0 - \alpha\frac{\partial}{\partial\theta_0}J\big(\theta_0,\theta_1\big)$
4     $tmp1 := \theta_1 - \alpha\frac{\partial}{\partial\theta_1}J\big(\theta_0,\theta_1\big)$
5     $\theta_{i-1} := tmp0$
6     $\theta_i := tmp1$
7 end
```

Here, θ_0 and θ_1 must be updated simultaneously.

Now, in order to apply the *Gradient Descent* to the linear regression model:

2.3.1. ***Gradient Descent for Linear Regression.***
Our linear regression model is described by:

$$\begin{aligned} h_\theta(x) &= \theta_0 + \theta_1 x \\ J(\theta_0, \theta_1) &= \frac{1}{2m}\frac{1}{2m}\sum_{i=1}^{m}\Big(h_\theta(x^{(i)} - y^{(i)}\Big)^2 \end{aligned}$$

in which we seek to minimize J as follows:

$$\min_{\theta_0,\theta_1} J(\theta_0, \theta_1) \qquad \theta_1 \in \mathbb{R}$$

Computing the derivative term:

$$\begin{aligned} \frac{\partial}{\partial\theta_j} J(\theta_0, \theta_1) &= \frac{\partial}{\partial\theta_j} \cdot \frac{1}{2m}\sum_{i=1}^{m}\Big(h_\theta(x^{(i)} - y^{(i)}\Big)^2 \\ &= \frac{\partial}{\partial\theta_j} \cdot \frac{1}{2m}\sum_{i=1}^{m}\Big(\theta_0 + \theta_1 x^{(x)} - y^{(i)}\Big)^2 \end{aligned}$$

In which each θ term is:

$$\begin{aligned} j = 0 : \frac{\partial}{\partial\theta_0} J(\theta_0, \theta_1) &= \frac{\partial}{\partial\theta_0}\Big(\theta_0 + \theta_1 x - y\Big)^2 \\ &= 2\Big(\theta_0 + \theta_1 x - y\Big) \cdot 1 \\ &= 2\Big(h_\theta(x^{(i)}) - y^{(i)}\Big) \end{aligned}$$

$$\begin{aligned} j = 1 : \frac{\partial}{\partial\theta_1} J(\theta_0, \theta_1) &= \frac{\partial}{\partial\theta_0}\Big(\theta_0 + \theta_1 x - y\Big)^2 \\ &= 2\Big(h_\theta(x^{(i)}) - y^{(i)}\Big) \cdot x^{(i)} \end{aligned}$$

Then, our gradient descent algorithm becomes:

Repeat until convergence: {

$$\begin{aligned} j = 0 : \frac{\partial}{\partial\theta_0} J(\theta_0, \theta_1) &= \frac{1}{m}\sum_{i=1}^{m}\Big(h_\theta(x^{(i)} - y^{(i)}\Big) \\ j = 1 : \frac{\partial}{\partial\theta_1} J(\theta_0, \theta_1) &= \frac{1}{m}\sum_{i=1}^{m}\Big(h_\theta(x^{(i)} - y^{(i)}\Big) \cdot x^{(i)} \end{aligned}$$

}

Now, since the linear regression model is always convex, this will always converge. This algorithm is sometimes called 'Batch' gradient descent. This refers to the fact that we are using the entire training sets.

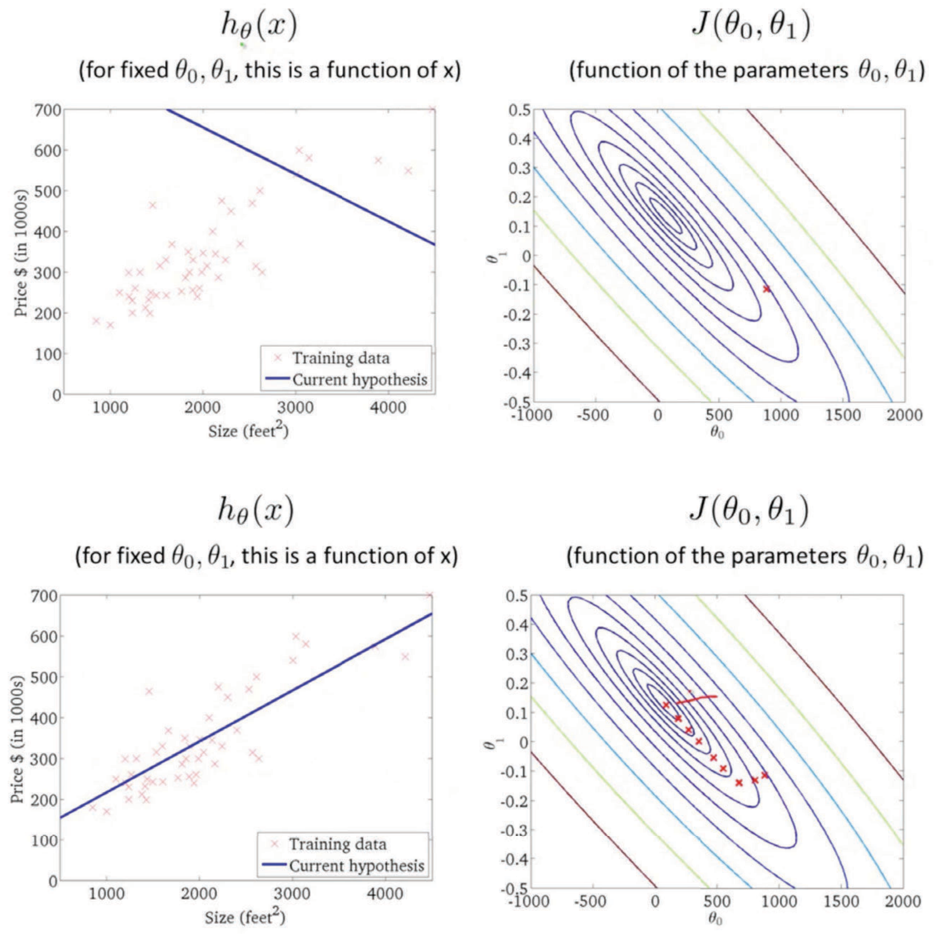


Figure 11. Gradient Descent and linear regression example