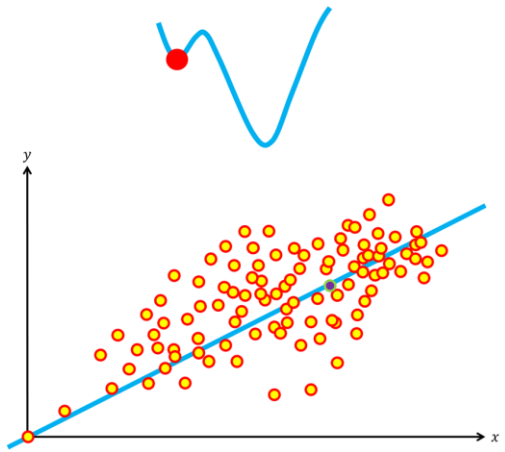
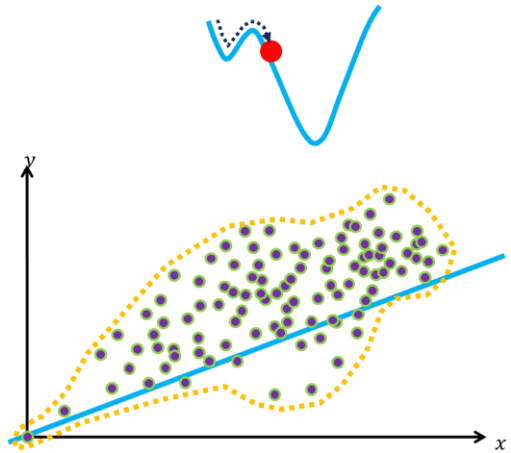
Deep Learning101

