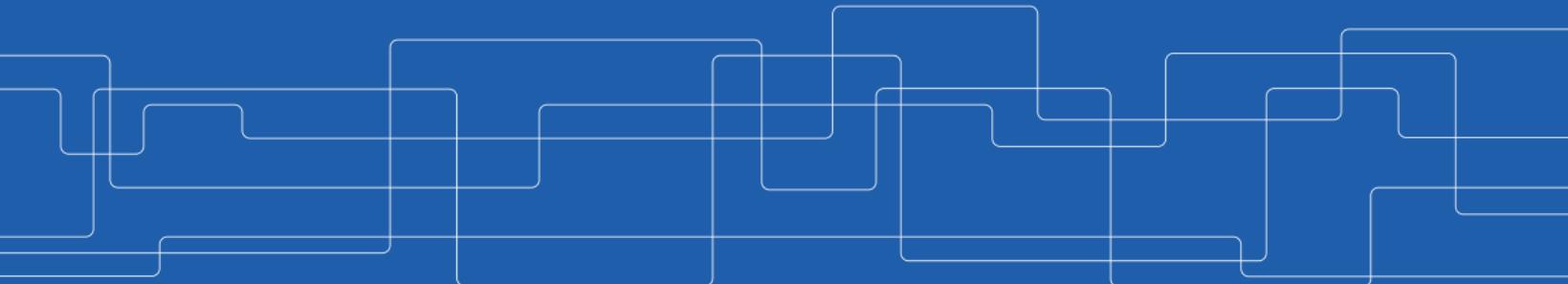




Machine Learning - Regressions

Amir H. Payberah
payberah@kth.se
2021-11-09



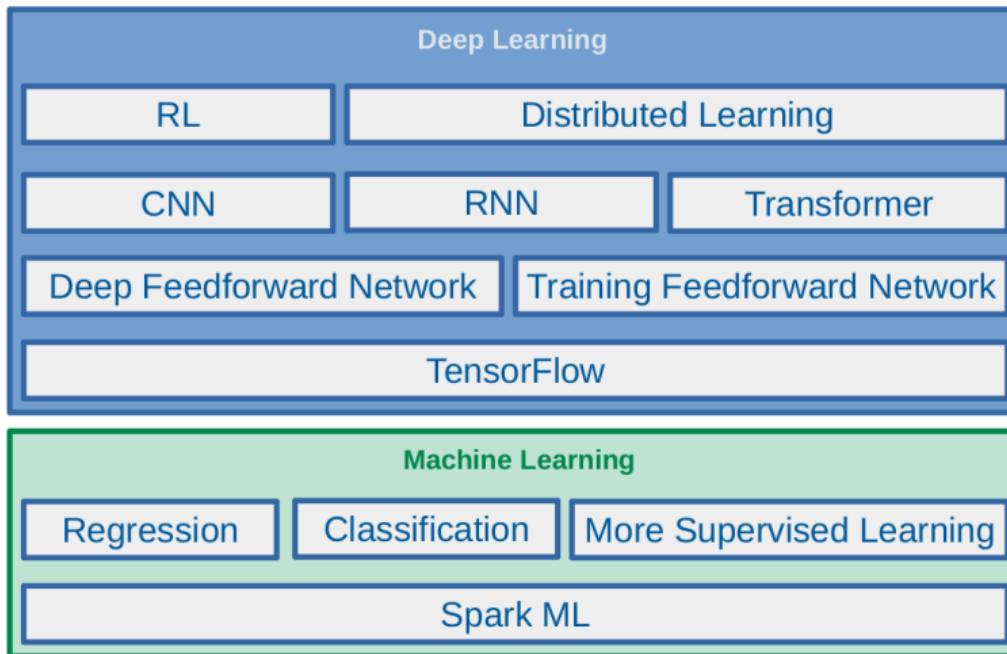


The Course Web Page

<https://id2223kth.github.io>
<https://tinyurl.com/6s5jy46a>

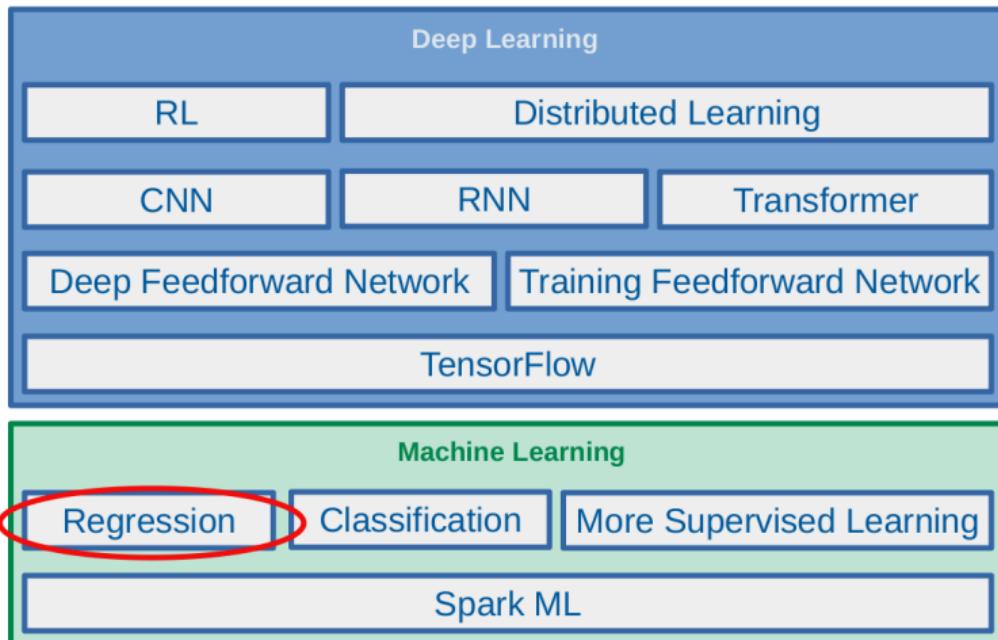


Where Are We?





Where Are We?





Let's Start with an Example



THIS IS MY DREAM HOUSE ALRIGHT, EXCEPT
IN MY DREAM IT WAS ABOUT HALF THIS PRICE.



The Housing Price Example (1/3)

- ▶ Given the dataset of m houses.

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
:	:	:



The Housing Price Example (1/3)

- Given the dataset of m houses.

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
:	:	:

- Predict the prices of other houses, as a function of the size of living area and number of bedrooms?



The Housing Price Example (2/3)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
:	:	:
:	:	:



The Housing Price Example (2/3)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
:	:	:

$$x^{(1)} = \begin{bmatrix} 2104 \\ 3 \end{bmatrix} \quad y^{(1)} = 400 \quad x^{(2)} = \begin{bmatrix} 1600 \\ 3 \end{bmatrix} \quad y^{(2)} = 330 \quad x^{(3)} = \begin{bmatrix} 2400 \\ 3 \end{bmatrix} \quad y^{(3)} = 369$$



The Housing Price Example (2/3)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
:	:	:

$$x^{(1)} = \begin{bmatrix} 2104 \\ 3 \end{bmatrix} \quad y^{(1)} = 400 \quad x^{(2)} = \begin{bmatrix} 1600 \\ 3 \end{bmatrix} \quad y^{(2)} = 330 \quad x^{(3)} = \begin{bmatrix} 2400 \\ 3 \end{bmatrix} \quad y^{(3)} = 369$$

$$X = \begin{bmatrix} x^{(1)\top} \\ x^{(2)\top} \\ x^{(3)\top} \\ \vdots \end{bmatrix} = \begin{bmatrix} 2104 & 3 \\ 1600 & 3 \\ 2400 & 3 \\ \vdots & \vdots \end{bmatrix} \quad y = \begin{bmatrix} 400 \\ 330 \\ 369 \\ \vdots \end{bmatrix}$$

The Housing Price Example (2/3)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
:	:	:

$$x^{(1)} = \begin{bmatrix} 2104 \\ 3 \end{bmatrix} \quad y^{(1)} = 400 \quad x^{(2)} = \begin{bmatrix} 1600 \\ 3 \end{bmatrix} \quad y^{(2)} = 330 \quad x^{(3)} = \begin{bmatrix} 2400 \\ 3 \end{bmatrix} \quad y^{(3)} = 369$$

$$X = \begin{bmatrix} x^{(1)\top} \\ x^{(2)\top} \\ x^{(3)\top} \\ \vdots \end{bmatrix} = \begin{bmatrix} 2104 & 3 \\ 1600 & 3 \\ 2400 & 3 \\ \vdots & \vdots \end{bmatrix} \quad y = \begin{bmatrix} 400 \\ 330 \\ 369 \\ \vdots \end{bmatrix}$$

- $x^{(i)} \in \mathbb{R}^2$: $x_1^{(i)}$ is the living area, and $x_2^{(i)}$ is the number of bedrooms of the i th house in the training set.

The Housing Price Example (3/3)

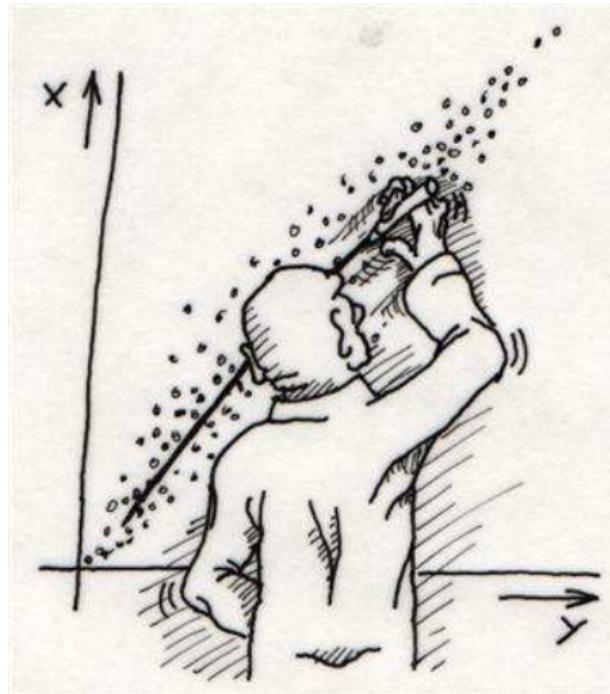
Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
:	:	:

- ▶ Predict the prices of other houses \hat{y} as a function of the size of their living areas x_1 , and number of bedrooms x_2 , i.e., $\hat{y} = f(x_1, x_2)$
- ▶ E.g., what is \hat{y} , if $x_1 = 4000$ and $x_2 = 4$?

