Chapter 2

Univariate Linear Regression

2.1 Model Representation

In the example of Housing Prices in Portland, OR, we have the following data: Here are

Example (m)	**Size in sqft** (x)	**Price in \$1000** (y)
1	2104	460
2	1416	232
3	1534	315
4	852	178
...	...	...

Table 2.1: Training set of housing prices (Portland, OR)

some notations that use nomenclature in machine learning and see Figure 2.1:

- m = Number of training examples
- x = "input" variable, or "features"
- y = "output" variable, or "target" variable, or true value

In our housing price example, it is **size x –> model $f(x)$ –> estimated price $\hat{y}$**

How to represent model hypothesis function f?

Formula 2.1 (Univariate Linear Regression)**.** *Model function* $f_{w,b}(x) = wx + b$

where w and b are parameters (or, coefficients, weights). We use superscript $^{(i)}$ to denote the i-th example in the training sample (e.g. $x^{(3)} = 1534$). Therefore, we further has:

$$\widehat{y}^{(i)} = f_{w,b}(x^{(i)}) = wx^{(i)} + b$$

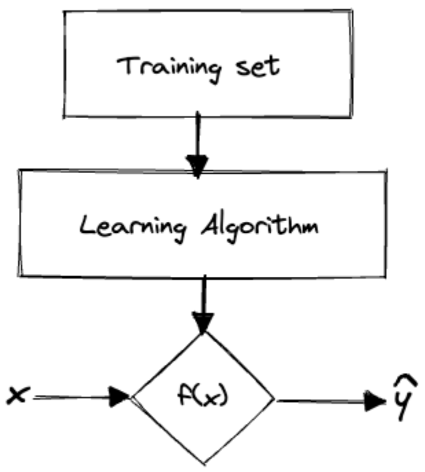

Figure 2.1: Linear Regression Model. **Training Set** include **feature variable x** with labeled "right answer" **target y**. $f(x)$, i.e. **the model**, denotes hypothesis function. $\hat{y}$ denotes **predicted value of y**, output of the model, or estimated y.

Then we can derive the cost function.

2.2 Cost Function: Squared Error

Training Set	x (features), y (targets)
Model	$f_{w,b}(x) = wx + b = \hat{y}$
Parameters	w, b (coefficients/weights)

Table 2.2: Univariate Linear Regression Model

Cost function is the accumulative error of predicted y (i.e. $\hat{y}$) compared with true value target y. We use $J(w, b)$ to indicate cost function. In univariate linear regression, we usually use the **squared error cost function**:

Formula 2.2 (Squared Error Cost Function). *Goal:* $\min_{w,b} J(w, b)$*, where:*

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y^{(i)}} - y^{(i)})^2 = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})^2$$

similarly, m is the number of training examples and i is the i-th example. Here, we need to find w and b such that $J(w, b)$ is the smallest (i.e. $\hat{y}$ is closest to its targeting value y).

For cost function visualization, if $f(x)$ is a linear function, $J(w, b)$ is a convex parabola. And in 3D, we can think $J(w, b)$ as a hammock; where in 2D, $J(w, b)$ represents the contour map.

Next question is: How to find such J_{min}? We will need Gradient Descent Algorithm.

2.3 Gradient Descent

2.3.1 Implementing Gradient Descent

We can think of the cost function $J(w, b)$ as a hill, and in order to find the minimum of $J(w, b)$, we just need to get downhill with baby steps and we can reach the bottom gradually. But, what direction should I take downhill? –> Steepest descent for each baby step (the idea of **greedy algorithm**!), therefore we need the property of gradient descent from calculus. An interesting property of gradient descent is that: **initial guess matters**! Start at different places will end up at different local minima.

Algorithm 2 Gradient Descent Algorithm

```
Initial guess $w_0$, $b_0$
for i in range($m$) do
    keep changing $w, b$ to reduce $J(w, b)$
end for
Until settle at or near a minimum
if $J(w, b)$ decrease by less than $\epsilon$ then
    Reach min $J(w, b)$.
end if
```

While writing code, we will need to update w and b in each iteration of training examples (i-th example, total of m examples). We will repeat the following two updates until convergence.

$$w \leftarrow w - \alpha \frac{\partial}{\partial w} J(w, b) \tag{2.1}$$

$$b \leftarrow b - \alpha \frac{\partial}{\partial b} J(w, b) \tag{2.2}$$

where α denotes the **learning rate** of each baby step. Another important detail is to simultaneously update w and b at the same time:

$$\texttt{tmp\_w} = w - \alpha \frac{\partial}{\partial w} J(w, b) \tag{2.3}$$
$$\texttt{tem\_b} = b - \alpha \frac{\partial}{\partial b} J(w, b) \tag{2.4}$$
$$w \leftarrow \texttt{tmp\_w} \tag{2.5}$$
$$b \leftarrow \texttt{tmp\_b} \tag{2.6}$$

It's important to note that we **cannot** interchange the above second and third line, because we will need to use the old value of w and b to calculate the partial derivative of the cost function $J(w, b)$ for each update.

2.3.2 Learning Rate α

We need a good balance for choosing the right learning rate α. If α too small, it will be super slow to get to the minimum. But if α is too large, we will overshoot and the algorithm will diverge. Moreover, given a fixed learning rate, the gradient descent method can reach local minimum with different sizes of steps because when near local minimum, the partial derivative becomes smaller, therefore steps become smaller as well even with fixed α.

2.4 Gradient Descent for Linear Regression

How do we put linear regression model (i.e. $f_{w,b}(x) = wx + b$), cost function (i.e. $J(w, b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^i) - y^i)^2$), and gradient descent (i.e. $w = w - \alpha\frac{\partial J}{\partial w}$, $b = b - \alpha\frac{\partial J}{\partial b}$) together?

Easy! Just plug linear model $f_{w,b}(x)$ into each example's $f_{w,b}(x^i)$ in the cost function $J(w, b)$, and plug the example's cost function into the gradient descent algorithm, so that each step the algorithm can accumulate all examples' information and update accordingly. The gradient descent for the linear regression model is thus as follows (need to use **calculus chain rule**):

$$w \;\leftarrow\; w - \alpha \cdot \frac{1}{2m} \sum_{i=1}^{m} (wx^{(i)} + b - y^{(i)}) \cdot x^{(i)} \tag{2.7}$$
$$b \;\leftarrow\; b - \alpha \cdot \frac{1}{2m} \sum_{i=1}^{m} (wx^{(i)} + b - y^{(i)}) \tag{2.8}$$

Remember, the above gradient descent formulas are for linear regression, they can be different when we have different cost function $J(w, b)$ as well as model function $f_{w,b}$ (for example logistic regression).

As we can see here, the gradient descent will update all examples' error in one step, therefore, the nomenclature is "**Batch Gradient Descent**", meaning that each step of gradient descent uses all the training examples.

Another important attribute for linear regression with squared error cost function is that the cost function is hammock shaped, i.e."**convex**", therefore after we correctly implemented gradient descent, we will likely arrive at the global minimum of the cost function.