확률적 경사하강법

Stochastic Gradient Descent





신박AI

안녕하세요 신박AI입니다

신박AI

이번 영상에서는 경사하강법의 종류인

신박AI

배치 경사하강법, 확률 경사하강법
Batch gradient descent, Stochastic gradient descent

신박AI

그리고 미니-배치 경사하강법 mini-batch
gradient descent 에 대해 알아보도록 하겠습니다

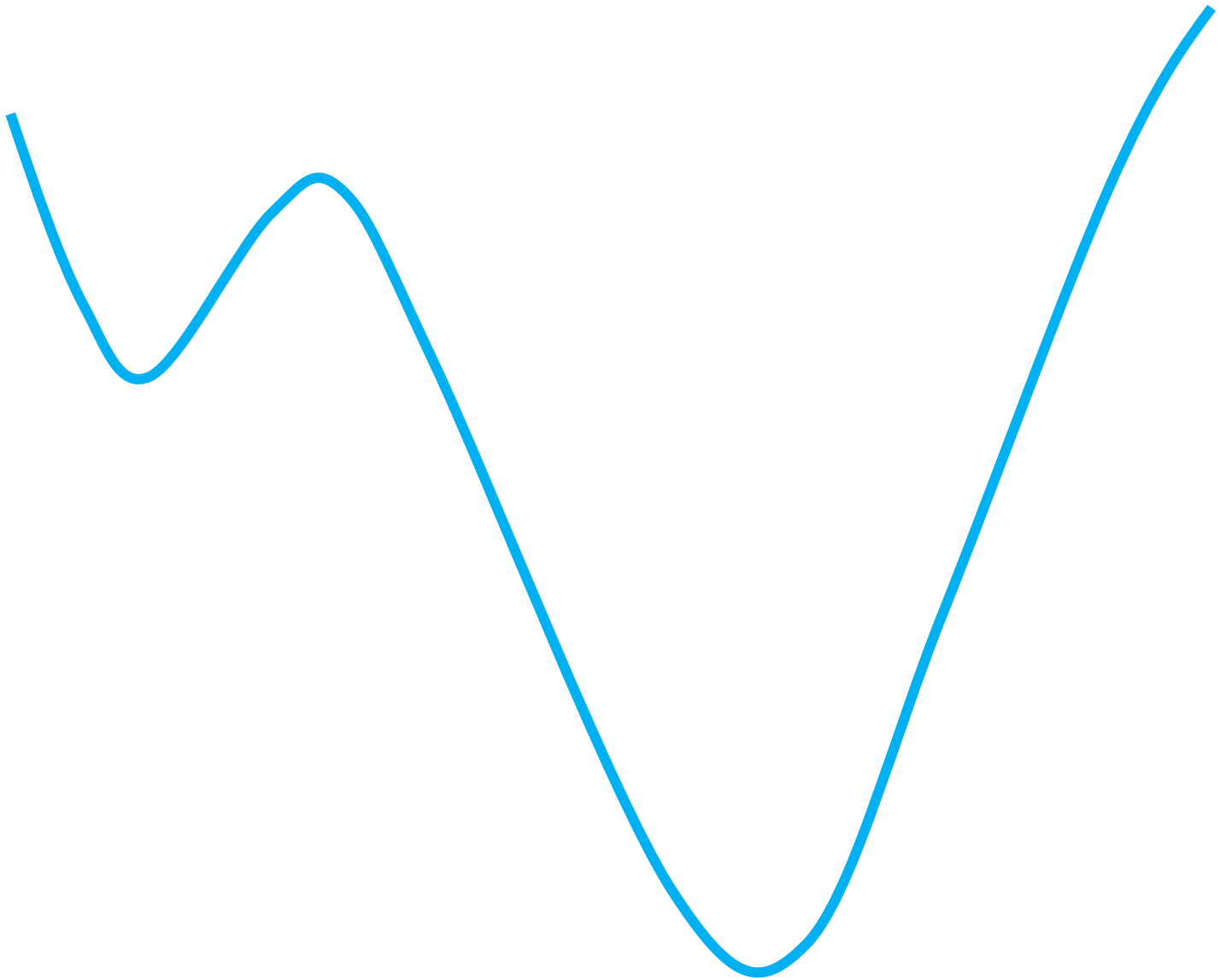
신박AI

경사하강법에서 잠시 말씀드렸듯이

신경망의 학습에는 피할 수 없는 한가지
문제점이 있는데

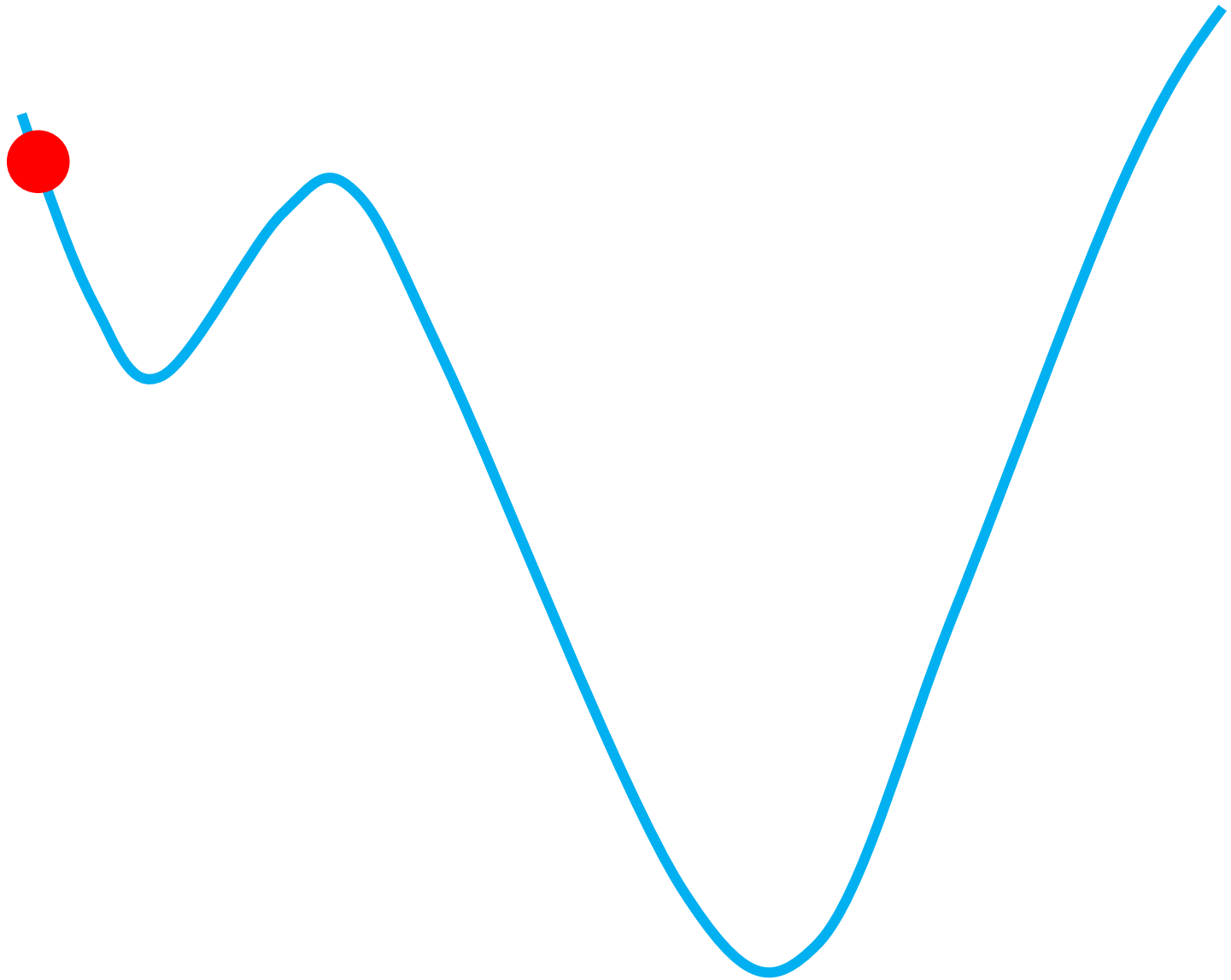
그것은 바로 로컬 미니마의 문제였습니다

로컬 미니마라는 것은 신경망의 학습과정에서 발생하는 현상으로



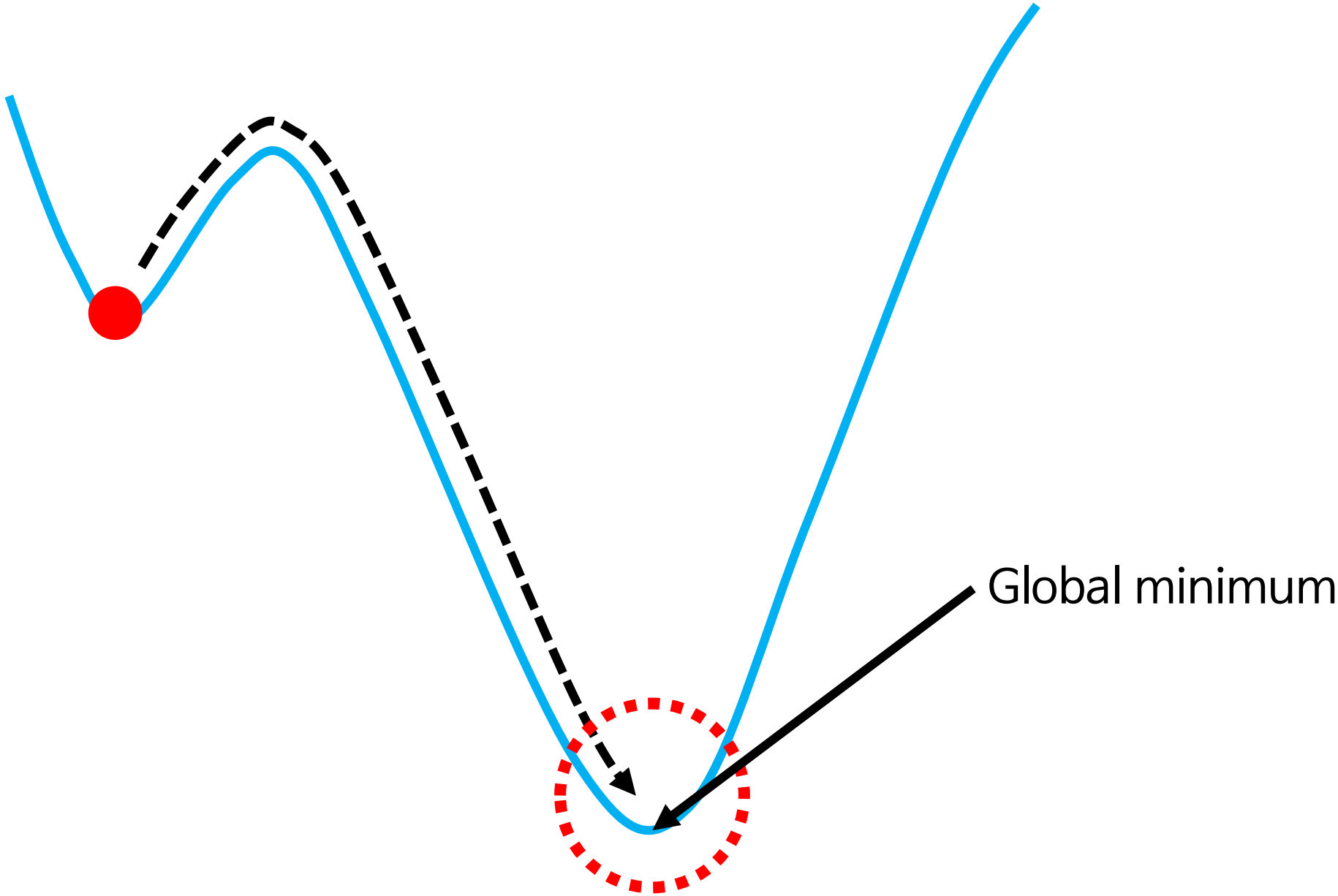