The Housing Price Example (3/3)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
:	:	:

- ▶ Predict the prices of other houses \hat{y} as a function of the size of their living areas x_1 , and number of bedrooms x_2 , i.e., $\hat{y} = f(x_1, x_2)$
- ▶ E.g., what is \hat{y} , if $x_1 = 4000$ and $x_2 = 4$?
- ▶ As an initial choice: $\hat{y} = f_w(x) = w_1x_1 + w_2x_2$



[<http://www.vias.org/science.cartoons/regression.html>]



Linear Regression



Linear Regression (1/2)

- ▶ Our goal: to build a system that takes input $x \in \mathbb{R}^n$ and predicts output $\hat{y} \in \mathbb{R}$.



Linear Regression (1/2)

- ▶ Our goal: to build a system that takes input $x \in \mathbb{R}^n$ and predicts output $\hat{y} \in \mathbb{R}$.
- ▶ In linear regression, the output \hat{y} is a linear function of the input x .

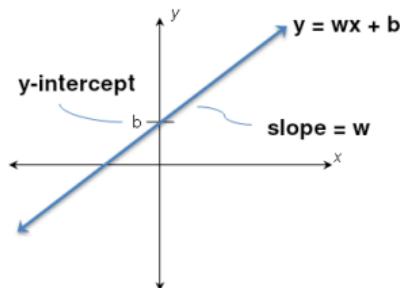
$$\begin{aligned}\hat{y} &= f_w(x) = w_1x_1 + w_2x_2 + \cdots + w_nx_n \\ \hat{y} &= w^T x\end{aligned}$$

- \hat{y} : the predicted value
- n : the number of features
- x_i : the i th feature value
- w_j : the j th model parameter ($w \in \mathbb{R}^n$)

Linear Regression (2/2)

- ▶ Linear regression often has one additional parameter, called **intercept b**:

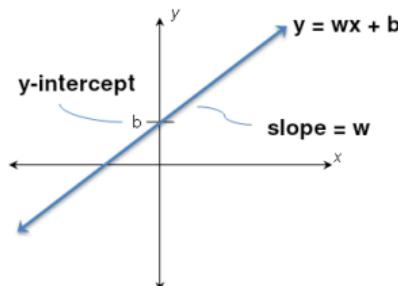
$$\hat{y} = w^T x + b$$



Linear Regression (2/2)

- ▶ Linear regression often has one additional parameter, called **intercept b**:

$$\hat{y} = \mathbf{w}^T \mathbf{x} + b$$



- ▶ Instead of adding the bias parameter **b**, we can augment **x** with an **extra entry** that is **always set to 1**.

$$\hat{y} = f_w(\mathbf{x}) = w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n, \text{ where } x_0 = 1$$



Linear Regression - Model Parameters

- ▶ Parameters $w \in \mathbb{R}^n$ are values that control the behavior of the model.



Linear Regression - Model Parameters

- ▶ Parameters $w \in \mathbb{R}^n$ are values that control the behavior of the model.
- ▶ w are a set of weights that determine how each feature affects the prediction.



Linear Regression - Model Parameters

- ▶ Parameters $w \in \mathbb{R}^n$ are values that control the behavior of the model.
- ▶ w are a set of weights that determine how each feature affects the prediction.
 - $w_i > 0$: increasing the value of the feature x_i , increases the value of our prediction \hat{y} .



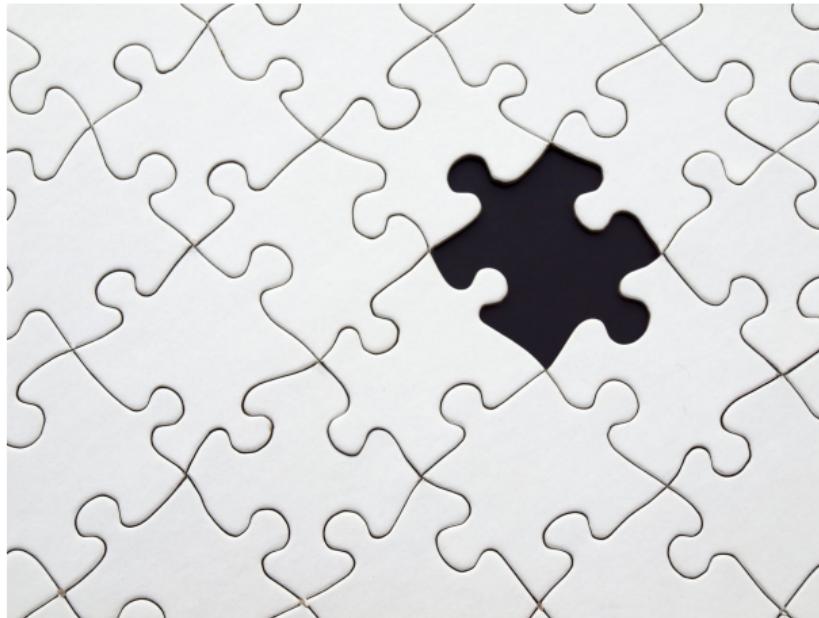
Linear Regression - Model Parameters

- ▶ Parameters $w \in \mathbb{R}^n$ are values that control the behavior of the model.
- ▶ w are a set of weights that determine how each feature affects the prediction.
 - $w_i > 0$: increasing the value of the feature x_i , increases the value of our prediction \hat{y} .
 - $w_i < 0$: increasing the value of the feature x_i , decreases the value of our prediction \hat{y} .



Linear Regression - Model Parameters

- ▶ Parameters $w \in \mathbb{R}^n$ are values that control the behavior of the model.
- ▶ w are a set of weights that determine how each feature affects the prediction.
 - $w_i > 0$: increasing the value of the feature x_i , increases the value of our prediction \hat{y} .
 - $w_i < 0$: increasing the value of the feature x_i , decreases the value of our prediction \hat{y} .
 - $w_i = 0$: the value of the feature x_i , has no effect on the prediction \hat{y} .

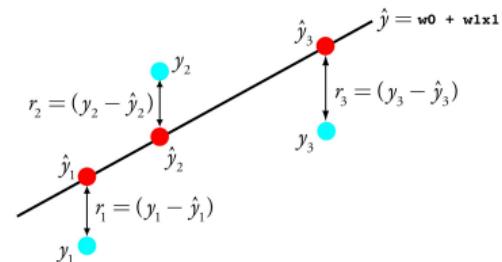
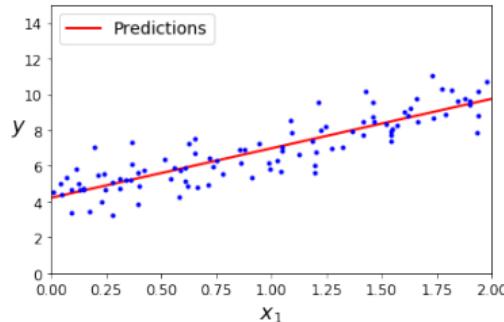


$$\hat{y} = f_w(x) = w_0x_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n$$



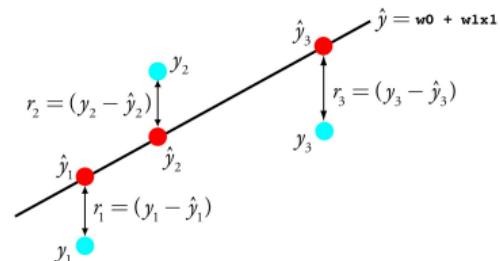
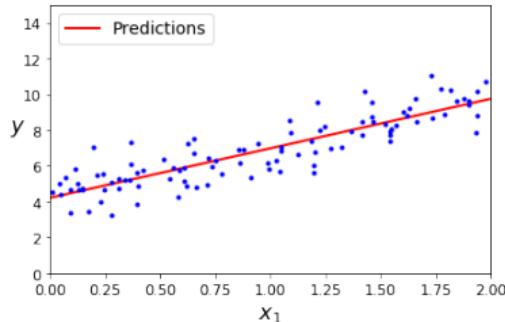
How to Learn Model Parameters w ?

Linear Regression - Cost Function (1/2)



- One reasonable model should make \hat{y} close to y , at least for the training dataset.

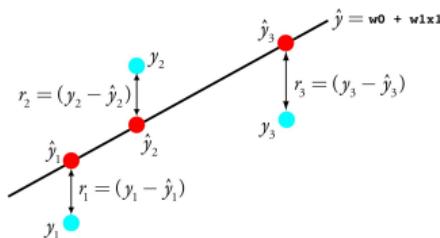
Linear Regression - Cost Function (1/2)



- One reasonable model should make \hat{y} close to y , at least for the training dataset.
- Residual: the difference between the dependent variable y and the predicted value \hat{y} .

$$r^{(i)} = y^{(i)} - \hat{y}^{(i)}$$

Linear Regression - Cost Function (2/2)

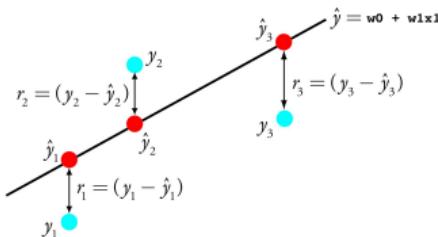


► Cost function $J(w)$

- For each value of the w , it measures how close the $\hat{y}^{(i)}$ is to the corresponding $y^{(i)}$.
- We can define $J(w)$ as the mean squared error (MSE):

$$J(w) = \text{MSE}(w) = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$

Linear Regression - Cost Function (2/2)



► Cost function $J(w)$

- For each value of the w , it measures how close the $\hat{y}^{(i)}$ is to the corresponding $y^{(i)}$.
- We can define $J(w)$ as the mean squared error (MSE):

$$\begin{aligned} J(w) &= \text{MSE}(w) = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2 \\ &= E[(\hat{y} - y)^2] = \frac{1}{m} \|\hat{y} - y\|_2^2 \end{aligned}$$



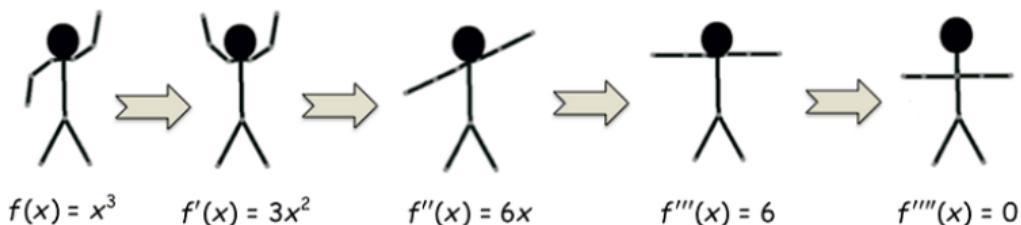
How to Learn Model Parameters?