신박AI

학습과정중 손실함수의 최소값을 찾아가는 도중에



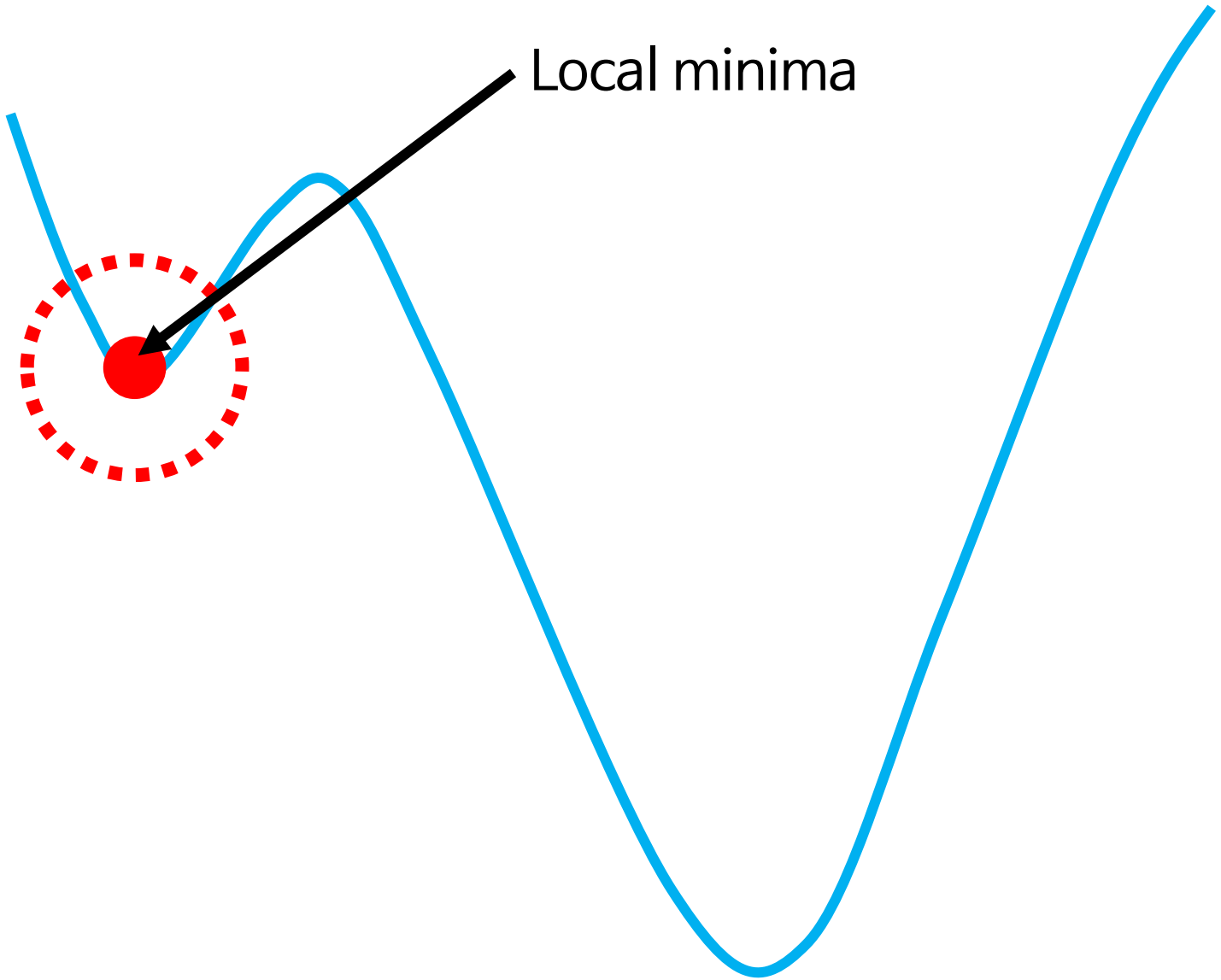
신박AI

손실함수의 전역 최소값 (Global minimum)이 아닌



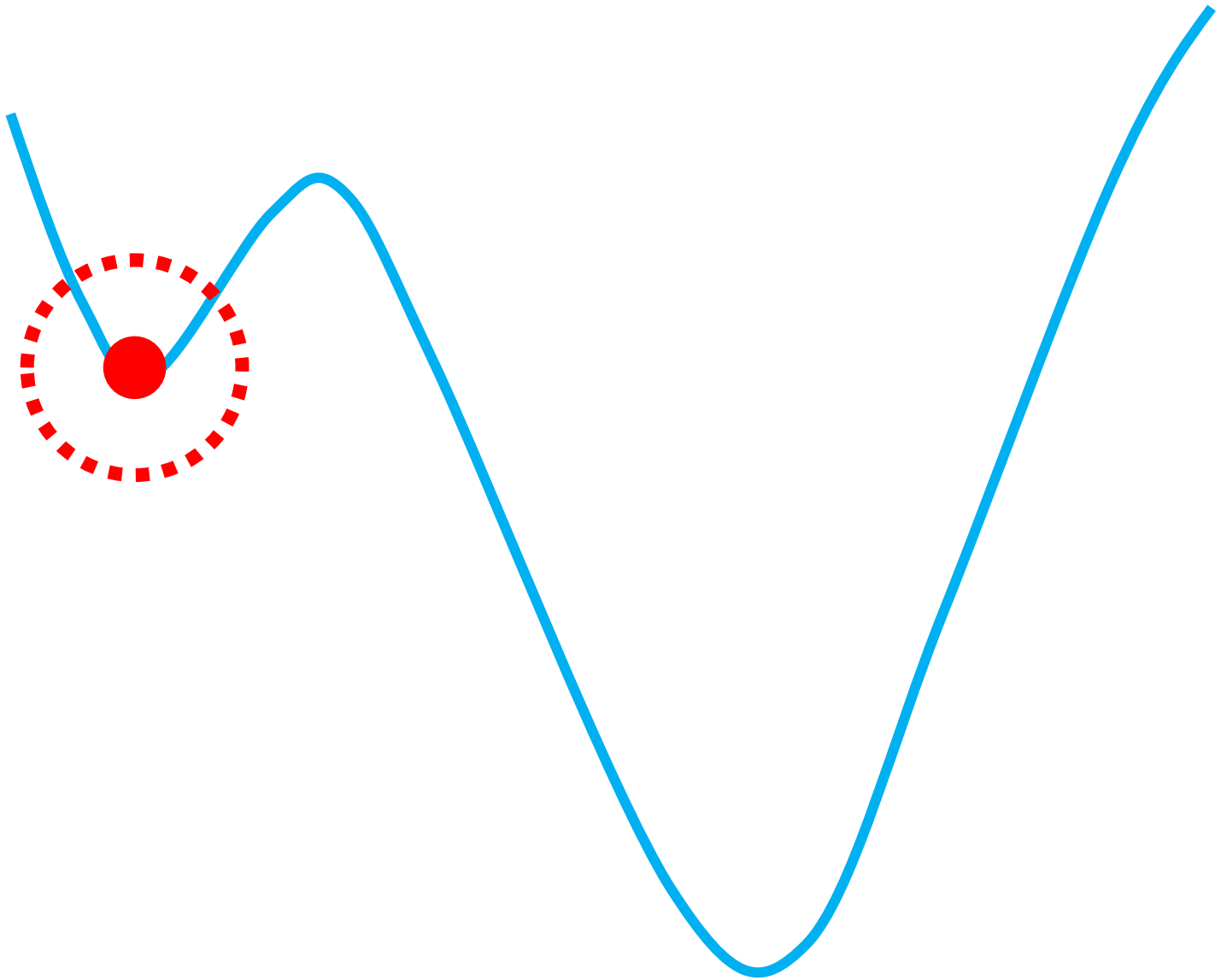
신박AI

지역 최소값 (Local minima)에 머무르는 것을 말합니다



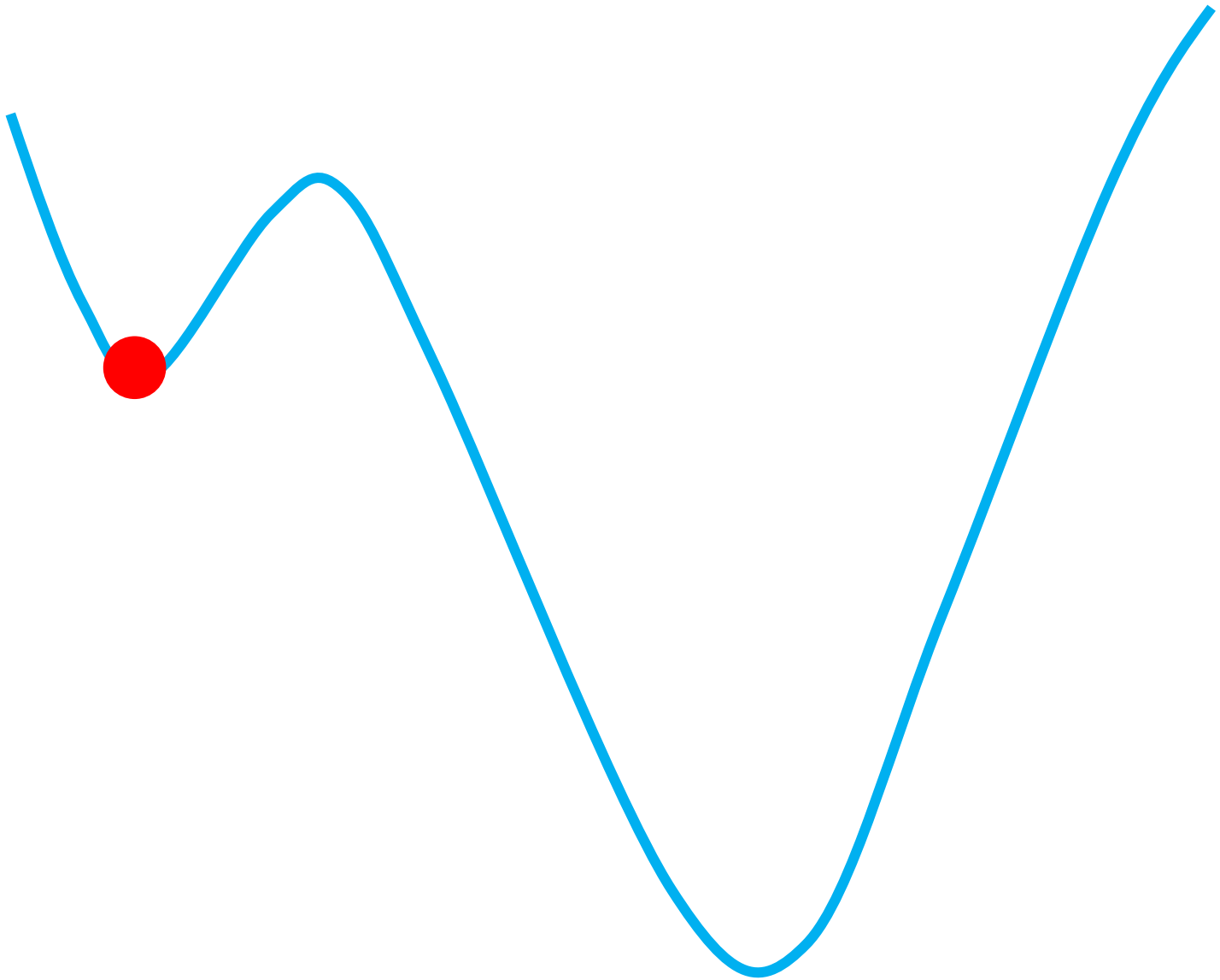
신박AI

로컬 미니마의 문제의 완전한 해결책은 없지만,



신박AI

딥러닝 학자들은 로컬 미니마 문제를 개선하기 위해



신박AI

여러가지 방법들을 개발했습니다

신박AI

여러가지 방법들을 개발했습니다

초기값 설정: Xavier 초기값 설정, He 초기값 설정 등..

신박AI

여러가지 방법들을 개발했습니다

초기값 설정: Xavier 초기값 설정, He 초기값 설정 등..

학습률 조정: AdaGrad, Adam 등..

신박AI

여러가지 방법들을 개발했습니다

초기값 설정: Xavier 초기값 설정, He 초기값 설정 등..

학습률 조정: AdaGrad, Adam 등..

학습데이터 스케줄링: 확률적 경사하강법, 미니배치 경사하강법..

신박AI

이 중에서 오늘 우리는 학습데이터 스케줄링에 관하여 다루어 보도록 하겠습니다

초기값 설정: Xavier 초기값 설정, He 초기값 설정 등..

학습률 조정: AdaGrad, Adam 등..

학습데이터 스케줄링: 확률적 경사하강법, 미니배치 경사하강법..

신박AI

학습데이터 스케줄링이라는 것은 다음과 같이 이해하시면 좋을 것 같습니다

신박AI

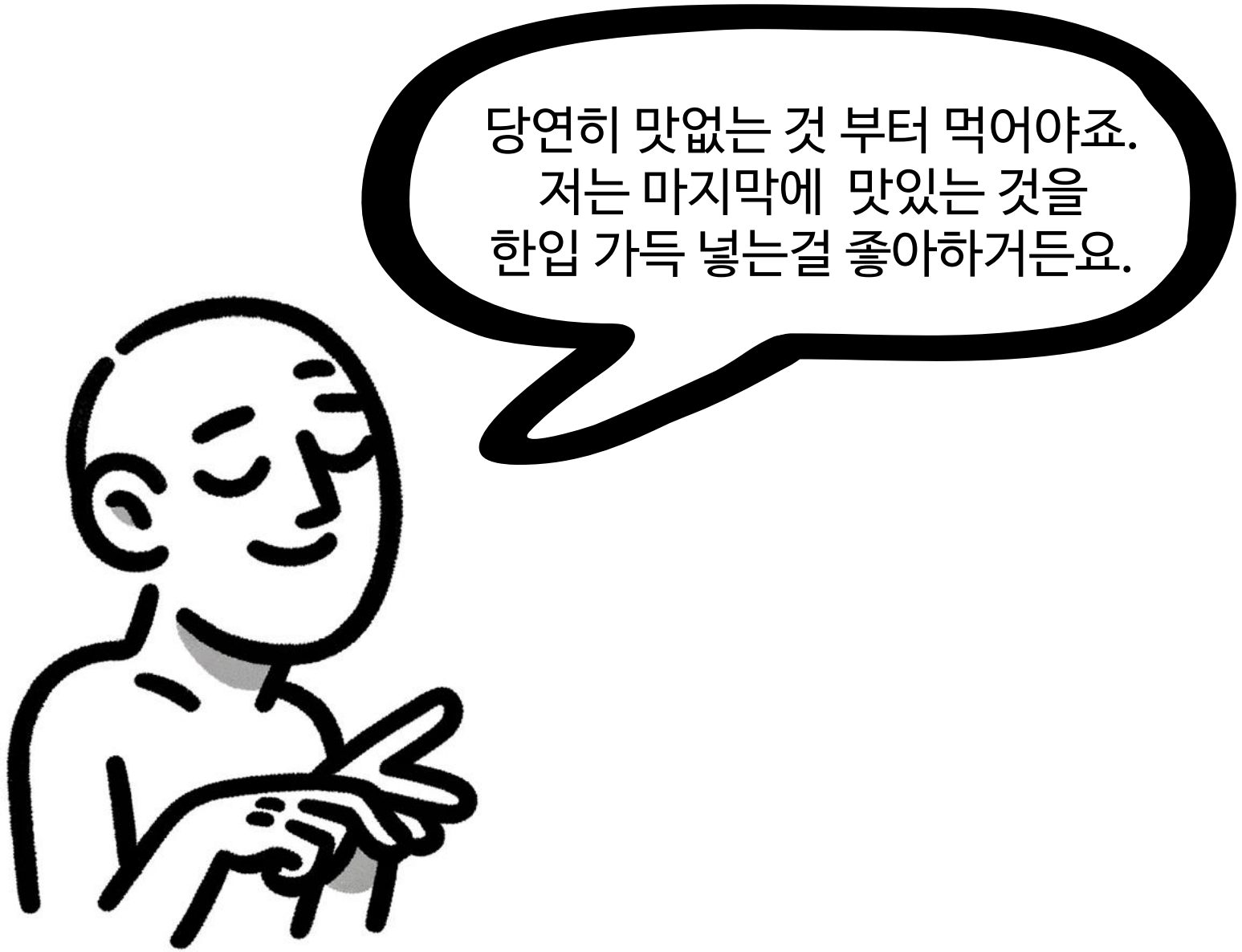
여러분들은 만약 이와 같은 음식이 나오면,

신박AI

좋아하는 것부터 드시나요 아니면 싫어하는 것부터 드시나요?