- ▶ We want to choose w so as to minimize $J(w)$.
- ▶ Two approaches to find w :
 - Normal equation
 - Gradient descent



Normal Equation

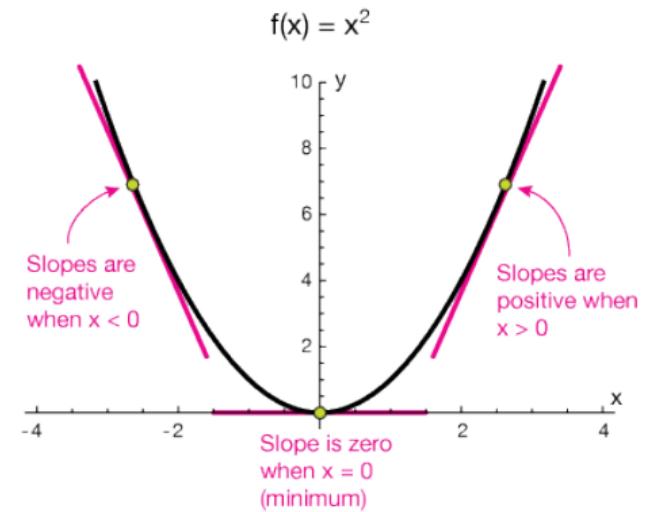
Derivatives and Gradient (1/4)



[<https://mathequality.wordpress.com/2012/09/26/derivative-dance-gangnam-style/>]

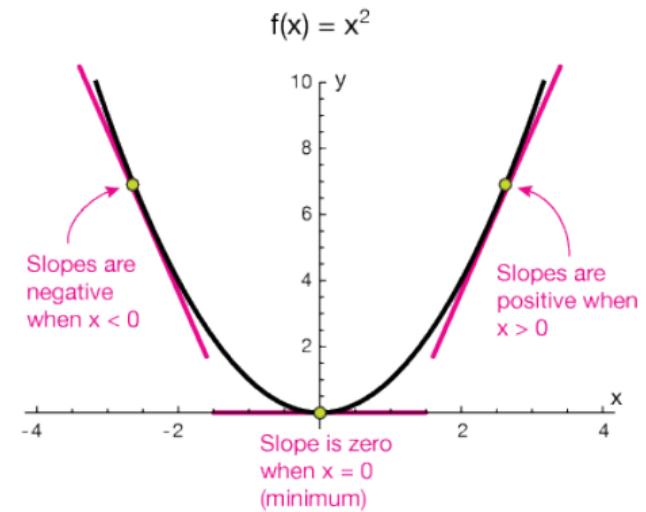
Derivatives and Gradient (2/4)

- The **first derivative** of $f(x)$, shown as $f'(x)$, shows the **slope** of the **tangent line** to the function at the point x .



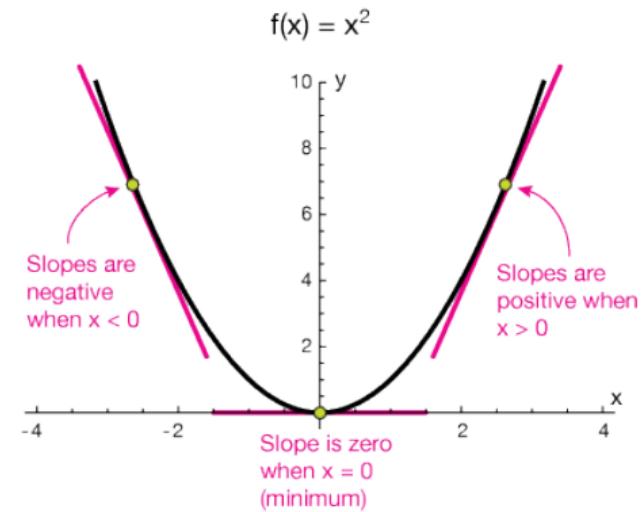
Derivatives and Gradient (2/4)

- ▶ The **first derivative** of $f(x)$, shown as $f'(x)$, shows the **slope** of the **tangent line** to the function at the point x .
- ▶ $f(x) = x^2 \Rightarrow f'(x) = 2x$



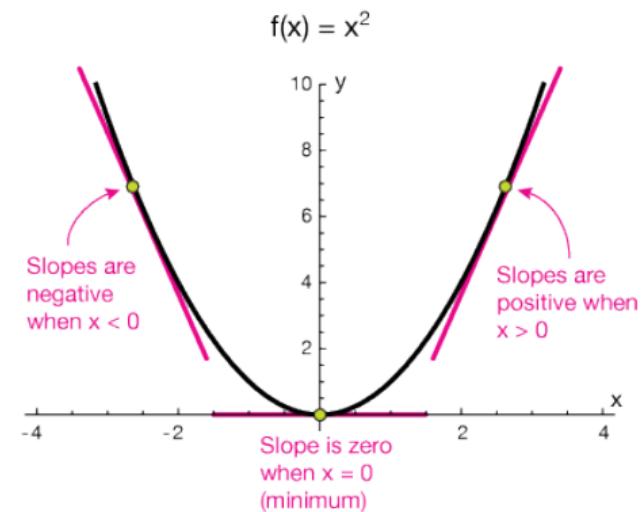
Derivatives and Gradient (2/4)

- ▶ The **first derivative** of $f(x)$, shown as $f'(x)$, shows the **slope** of the **tangent line** to the function at the point x .
- ▶ $f(x) = x^2 \Rightarrow f'(x) = 2x$
- ▶ If $f(x)$ is increasing, then $f'(x) > 0$



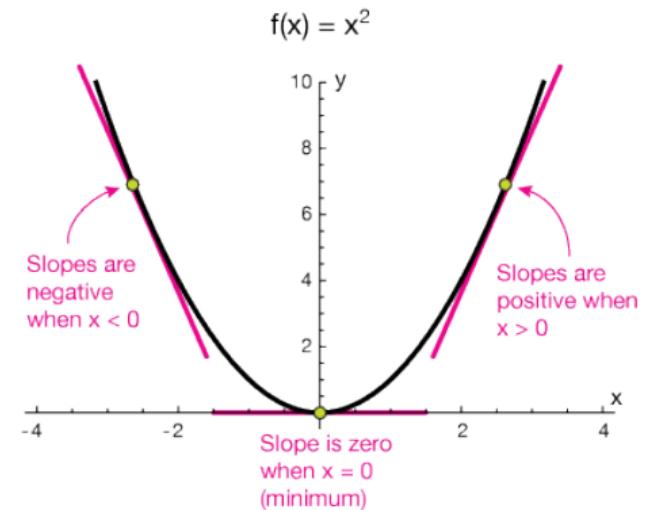
Derivatives and Gradient (2/4)

- ▶ The **first derivative** of $f(x)$, shown as $f'(x)$, shows the **slope** of the **tangent line** to the function at the point x .
- ▶ $f(x) = x^2 \Rightarrow f'(x) = 2x$
- ▶ If $f(x)$ is increasing, then $f'(x) > 0$
- ▶ If $f(x)$ is decreasing, then $f'(x) < 0$



Derivatives and Gradient (2/4)

- ▶ The **first derivative** of $f(x)$, shown as $f'(x)$, shows the **slope** of the **tangent line** to the function at the point x .
- ▶ $f(x) = x^2 \Rightarrow f'(x) = 2x$
- ▶ If $f(x)$ is increasing, then $f'(x) > 0$
- ▶ If $f(x)$ is decreasing, then $f'(x) < 0$
- ▶ If $f(x)$ is at local minimum/maximum, then $f'(x) = 0$





Derivatives and Gradient (3/4)

- ▶ What if a function has multiple arguments, e.g., $f(x_1, x_2, \dots, x_n)$
- ▶ **Partial derivatives:** the derivative with respect to a particular argument.
 - $\frac{\partial f}{\partial x_1}$, the derivative with respect to x_1
 - $\frac{\partial f}{\partial x_2}$, the derivative with respect to x_2



Derivatives and Gradient (3/4)

- ▶ What if a function has multiple arguments, e.g., $f(x_1, x_2, \dots, x_n)$
- ▶ **Partial derivatives:** the derivative with respect to a particular argument.
 - $\frac{\partial f}{\partial x_1}$, the derivative with respect to x_1
 - $\frac{\partial f}{\partial x_2}$, the derivative with respect to x_2
- ▶ $\frac{\partial f}{\partial x_i}$: shows how much the function f will change, if we change x_i .



Derivatives and Gradient (3/4)

- ▶ What if a function has multiple arguments, e.g., $f(x_1, x_2, \dots, x_n)$
- ▶ **Partial derivatives:** the derivative with respect to a particular argument.
 - $\frac{\partial f}{\partial x_1}$, the derivative with respect to x_1
 - $\frac{\partial f}{\partial x_2}$, the derivative with respect to x_2
- ▶ $\frac{\partial f}{\partial x_i}$: shows how much the function f will change, if we change x_i .
- ▶ **Gradient:** the vector of all partial derivatives for a function f .

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$



Derivatives and Gradient (4/4)

- ▶ What is the gradient of $f(x_1, x_2, x_3) = x_1 - x_1x_2 + x_3^2$?

Derivatives and Gradient (4/4)

- ▶ What is the gradient of $f(x_1, x_2, x_3) = x_1 - x_1x_2 + x_3^2$?