신박AI

어떤 사람들은 싫어하는 음식부터 먹기도 하고



신박AI

어떤 사람들은 좋아하는 음식을 먼저 먹기도 합니다

신박AI

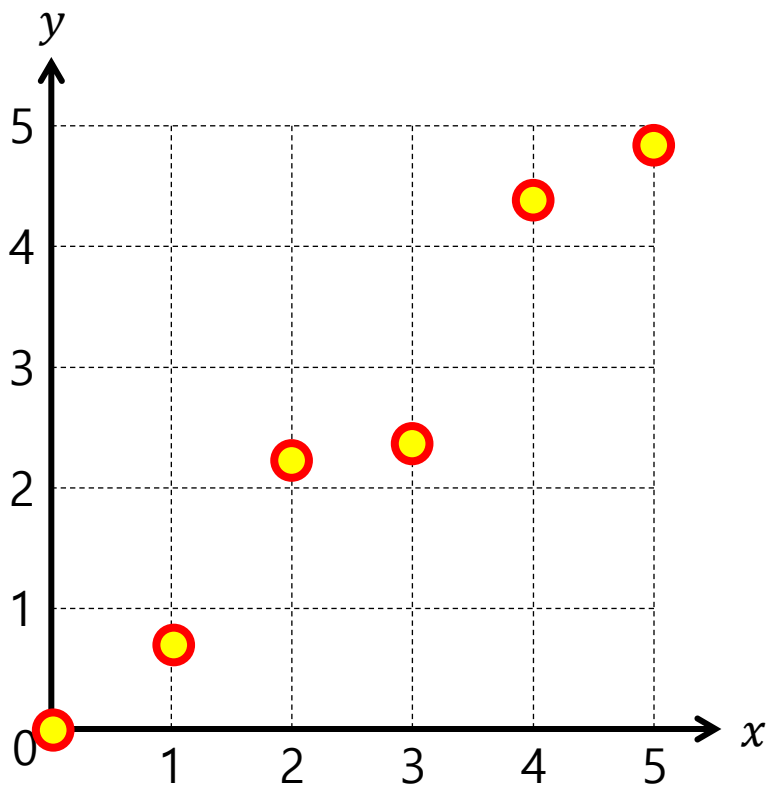
어차피 배속에 들어가는 음식의 총량은 같지만 순서를 달리함으로 만족도가 달라질 수 있습니다

신박AI

이처럼 똑 같은 전체 데이터라 할지라도 그 데이터들을 어떤 순서로 학습시키느냐에 따라 그 결과가 달라질 수가 있습니다

그것이 오늘 배울 확률적 경사하강법의 개념이라 할 수 있습니다.

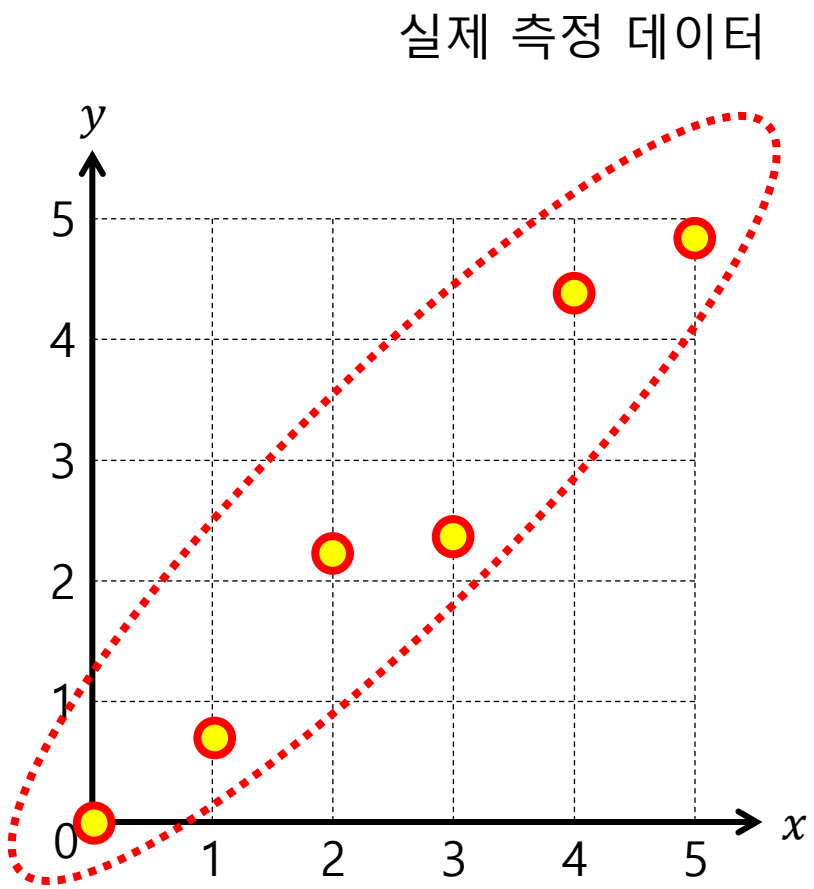
확률적 경사하강법을 보다 쉽게 설명 드리기 위해서 다음과 같은 상황을 가정해 보겠습니다