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial}{\partial x_1}(x_1 - x_1x_2 + x_3^2) \\ \frac{\partial}{\partial x_2}(x_1 - x_1x_2 + x_3^2) \\ \frac{\partial}{\partial x_3}(x_1 - x_1x_2 + x_3^2) \end{bmatrix} = \begin{bmatrix} 1 - x_2 \\ -x_1 \\ 2x_3 \end{bmatrix}$$



Normal Equation (1/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$\hat{y} = w^T x$$

Normal Equation (1/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$\hat{y} = w^T x$$

$$X = \begin{bmatrix} [x_1^{(1)}, x_2^{(1)}, \dots, x_n^{(1)}] \\ [x_1^{(2)}, x_2^{(2)}, \dots, x_n^{(2)}] \\ \vdots \\ [x_1^{(m)}, x_2^{(m)}, \dots, x_n^{(m)}] \end{bmatrix} = \begin{bmatrix} x^{(1)\top} \\ x^{(2)\top} \\ \vdots \\ x^{(m)\top} \end{bmatrix} \quad \hat{y} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(m)} \end{bmatrix}$$

Normal Equation (1/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$\hat{y} = w^T x$$

$$X = \begin{bmatrix} [x_1^{(1)}, x_2^{(1)}, \dots, x_n^{(1)}] \\ [x_1^{(2)}, x_2^{(2)}, \dots, x_n^{(2)}] \\ \vdots \\ [x_1^{(m)}, x_2^{(m)}, \dots, x_n^{(m)}] \end{bmatrix} = \begin{bmatrix} x^{(1)\top} \\ x^{(2)\top} \\ \vdots \\ x^{(m)\top} \end{bmatrix} \quad \hat{y} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(m)} \end{bmatrix}$$

$$\hat{y} = w^T X^\top \text{ or } \hat{y} = Xw$$



Normal Equation (2/2)

- ▶ To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

Normal Equation (2/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$J(w) = \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2, \nabla_w J(w) = 0$$

Normal Equation (2/2)

- ▶ To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$\begin{aligned}J(w) &= \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2, \quad \nabla_w J(w) = 0 \\&\Rightarrow \nabla_w \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2 = 0\end{aligned}$$

Normal Equation (2/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$\begin{aligned}J(w) &= \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2, \quad \nabla_w J(w) = 0 \\&\Rightarrow \nabla_w \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2 = 0 \\&\Rightarrow \nabla_w \frac{1}{m} \|\mathbf{Xw} - \mathbf{y}\|_2^2 = 0\end{aligned}$$

Normal Equation (2/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$\begin{aligned}J(w) &= \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2, \quad \nabla_w J(w) = 0 \\&\Rightarrow \nabla_w \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2 = 0 \\&\Rightarrow \nabla_w \frac{1}{m} \|\mathbf{X}w - \mathbf{y}\|_2^2 = 0 \\&\Rightarrow \nabla_w (\mathbf{X}w - \mathbf{y})^\top (\mathbf{X}w - \mathbf{y}) = 0\end{aligned}$$

Normal Equation (2/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$J(w) = \frac{1}{m} \|\hat{y} - y\|_2^2, \nabla_w J(w) = 0$$

$$\Rightarrow \nabla_w \frac{1}{m} \|\hat{y} - y\|_2^2 = 0$$

$$\Rightarrow \nabla_w \frac{1}{m} \|Xw - y\|_2^2 = 0$$

$$\Rightarrow \nabla_w (Xw - y)^T (Xw - y) = 0$$

$$\Rightarrow \nabla_w (w^T X^T X w - 2w^T X^T y + y^T y) = 0$$

Normal Equation (2/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$J(w) = \frac{1}{m} \|\hat{y} - y\|_2^2, \nabla_w J(w) = 0$$

$$\Rightarrow \nabla_w \frac{1}{m} \|\hat{y} - y\|_2^2 = 0$$

$$\Rightarrow \nabla_w \frac{1}{m} \|Xw - y\|_2^2 = 0$$

$$\Rightarrow \nabla_w (Xw - y)^T (Xw - y) = 0$$

$$\Rightarrow \nabla_w (w^T X^T X w - 2w^T X^T y + y^T y) = 0$$

$$\Rightarrow 2X^T X w - 2X^T y = 0$$

Normal Equation (2/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$J(w) = \frac{1}{m} \|\hat{y} - y\|_2^2, \nabla_w J(w) = 0$$

$$\Rightarrow \nabla_w \frac{1}{m} \|\hat{y} - y\|_2^2 = 0$$

$$\Rightarrow \nabla_w \frac{1}{m} \|Xw - y\|_2^2 = 0$$

$$\Rightarrow \nabla_w (Xw - y)^T (Xw - y) = 0$$

$$\Rightarrow \nabla_w (w^T X^T X w - 2w^T X^T y + y^T y) = 0$$

$$\Rightarrow 2X^T X w - 2X^T y = 0$$

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



Normal Equation - Example (1/7)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540

- ▶ Predict the value of \hat{y} , when $x_1 = 4000$ and $x_2 = 4$.

Normal Equation - Example (1/7)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540

- ▶ Predict the value of \hat{y} , when $x_1 = 4000$ and $x_2 = 4$.
- ▶ We should find w_0 , w_1 , and w_2 in $\hat{y} = w_0 + w_1x_1 + w_2x_2$.
- ▶ $w = (X^T X)^{-1} X^T y$.



Normal Equation - Example (2/7)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540

$$X = \left[\begin{array}{c|ccc} 1 & 2104 & 3 \\ 1 & 1600 & 3 \\ 1 & 2400 & 3 \\ 1 & 1416 & 2 \\ 1 & 3000 & 4 \end{array} \right] \quad y = \begin{bmatrix} 400 \\ 330 \\ 369 \\ 232 \\ 540 \end{bmatrix}$$

Normal Equation - Example (3/7)

$$X^T X = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2104 & 1600 & 2400 & 1416 & 3000 \\ 3 & 3 & 3 & 2 & 4 \end{bmatrix} \begin{bmatrix} 1 & 2104 & 3 \\ 1 & 1600 & 3 \\ 1 & 2400 & 3 \\ 1 & 1416 & 2 \\ 1 & 3000 & 4 \end{bmatrix} = \begin{bmatrix} 5 & 10520 & 23751872 & 33144 \\ 15 & 33144 & 47 \end{bmatrix}$$



Normal Equation - Example (4/7)

$$(X^T X)^{-1} = \begin{bmatrix} 4.90366455e + 00 & 7.48766737e - 04 & -2.09302326e + 00 \\ 7.48766737e - 04 & 2.75281889e - 06 & -2.18023256e - 03 \\ -2.09302326e + 00 & -2.18023256e - 03 & 2.22674419e + 00 \end{bmatrix}$$

Normal Equation - Example (5/7)

$$X^T y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2104 & 1600 & 2400 & 1416 & 3000 \\ 3 & 3 & 3 & 2 & 4 \end{bmatrix} \begin{bmatrix} 400 \\ 330 \\ 369 \\ 232 \\ 540 \end{bmatrix} = \begin{bmatrix} 1871 \\ 4203712 \\ 5921 \end{bmatrix}$$

Normal Equation - Example (6/7)

$$\begin{aligned} w = (X^T X)^{-1} X^T y &= \begin{bmatrix} 4.90366455e + 00 & 7.48766737e - 04 & -2.09302326e + 00 \\ 7.48766737e - 04 & 2.75281889e - 06 & -2.18023256e - 03 \\ -2.09302326e + 00 & -2.18023256e - 03 & 2.22674419e + 00 \end{bmatrix} \begin{bmatrix} 1871 \\ 4203712 \\ 5921 \end{bmatrix} \\ &= \begin{bmatrix} -7.04346018e + 01 \\ 6.38433756e - 02 \\ 1.03436047e + 02 \end{bmatrix} \end{aligned}$$



Normal Equation - Example (7/7)

- ▶ Predict the value of y , when $x_1 = 4000$ and $x_2 = 4$.

$$\hat{y} = -7.04346018e + 01 + 6.38433756e - 02 \times 4000 + 1.03436047e + 02 \times 4 \approx 599$$



Normal Equation in Spark

```
case class house(x1: Long, x2: Long, y: Long)

val trainData = Seq(house(2104, 3, 400), house(1600, 3, 330), house(2400, 3, 369),
                    house(1416, 2, 232), house(3000, 4, 540)).toDF

val testData = Seq(house(4000, 4, 0)).toDF
```



Normal Equation in Spark

```
case class house(x1: Long, x2: Long, y: Long)

val trainData = Seq(house(2104, 3, 400), house(1600, 3, 330), house(2400, 3, 369),
                    house(1416, 2, 232), house(3000, 4, 540)).toDF

val testData = Seq(house(4000, 4, 0)).toDF
```

```
import org.apache.spark.ml.feature.VectorAssembler

val va = new VectorAssembler().setInputCols(Array("x1", "x2")).setOutputCol("features")

val train = va.transform(trainData)
val test = va.transform(testData)
```



Normal Equation in Spark

```
case class house(x1: Long, x2: Long, y: Long)

val trainData = Seq(house(2104, 3, 400), house(1600, 3, 330), house(2400, 3, 369),
                    house(1416, 2, 232), house(3000, 4, 540)).toDF

val testData = Seq(house(4000, 4, 0)).toDF
```

```
import org.apache.spark.ml.feature.VectorAssembler

val va = new VectorAssembler().setInputCols(Array("x1", "x2")).setOutputCol("features")

val train = va.transform(trainData)
val test = va.transform(testData)
```

```
import org.apache.spark.ml.regression.LinearRegression

val lr = new LinearRegression().setFeaturesCol("features").setLabelCol("y").setSolver("normal")
val lrModel = lr.fit(train)
lrModel.transform(test).show
```



Normal Equation - Computational Complexity

- ▶ The computational complexity of inverting $X^T X$ is $O(n^3)$.
 - For an $m \times n$ matrix (where n is the number of features).



Normal Equation - Computational Complexity

- ▶ The computational complexity of inverting $X^T X$ is $O(n^3)$.
 - For an $m \times n$ matrix (where n is the number of features).
- ▶ But, this equation is linear with regards to the number of instances in the training set (it is $O(m)$).
 - It handles large training sets efficiently, provided they can fit in memory.



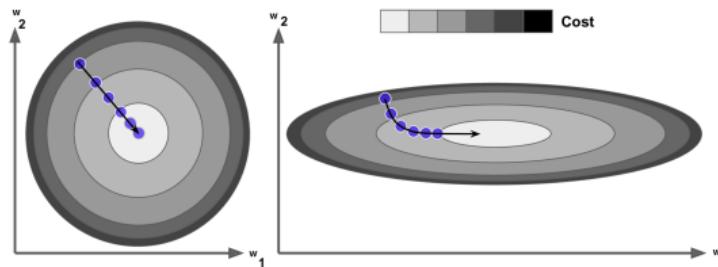
[<https://dailyfintech.com/2017/03/13/now-all-we-need-is-for-blockchain-to-become-technologically-boring>]



Gradient Descent

Gradient Descent (1/2)

- ▶ Gradient descent is a generic optimization algorithm capable of finding optimal solutions to a wide range of problems.
- ▶ The idea: to tweak parameters iteratively in order to minimize a cost function.



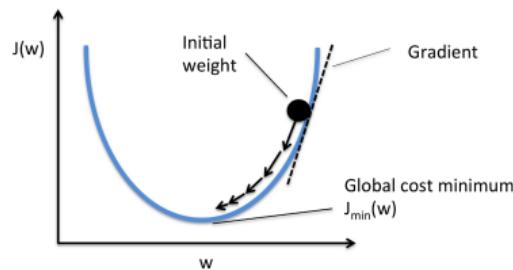
Gradient Descent (2/2)

- ▶ Suppose you are **lost** in the **mountains** in a dense fog.
- ▶ You can only feel the **slope** of the ground below your feet.
- ▶ A strategy to **get to the bottom** of the valley is to **go downhill** in the **direction of the steepest slope**.