x	y	$\hat{y}$
0	0	
1	0.8	
2	2.2	
3	2.3	
4	4.3	
5	4.9	

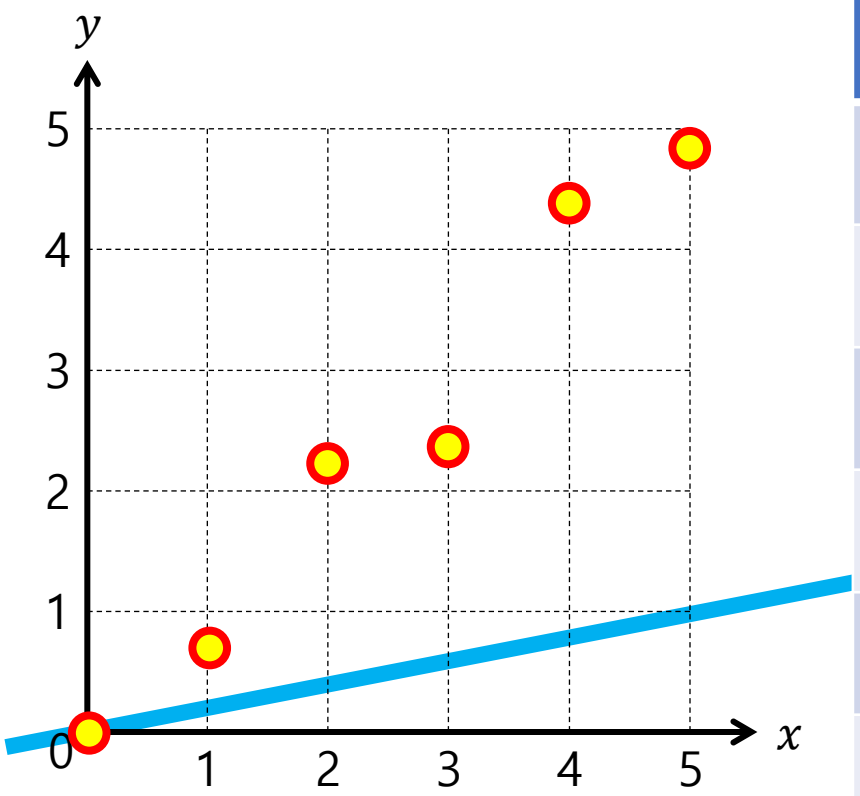
신박AI

그래프 상의 노란 점들이 실제 측정 데이터라 가정하고



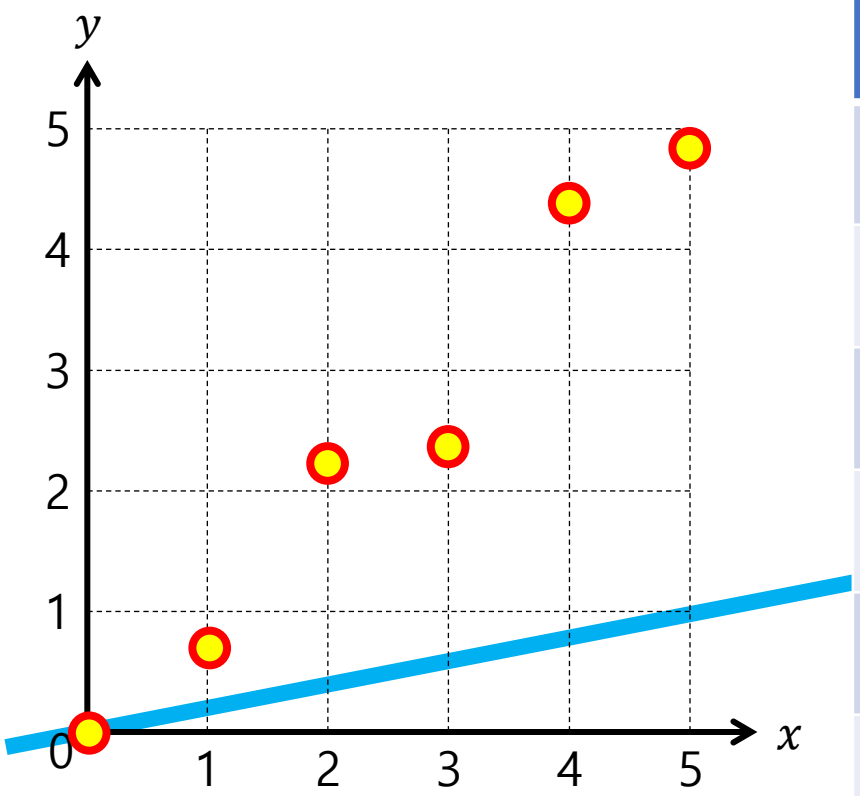
x	y	$\hat{y}$
0	0	
1	0.8	
2	2.2	
3	2.3	
4	4.3	
5	4.9	

우리는 다음과 같은 초기 직선으로부터



x	y	$\hat{y}$
0	0	
1	0.8	
2	2.2	
3	2.3	
4	4.3	
5	4.9	

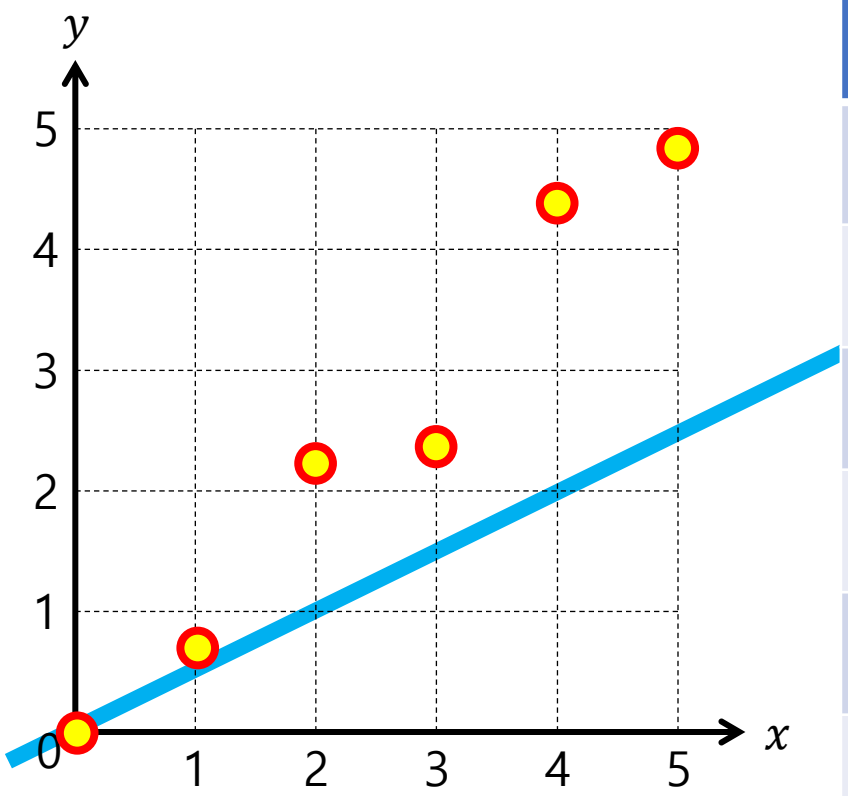
주어진 측정 데이터를 가장 잘 설명 할 수 있는 기울기를 찾아가는



x	y	$\hat{y}$
0	0	
1	0.8	
2	2.2	
3	2.3	
4	4.3	
5	4.9	

신박AI

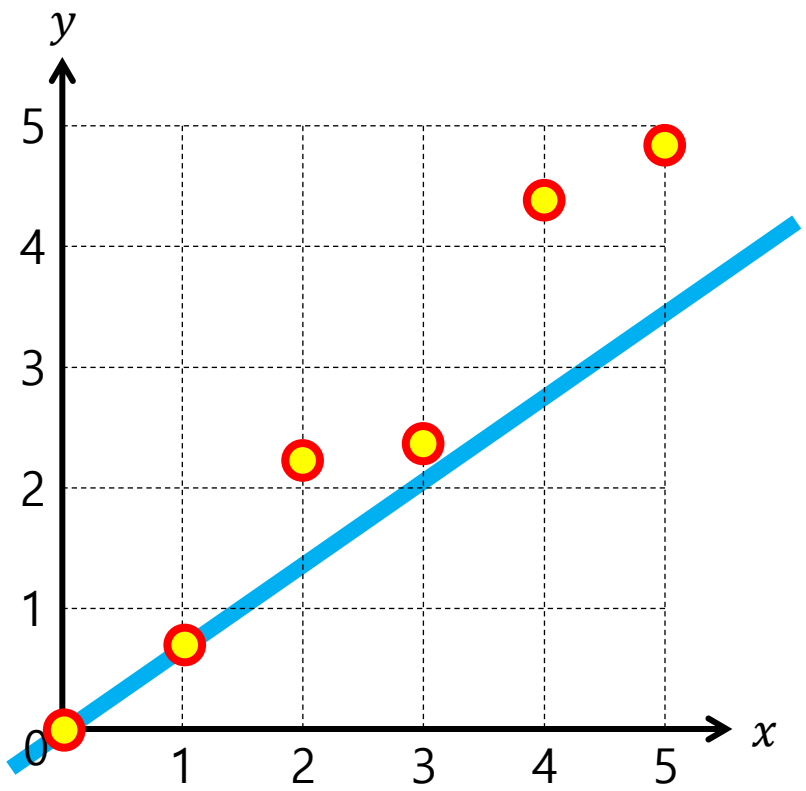
주어진 측정 데이터를 가장 잘 설명 할 수 있는 기울기를 찾아가는



x	y	$\hat{y}$
0	0	
1	0.8	
2	2.2	
3	2.3	
4	4.3	
5	4.9	

신박AI

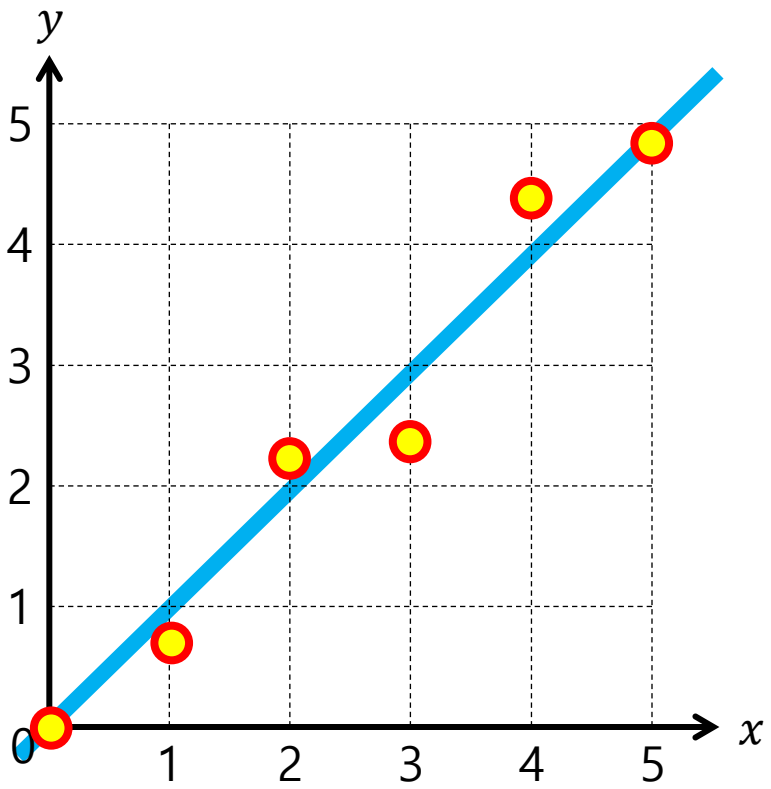
주어진 측정 데이터를 가장 잘 설명 할 수 있는 기울기를 찾아가는



x	y	$\hat{y}$
0	0	
1	0.8	
2	2.2	
3	2.3	
4	4.3	
5	4.9	

신박AI

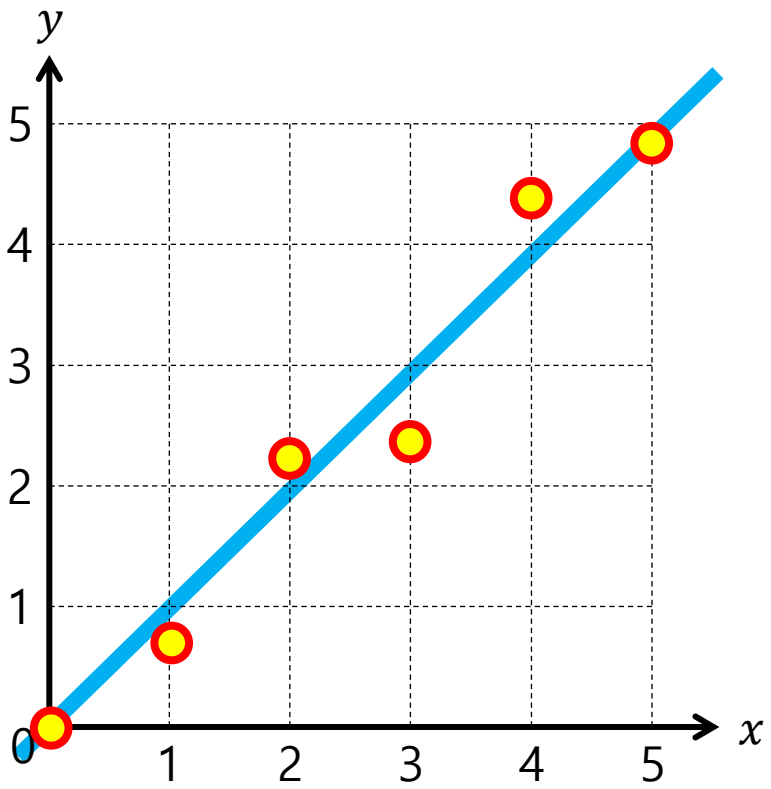
주어진 측정 데이터를 가장 잘 설명 할 수 있는 기울기를 찾아가는



x	y	$\hat{y}$
0	0	
1	0.8	
2	2.2	
3	2.3	
4	4.3	
5	4.9	

신박AI

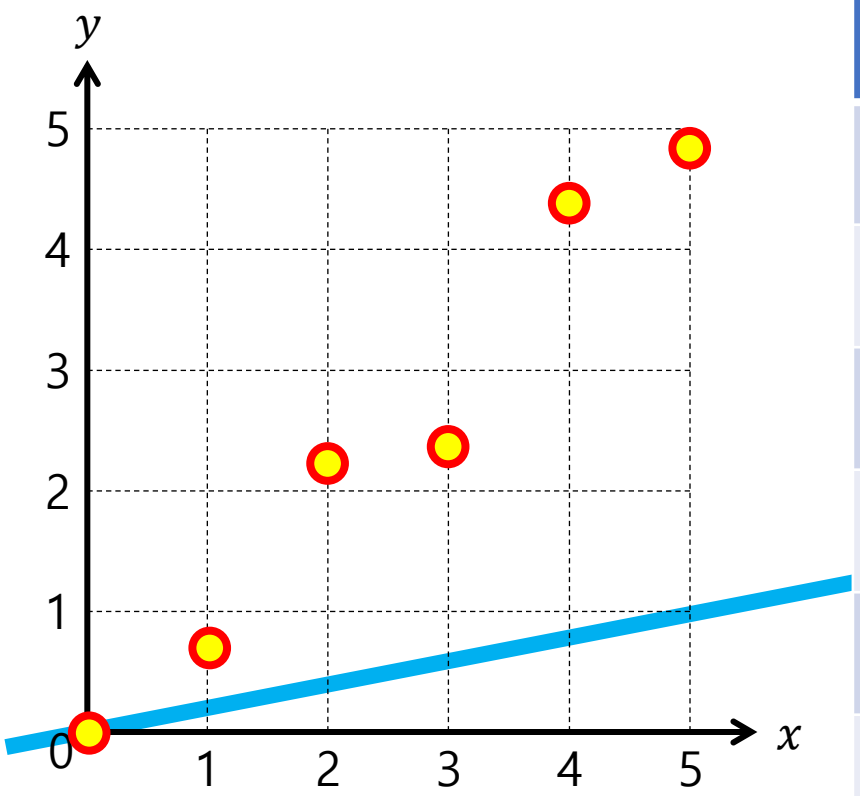
회귀모델 regression model을 가정해 보겠습니다.