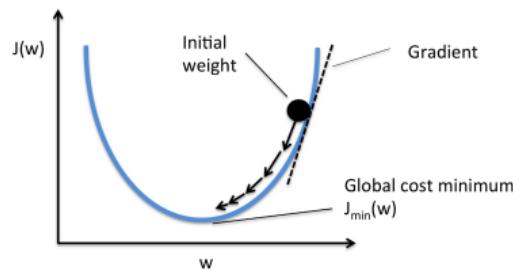
Gradient Descent - Iterative Optimization Algorithm

- ▶ Choose a starting point, e.g., filling w with random values.



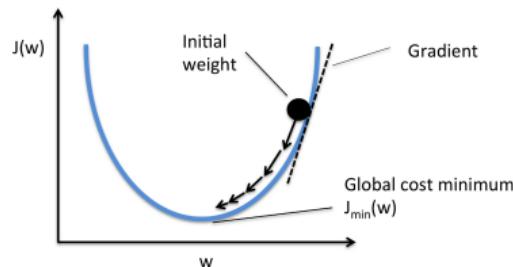
Gradient Descent - Iterative Optimization Algorithm

- ▶ Choose a starting point, e.g., filling w with random values.
- ▶ If the **stopping criterion** is true return the **current solution**, otherwise continue.



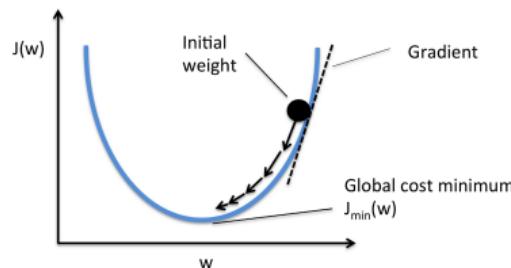
Gradient Descent - Iterative Optimization Algorithm

- ▶ Choose a starting point, e.g., filling w with random values.
- ▶ If the **stopping criterion** is true return the **current solution**, otherwise continue.
- ▶ Find a **descent direction**, a **direction in which the function value decreases** near the current point.



Gradient Descent - Iterative Optimization Algorithm

- ▶ Choose a starting point, e.g., filling w with random values.
- ▶ If the **stopping criterion** is true return the **current solution**, otherwise continue.
- ▶ Find a **descent direction**, a **direction in which the function value decreases** near the current point.
- ▶ Determine the **step size**, the **length of a step** in the given direction.





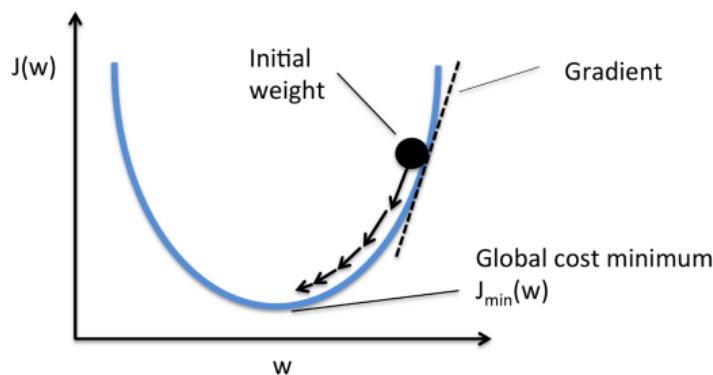
Gradient Descent - Key Points

- ▶ Stopping criterion
- ▶ Descent direction
- ▶ Step size (learning rate)

Gradient Descent - Stopping Criterion

- ▶ The **cost function minimum** property: the **gradient** has to be **zero**.

$$\nabla_w J(w) = 0$$





Gradient Descent - Descent Direction (1/2)

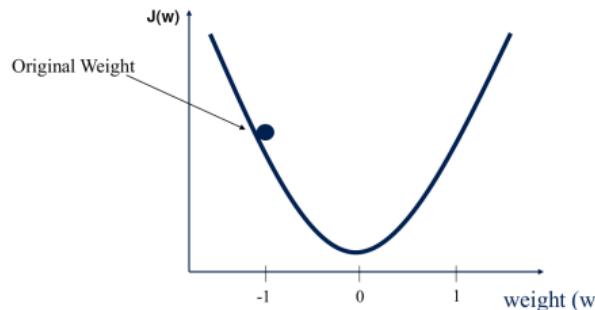
- ▶ Direction in which the **function value decreases** near the current point.
- ▶ Find the **direction of descent (slope)**.

Gradient Descent - Descent Direction (1/2)

- ▶ Direction in which the **function value decreases** near the current point.
- ▶ Find the **direction of descent (slope)**.
- ▶ Example:

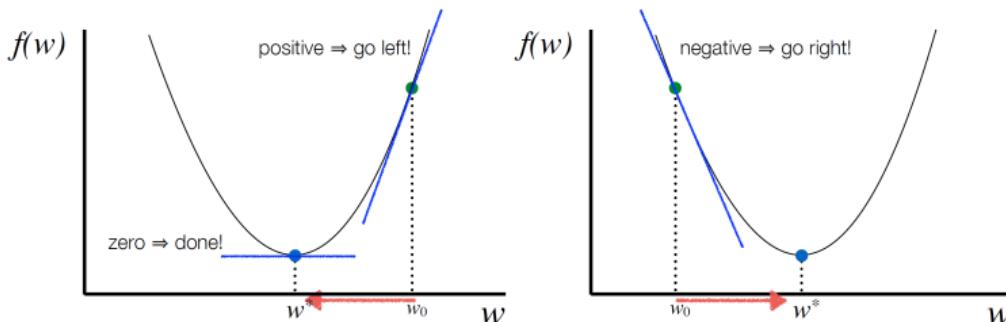
$$J(w) = w^2$$

$$\frac{\partial J(w)}{\partial w} = 2w = -2 \text{ at } w = -1$$



Gradient Descent - Descent Direction (2/2)

- Follow the opposite direction of the slope.



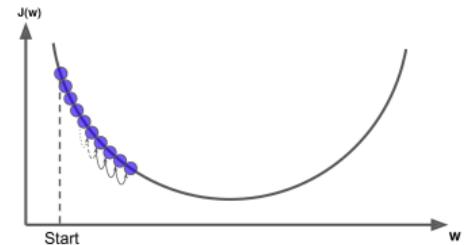


Gradient Descent - Learning Rate

- ▶ **Learning rate**: the length of steps.

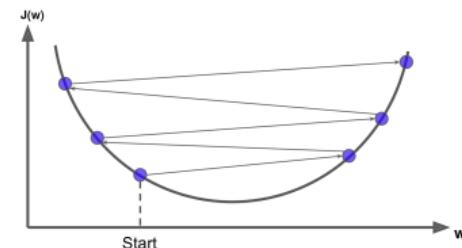
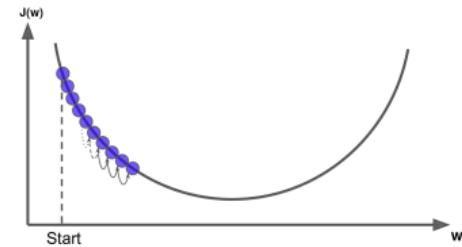
Gradient Descent - Learning Rate

- ▶ **Learning rate**: the length of steps.
- ▶ If it is **too small**: many iterations to converge.



Gradient Descent - Learning Rate

- ▶ **Learning rate**: the length of steps.
- ▶ If it is **too small**: many iterations to converge.
- ▶ If it is **too high**: the algorithm might diverge.



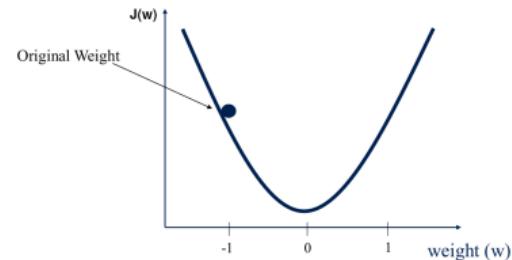


Gradient Descent - How to Learn Model Parameters w?

- **Goal:** find w that minimizes $J(w) = \sum_{i=1}^m (w^T x^{(i)} - y^{(i)})^2$.

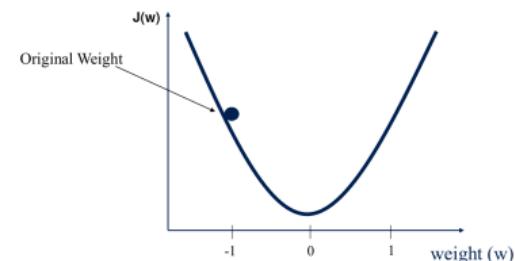
Gradient Descent - How to Learn Model Parameters w ?

- ▶ **Goal:** find w that minimizes $J(w) = \sum_{i=1}^m (w^T x^{(i)} - y^{(i)})^2$.
- ▶ Start at a **random point**, and repeat the following **steps**, until the **stopping criterion** is satisfied:



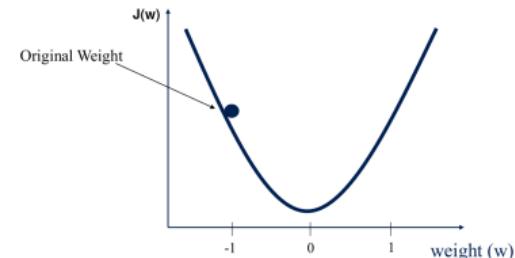
Gradient Descent - How to Learn Model Parameters w ?

- ▶ **Goal:** find w that minimizes $J(w) = \sum_{i=1}^m (w^T x^{(i)} - y^{(i)})^2$.
- ▶ Start at a random point, and repeat the following steps, until the stopping criterion is satisfied:
 1. Determine a descent direction $\frac{\partial J(w)}{\partial w}$



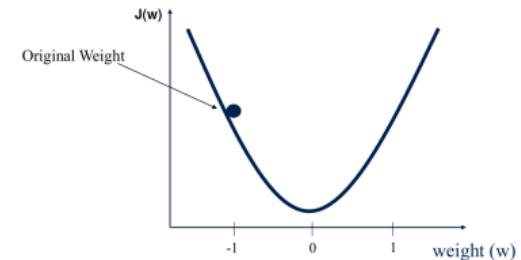
Gradient Descent - How to Learn Model Parameters w ?

- ▶ **Goal:** find w that minimizes $J(w) = \sum_{i=1}^m (w^T x^{(i)} - y^{(i)})^2$.
- ▶ Start at a random point, and repeat the following steps, until the stopping criterion is satisfied:
 1. Determine a descent direction $\frac{\partial J(w)}{\partial w}$
 2. Choose a step size η



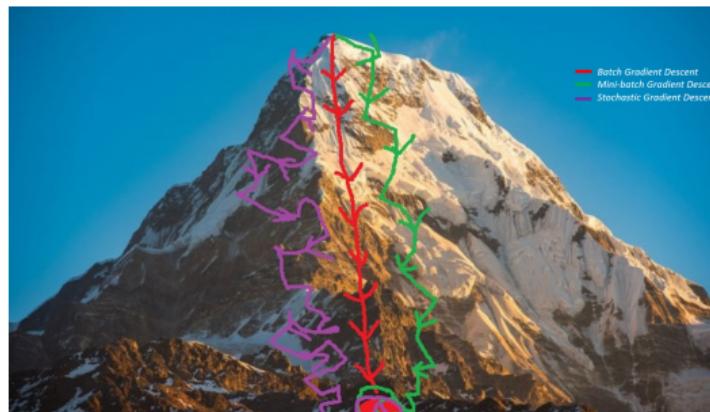
Gradient Descent - How to Learn Model Parameters w ?

- ▶ Goal: find w that minimizes $J(w) = \sum_{i=1}^m (w^T x^{(i)} - y^{(i)})^2$.
- ▶ Start at a random point, and repeat the following steps, until the stopping criterion is satisfied:
 1. Determine a descent direction $\frac{\partial J(w)}{\partial w}$
 2. Choose a step size η
 3. Update the parameters: $w^{(\text{next})} = w - \eta \frac{\partial J(w)}{\partial w}$
(should be done for all parameters simultaneously)



Gradient Descent - Different Algorithms

- ▶ Batch gradient descent
- ▶ Stochastic gradient descent
- ▶ Mini-batch gradient descent



[<https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3>]



Batch Gradient Descent



Batch Gradient Descent (1/2)

- ▶ Repeat the following **steps**, until the **stopping criterion** is satisfied:

1. Determine a **descent direction** $\frac{\partial J(w)}{\partial w}$ for all parameters w .

$$J(w) = \sum_{i=1}^m (w^\top x^{(i)} - y^{(i)})^2$$



Batch Gradient Descent (1/2)

- ▶ Repeat the following **steps**, until the stopping criterion is satisfied:

1. Determine a **descent direction** $\frac{\partial J(w)}{\partial w}$ for all parameters w .

$$J(w) = \sum_{i=1}^m (w^T x^{(i)} - y^{(i)})^2$$

$$\frac{\partial J(w)}{\partial w_j} = \frac{2}{m} \sum_{i=1}^m (w^T x^{(i)} - y^{(i)}) x_j^{(i)}$$
$$\nabla_w J(w) = \begin{bmatrix} \frac{\partial J(w)}{\partial w_0} \\ \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{bmatrix} = \frac{2}{m} X^T (Xw - y)$$

2. Choose a **step size** η



Batch Gradient Descent (1/2)

- ▶ Repeat the following **steps**, until the stopping criterion is satisfied:

1. Determine a **descent direction** $\frac{\partial J(w)}{\partial w}$ for all parameters w .

$$J(w) = \sum_{i=1}^m (w^T x^{(i)} - y^{(i)})^2$$

$$\frac{\partial J(w)}{\partial w_j} = \frac{2}{m} \sum_{i=1}^m (w^T x^{(i)} - y^{(i)}) x_j^{(i)}$$
$$\nabla_w J(w) = \begin{bmatrix} \frac{\partial J(w)}{\partial w_0} \\ \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{bmatrix} = \frac{2}{m} X^T (Xw - y)$$

2. Choose a **step size** η
3. **Update** the parameters: $w^{(\text{next})} = w - \eta \nabla_w J(w)$



Batch Gradient Descent (2/2)

- ▶ The algorithm is called **Batch Gradient Descent**, because at each step, calculations are over the **full training set X** .
- ▶ As a result it is **slow on very large training sets**, i.e., large m .
- ▶ But, it **scales well** with the **number of features n** .



Batch Gradient Descent - Example (1/5)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2$$

$$X = \left[\begin{array}{c|ccc} 1 & 2104 & 3 \\ 1 & 1600 & 3 \\ 1 & 2400 & 3 \\ 1 & 1416 & 2 \\ 1 & 3000 & 4 \end{array} \right] \quad y = \left[\begin{array}{c} 400 \\ 330 \\ 369 \\ 232 \\ 540 \end{array} \right]$$

Batch Gradient Descent - Example (2/5)

$$X = \left[\begin{array}{c|ccc} 1 & 2104 & 3 \\ 1 & 1600 & 3 \\ 1 & 2400 & 3 \\ 1 & 1416 & 2 \\ 1 & 3000 & 4 \end{array} \right] \quad y = \left[\begin{array}{c} 400 \\ 330 \\ 369 \\ 232 \\ 540 \end{array} \right]$$

$$\begin{aligned}\frac{\partial J(w)}{\partial w_0} &= \frac{2}{m} \sum_{i=1}^m (w^T x^{(i)} - y^{(i)}) x_0^{(i)} \\ &= \frac{2}{5} [(w_0 + 2104w_1 + 3w_2 - 400) + (w_0 + 1600w_1 + 3w_2 - 330) + \\ &\quad (w_0 + 2400w_1 + 3w_2 - 369) + (w_0 + 1416w_1 + 2w_2 - 232) + (w_0 + 3000w_1 + 4w_2 - 540)]\end{aligned}$$



Batch Gradient Descent - Example (3/5)

$$X = \left[\begin{array}{c|ccc} 1 & 2104 & 3 \\ 1 & 1600 & 3 \\ 1 & 2400 & 3 \\ 1 & 1416 & 2 \\ 1 & 3000 & 4 \end{array} \right] \quad y = \left[\begin{array}{c} 400 \\ 330 \\ 369 \\ 232 \\ 540 \end{array} \right]$$

$$\begin{aligned}\frac{\partial J(w)}{\partial w_1} &= \frac{2}{m} \sum_{i=1}^m (w^\top x^{(i)} - y^{(i)}) x_1^{(i)} \\ &= \frac{2}{5} [2104(w_0 + 2104w_1 + 3w_2 - 400) + 1600(w_0 + 1600w_1 + 3w_2 - 330) + \\ &\quad 2400(w_0 + 2400w_1 + 3w_2 - 369) + 1416(w_0 + 1416w_1 + 2w_2 - 232) + 3000(w_0 + 3000w_1 + 4w_2 - 540)]\end{aligned}$$

Batch Gradient Descent - Example (4/5)

$$X = \left[\begin{array}{c|ccc} 1 & 2104 & 3 \\ 1 & 1600 & 3 \\ 1 & 2400 & 3 \\ 1 & 1416 & 2 \\ 1 & 3000 & 4 \end{array} \right] \quad y = \left[\begin{array}{c} 400 \\ 330 \\ 369 \\ 232 \\ 540 \end{array} \right]$$

$$\begin{aligned}\frac{\partial J(w)}{\partial w_2} &= \frac{2}{m} \sum_{i=1}^m (w^T x^{(i)} - y^{(i)}) x_2^{(i)} \\ &= \frac{2}{5} [3(w_0 + 2104w_1 + 3w_2 - 400) + 3(w_0 + 1600w_1 + 3w_2 - 330) + \\ &\quad 3(w_0 + 2400w_1 + 3w_2 - 369) + 2(w_0 + 1416w_1 + 2w_2 - 232) + 4(w_0 + 3000w_1 + 4w_2 - 540)]\end{aligned}$$



Batch Gradient Descent - Example (5/5)

$$w_0^{(\text{next})} = w_0 - \eta \frac{\partial J(w)}{\partial w_0}$$

$$w_1^{(\text{next})} = w_1 - \eta \frac{\partial J(w)}{\partial w_1}$$

$$w_2^{(\text{next})} = w_2 - \eta \frac{\partial J(w)}{\partial w_2}$$



Stochastic Gradient Descent



Stochastic Gradient Descent (1/3)

- ▶ Batch gradient descent problem: it's slow, because it uses the whole training set to compute the gradients at every step.

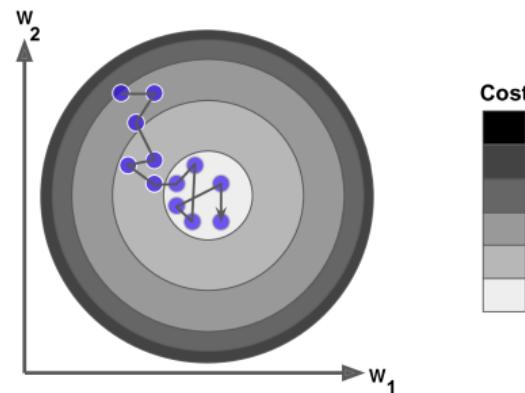


Stochastic Gradient Descent (1/3)

- ▶ **Batch gradient descent problem:** it's **slow**, because it uses the **whole training set** to compute the gradients at **every step**.
- ▶ **Stochastic gradient descent** computes the gradients based on only a **single instance**.
 - It picks a **random instance** in the **training set** at **every step**.

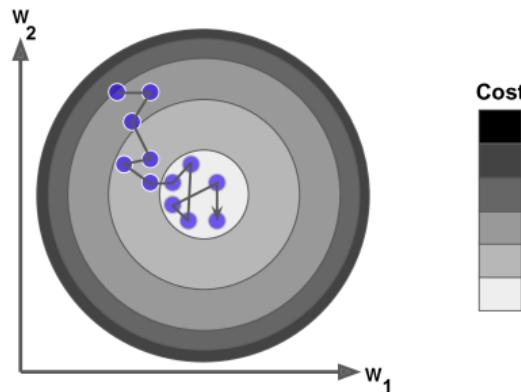
Stochastic Gradient Descent (2/3)

- ▶ The algorithm is much **faster**, but **less regular** than batch gradient descent.



Stochastic Gradient Descent (2/3)

- ▶ The algorithm is much **faster**, but **less regular** than batch gradient descent.
 - Instead of decreasing until it reaches the minimum, the **cost function will bounce up and down**.
 - It **never settles down**.





Stochastic Gradient Descent (3/3)

- ▶ With randomness the algorithm **can never settle at the minimum**.
- ▶ One solution is **simulated annealing**: start with **large learning rate**, then make it **smaller and smaller**.



Stochastic Gradient Descent (3/3)

- ▶ With randomness the algorithm **can never settle at the minimum**.
- ▶ One solution is **simulated annealing**: start with **large learning rate**, then make it **smaller and smaller**.
- ▶ **Learning schedule**: the function that **determines the learning rate** at each step.

Stochastic Gradient Descent - Example (1/3)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2$$

$$X = \left[\begin{array}{ccc|c} 1 & 2104 & 3 & 400 \\ 1 & 1600 & 3 & 330 \\ 1 & 2400 & 3 & 369 \\ 1 & 1416 & 2 & 232 \\ 1 & 3000 & 4 & 540 \end{array} \right] \quad y = \begin{bmatrix} 400 \\ 330 \\ 369 \\ 232 \\ 540 \end{bmatrix}$$

Stochastic Gradient Descent - Example (2/3)

$$X = \left[\begin{array}{c|ccc} 1 & 2104 & 3 \\ 1 & 1600 & 3 \\ 1 & 2400 & 3 \\ 1 & 1416 & 2 \\ 1 & 3000 & 4 \end{array} \right] \quad y = \left[\begin{array}{c} 400 \\ 330 \\ 369 \\ 232 \\ 540 \end{array} \right]$$

$$\frac{\partial J(w)}{\partial w_0} = \frac{2}{m}(w^T x^{(i)} - y^{(i)})x_0^{(i)} = \frac{2}{5}[(w_0 + 1600w_1 + 3w_2 - 330)]$$

$$\frac{\partial J(w)}{\partial w_1} = \frac{2}{m}(w^T x^{(i)} - y^{(i)})x_1^{(i)} = \frac{2}{5}[1600(w_0 + 1600w_1 + 3w_2 - 330)]$$

$$\frac{\partial J(w)}{\partial w_2} = \frac{2}{m}(w^T x^{(i)} - y^{(i)})x_2^{(i)} = \frac{2}{5}[3(w_0 + 1600w_1 + 3w_2 - 330)]$$



Stochastic Gradient Descent - Example (3/3)

$$w_0^{(\text{next})} = w_0 - \eta \frac{\partial J(w)}{\partial w_0}$$

$$w_1^{(\text{next})} = w_1 - \eta \frac{\partial J(w)}{\partial w_1}$$

$$w_2^{(\text{next})} = w_2 - \eta \frac{\partial J(w)}{\partial w_2}$$



Mini-Batch Gradient Descent

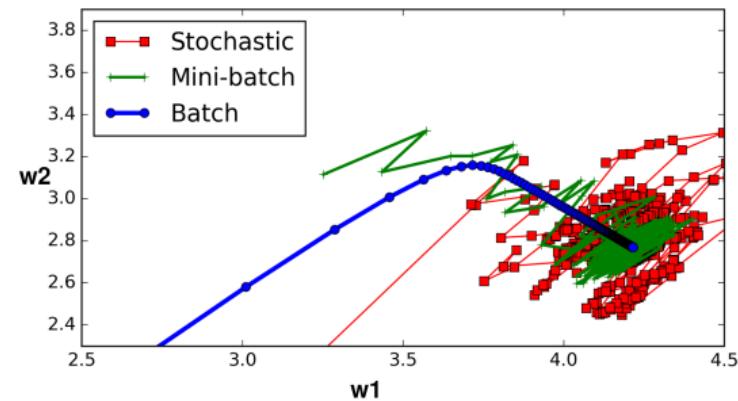


Mini-Batch Gradient Descent

- ▶ **Batch gradient descent**: at each step, it computes the gradients based on the **full training set**.
- ▶ **Stochastic gradient descent**: at each step, it computes the gradients based on **just one instance**.
- ▶ **Mini-batch gradient descent**: at each step, it computes the gradients based on small random sets of instances called **mini-batches**.

Comparison of Algorithms for Linear Regression

Algorithm	Large m	Large n
Normal Equation	Fast	Slow
Batch GD	Slow	Fast
Stochastic GD	Fast	Fast
Mini-batch GD	Fast	Fast





Gradient Descent in Spark

```
val data = spark.read.format("libsvm").load("data.txt")
```



Gradient Descent in Spark

```
val data = spark.read.format("libsvm").load("data.txt")
```

```
import org.apache.spark.ml.regression.LinearRegression
```

```
val lr = new LinearRegression().setMaxIter(10)
```

```
val lrModel = lr.fit(data)
```



Gradient Descent in Spark

```
val data = spark.read.format("libsvm").load("data.txt")

import org.apache.spark.ml.regression.LinearRegression

val lr = new LinearRegression().setMaxIter(10)

val lrModel = lr.fit(data)

println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")

val trainingSummary = lrModel.summary
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
```



Generalization



Training Data and Test Data

- ▶ Split data into a **training set** and a **test set**.

```
val data = spark.read.format("libsvm").load("data.txt")
val Array(trainDF, testDF) = data.randomSplit(Array(0.8, 0.2))
```

Full Dataset:

Training Data	Test Data
---------------	-----------



Training Data and Test Data

- ▶ Split data into a **training set** and a **test set**.
- ▶ Use **training set** when **training a machine learning model**.
 - Compute **training error** on the training set.
 - Try to **reduce** this training error.

```
val data = spark.read.format("libsvm").load("data.txt")  
  
val Array(trainDF, testDF) = data.randomSplit(Array(0.8, 0.2))
```

Full Dataset:

Training Data	Test Data
---------------	-----------



Training Data and Test Data

- ▶ Split data into a **training set** and a **test set**.
- ▶ Use **training set** when **training a machine learning model**.
 - Compute **training error** on the training set.
 - Try to **reduce** this training error.
- ▶ Use **test set** to **measure the accuracy of the model**.
 - **Test error** is the error when you run the **trained model** on **test data (new data)**.

```
val data = spark.read.format("libsvm").load("data.txt")  
  
val Array(trainDF, testDF) = data.randomSplit(Array(0.8, 0.2))
```

Full Dataset:

Training Data	Test Data
---------------	-----------



Generalization

- ▶ **Generalization:** make a model that performs **well** on **test data**.
 - Have a **small test error**.



Generalization

- ▶ **Generalization:** make a model that performs **well** on **test data**.
 - Have a **small test error**.
- ▶ **Challenges**
 1. Make the **training error small**.
 2. Make the **gap** between **training** and **test error small**.



More About The Test Error

- ▶ The **test error** is defined as the **expected value** of the **error** on test set.

$$\text{MSE} = \frac{1}{k} \sum_i^k (\hat{y}^{(i)} - y^{(i)})^2, \text{ k: the num. of instances in the test set}$$
$$= E[(\hat{y} - y)^2]$$



More About The Test Error

- The **test error** is defined as the **expected value** of the **error** on test set.

$$\begin{aligned} \text{MSE} &= \frac{1}{k} \sum_i^k (\hat{y}^{(i)} - y^{(i)})^2, \quad k: \text{the num. of instances in the test set} \\ &= E[(\hat{y} - y)^2] \end{aligned}$$

- A model's **test error** can be expressed as the **sum** of **bias** and **variance**.

$$E[(\hat{y} - y)^2] = \text{Bias}[\hat{y}, y]^2 + \text{Var}[\hat{y}] + \varepsilon^2$$





Bias and Underfitting

- ▶ Bias: the expected deviation from the true value of the function.

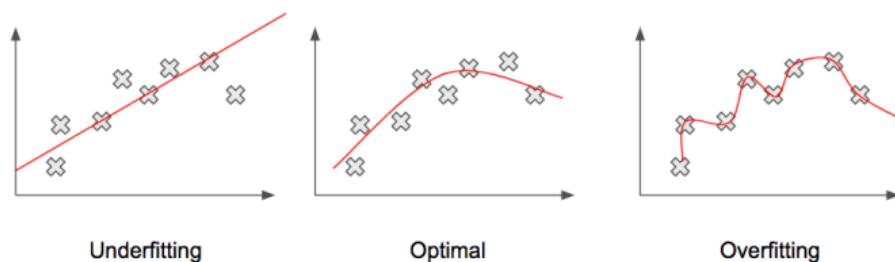
$$\text{Bias}[\hat{y}, y] = E[\hat{y}] - y$$

Bias and Underfitting

- ▶ Bias: the expected deviation from the true value of the function.

$$\text{Bias}[\hat{y}, y] = E[\hat{y}] - y$$

- ▶ A high-bias model is most likely to underfit the training data.
 - High error value on the training set.

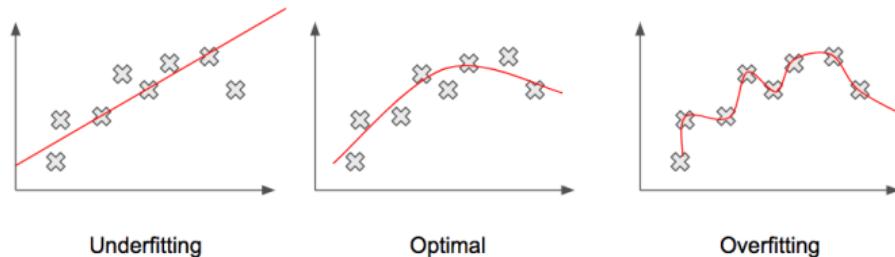


Bias and Underfitting

- ▶ Bias: the expected deviation from the true value of the function.

$$\text{Bias}[\hat{y}, y] = E[\hat{y}] - y$$

- ▶ A high-bias model is most likely to underfit the training data.
 - High error value on the training set.
- ▶ Underfitting happens when the model is too simple to learn the underlying structure of the data.





Variance and Overfitting

- ▶ **Variance:** how much a model changes if you train it on a different training set.

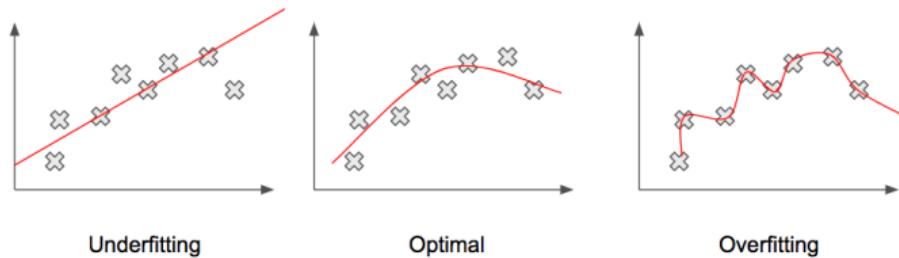
$$\text{Var}[\hat{y}] = E[(\hat{y} - E[\hat{y}])^2]$$

Variance and Overfitting

- ▶ **Variance**: how much a model changes if you train it on a different training set.

$$\text{Var}[\hat{y}] = E[(\hat{y} - E[\hat{y}])^2]$$

- ▶ A **high-variance** model is most likely to **overfit** the training data.
 - The **gap** between the **training error** and **test error** is **too large**.

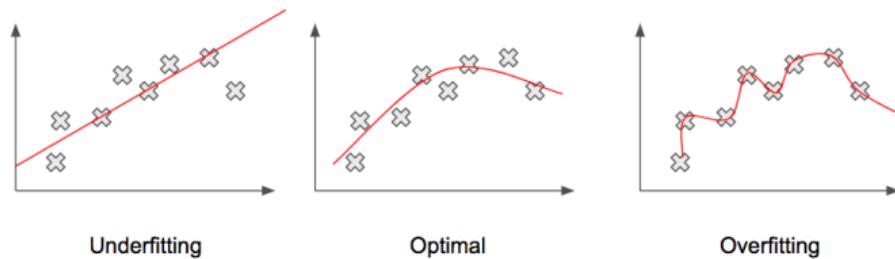


Variance and Overfitting

- ▶ **Variance**: how much a model changes if you train it on a different training set.

$$\text{Var}[\hat{y}] = E[(\hat{y} - E[\hat{y}])^2]$$

- ▶ A **high-variance** model is most likely to **overfit** the training data.
 - The **gap** between the **training error** and **test error** is **too large**.
- ▶ **Overfitting** happens when the **model is too complex** relative to the amount and noisiness of the training data.





The Bias/Variance Tradeoff (1/2)

- ▶ Assume a model with two parameters w_0 (intercept) and w_1 (slope): $\hat{y} = w_0 + w_1 x$



The Bias/Variance Tradeoff (1/2)

- ▶ Assume a model with **two parameters w_0** (**intercept**) and **w_1** (**slope**): $\hat{y} = w_0 + w_1 x$
- ▶ They give the learning algorithm **two degrees of freedom**.



The Bias/Variance Tradeoff (1/2)

- ▶ Assume a model with two parameters w_0 (intercept) and w_1 (slope): $\hat{y} = w_0 + w_1x$
- ▶ They give the learning algorithm two degrees of freedom.
- ▶ We tweak both the w_0 and w_1 to adapt the model to the training data.

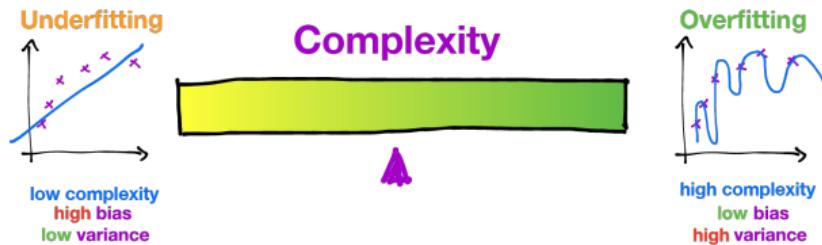


The Bias/Variance Tradeoff (1/2)

- ▶ Assume a model with two parameters w_0 (intercept) and w_1 (slope): $\hat{y} = w_0 + w_1 x$
- ▶ They give the learning algorithm two degrees of freedom.
- ▶ We tweak both the w_0 and w_1 to adapt the model to the training data.
- ▶ If we forced $w_0 = 0$, the algorithm would have only one degree of freedom and would have a much harder time fitting the data properly.

The Bias/Variance Tradeoff (2/2)

- ▶ Increasing degrees of freedom will typically increase its variance and reduce its bias.
- ▶ Decreasing degrees of freedom increases its bias and reduces its variance.
- ▶ This is why it is called a **tradeoff**.



© Machine Learning @ Berkeley

[<https://ml.berkeley.edu/blog/2017/07/13/tutorial-4>]



Regularization (1/2)

- ▶ One way to reduce the risk of overfitting is to have fewer degrees of freedom.
- ▶ Regularization is a technique to reduce the risk of overfitting.
- ▶ For a linear model, regularization is achieved by constraining the weights of the model.

$$J(w) = \text{MSE}(w) + \lambda R(w)$$



Regularization (2/2)

- ▶ Lasso regression (1): $R(w) = \lambda \sum_{i=1}^n |w_i|$ is added to the cost function:

$$J(w) = \text{MSE}(w) + \lambda \sum_{i=1}^n |w_i|$$



Regularization (2/2)

- ▶ Lasso regression (1): $R(w) = \lambda \sum_{i=1}^n |w_i|$ is added to the cost function:

$$J(w) = \text{MSE}(w) + \lambda \sum_{i=1}^n |w_i|$$

- ▶ Ridge regression (2): $R(w) = \lambda \sum_{i=1}^n w_i^2$ is added to the cost function.

$$J(w) = \text{MSE}(w) + \lambda \sum_{i=1}^n w_i^2$$



Regularization (2/2)

- ▶ Lasso regression (/1): $R(w) = \lambda \sum_{i=1}^n |w_i|$ is added to the cost function:

$$J(w) = \text{MSE}(w) + \lambda \sum_{i=1}^n |w_i|$$

- ▶ Ridge regression (/2): $R(w) = \lambda \sum_{i=1}^n w_i^2$ is added to the cost function.

$$J(w) = \text{MSE}(w) + \lambda \sum_{i=1}^n w_i^2$$

- ▶ ElasticNet: a middle ground between /1 and /2 regularization.

$$J(w) = \text{MSE}(w) + \alpha \lambda \sum_{i=1}^n |w_i| + (1 - \alpha) \lambda \sum_{i=1}^n w_i^2$$



Regularization in Spark

$$J(w) = \text{MSE}(w) + \alpha \lambda \sum_{i=1}^n |w_i| + (1 - \alpha) \lambda \sum_{i=1}^n w_i^2$$

- ▶ If $\alpha = 0$: L_2 regularization
- ▶ If $\alpha = 1$: L_1 regularization
- ▶ For α in $(0, 1)$: a combination of L_1 and L_2 regularizations

```
import org.apache.spark.ml.regression.LinearRegression

val lr = new LinearRegression().setElasticNetParam(0.8)

val lrModel = lr.fit(data)
```



Hyperparameters



Hyperparameters and Validation Sets (1/2)

- ▶ Hyperparameters are settings that we can use to control the behavior of a learning algorithm.



Hyperparameters and Validation Sets (1/2)

- ▶ Hyperparameters are settings that we can use to control the behavior of a learning algorithm.
- ▶ The values of hyperparameters are not adapted by the learning algorithm itself.
 - E.g., the α and λ values for regularization.



Hyperparameters and Validation Sets (1/2)

- ▶ Hyperparameters are settings that we can use to control the behavior of a learning algorithm.
- ▶ The values of hyperparameters are not adapted by the learning algorithm itself.
 - E.g., the α and λ values for regularization.
- ▶ We do not learn the hyperparameter.
 - It is not appropriate to learn that hyperparameter on the training set.
 - If learned on the training set, such hyperparameters would always result in overfitting.



Hyperparameters and Validation Sets (2/2)

- ▶ To find **hyperparameters**, we need a **validation set** of examples that the **training algorithm does not observe**.



Hyperparameters and Validation Sets (2/2)

- ▶ To find **hyperparameters**, we need a **validation set** of examples that the **training algorithm does not observe**.
- ▶ We construct the **validation set** from the **training data** (**not the test data**).

Hyperparameters and Validation Sets (2/2)

- ▶ To find **hyperparameters**, we need a **validation set** of examples that the **training algorithm does not observe**.
- ▶ We construct the **validation set** from the **training data** (**not the test data**).
- ▶ We split the **training data** into two disjoint subsets:
 1. One is used to **learn the parameters**.
 2. The other one (the **validation set**) is used to **estimate the test error** **during or after training**, allowing for the **hyperparameters** to be updated accordingly.

Full Dataset:

Training Data	Validation Data	Test Data

Cross-Validation

- ▶ **Cross-validation:** a technique to avoid **wasting too much training data in validation sets.**



Cross-Validation

- ▶ **Cross-validation:** a technique to avoid **wasting too much training data** in **validation sets**.
- ▶ The **training set** is split into **complementary subsets**.



Cross-Validation

- ▶ **Cross-validation:** a technique to avoid **wasting too much training data** in **validation sets**.
- ▶ The **training set** is split into **complementary subsets**.
- ▶ Each model is **trained** against a different **combination** of these subsets and **validated** against the **remaining parts**.



Cross-Validation

- ▶ **Cross-validation:** a technique to avoid **wasting too much training data** in **validation sets**.
- ▶ The **training set** is split into **complementary subsets**.
- ▶ Each model is **trained** against a different **combination** of these subsets and **validated** against the **remaining parts**.
- ▶ Once the model type and hyperparameters have been selected, a **final model** is trained using these hyperparameters on the **full training set**, and the test error is measured on the **test set**.





Hyperparameters and Cross-Validation in Spark (1/2)

- ▶ **CrossValidator** to optimize hyperparameters in algorithms and model selection.
- ▶ It requires the following items:
 - **Estimator**: algorithm or Pipeline to tune.
 - Set of **ParamMaps**: parameters to choose from (also called a **parameter grid**).
 - **Evaluator**: metric to measure **how well a fitted Model does on held-out test data**.



Hyperparameters and Cross-Validation in Spark (2/2)

```
// construct a grid of parameters to search over.  
// this grid has 2 x 2 = 4 parameter settings for CrossValidator to choose from.  
val paramGrid = new ParamGridBuilder()  
  .addGrid(lr.regParam, Array(0.1, 0.01))  
  .addGrid(lr.elasticNetParam, Array(0.0, 1.0))  
  .build()
```



Hyperparameters and Cross-Validation in Spark (2/2)

```
// construct a grid of parameters to search over.  
// this grid has 2 x 2 = 4 parameter settings for CrossValidator to choose from.  
val paramGrid = new ParamGridBuilder()  
  .addGrid(lr.regParam, Array(0.1, 0.01))  
  .addGrid(lr.elasticNetParam, Array(0.0, 1.0))  
  .build()
```

```
val lr = new LinearRegression()  
  
// num folds = 3 => (2 x 2) x 3 = 12 different models being trained  
val cv = new CrossValidator()  
  .setEstimator(lr)  
  .setEvaluator(new RegressionEvaluator())  
  .setEstimatorParamMaps(paramGrid)  
  .setNumFolds(3)  
  
val cvModel = cv.fit(trainDF)
```



Summary

Summary

- ▶ Linear regression model $\hat{y} = w^T x$
 - Learning parameters w
 - Cost function $J(w)$
 - Learn parameters: normal equation, gradient descent (batch, stochastic, mini-batch)
- ▶ Generalization
 - Overfitting vs. underfitting
 - Bias vs. variance
 - Regularization: Lasso regression, Ridge regression, ElasticNet
- ▶ Hyperparameters and cross-validation



Reference

- ▶ Ian Goodfellow et al., Deep Learning (Ch. 4, 5)
- ▶ Aurélien Géron, Hands-On Machine Learning (Ch. 2, 4)
- ▶ Matei Zaharia et al., Spark - The Definitive Guide (Ch. 27)



Questions?