x	y	$\hat{y}$
0	0	
1	0.8	
2	2.2	
3	2.3	
4	4.3	
5	4.9	

신박AI

우선 초기 조건은 다음과 같고 손실함수는 MSE (Mean Squared Error) 함수를 사용하도록 하겠습니다



x	y	$\hat{y}$
0	0	
1	0.8	
2	2.2	
3	2.3	
4	4.3	
5	4.9	

$$\hat{y} = 0.2x$$

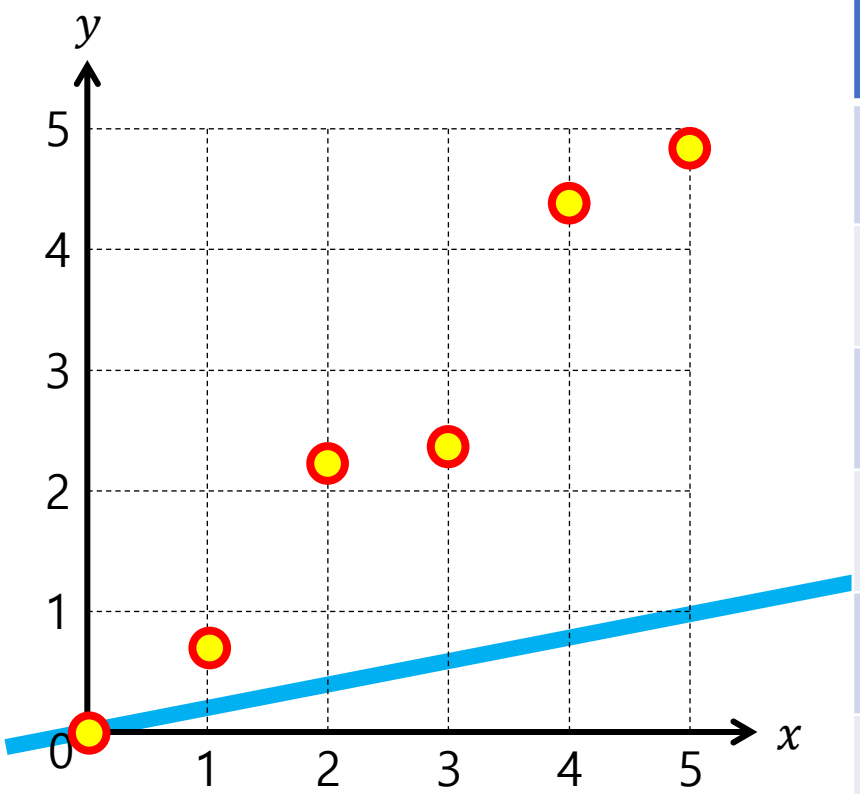
$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

신박AI

그러면 이 공식에 의해서 예측값은 다음과 같이 계산할 수 있습니다.

$$\hat{y} = 0.2x$$

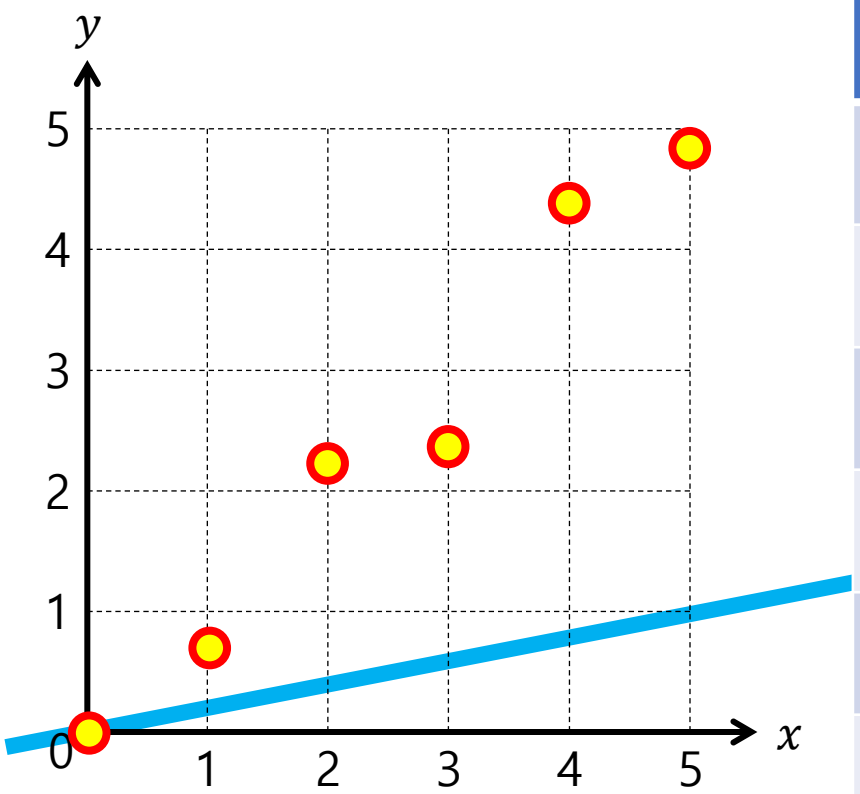
$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

신박AI

그리고 손실도 다음과 같이 계산해 볼 수 있습니다.



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$= \frac{1}{6}\{(0-0)^2 + (0.8-0.2)^2 + (2.2-0.4)^2 + (2.3-0.6)^2 + (4.3-0.8)^2 + (4.9-1)^2\}$$

$$= 5.66$$

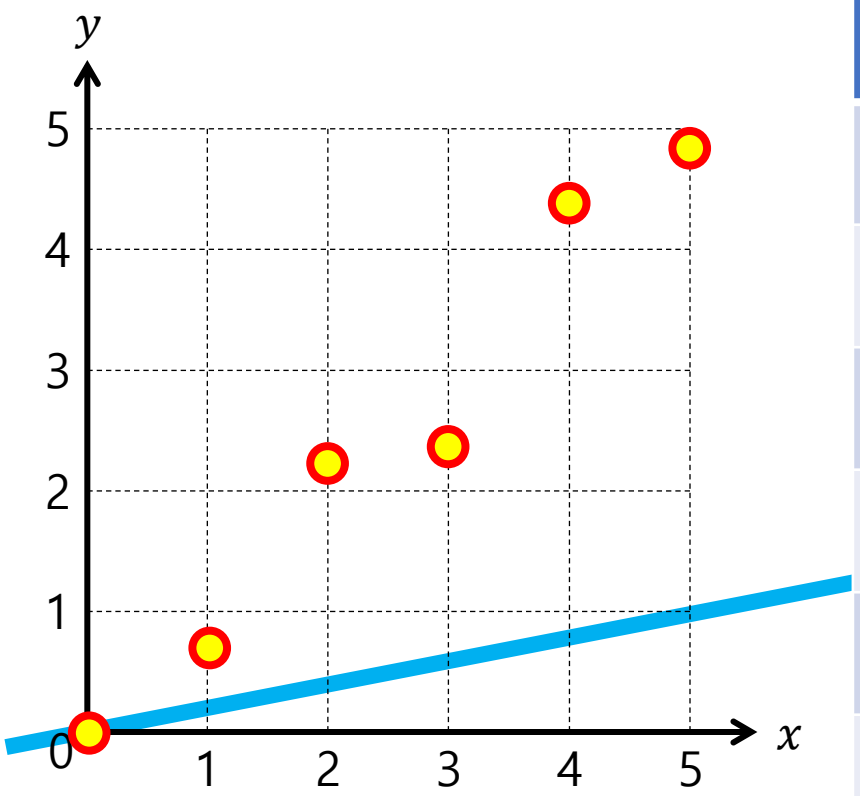
신박AI

그리고 경사하강법에 의해서 직선의 기울기를 다음과 같이 업데이트 할 수도 있습니다.

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

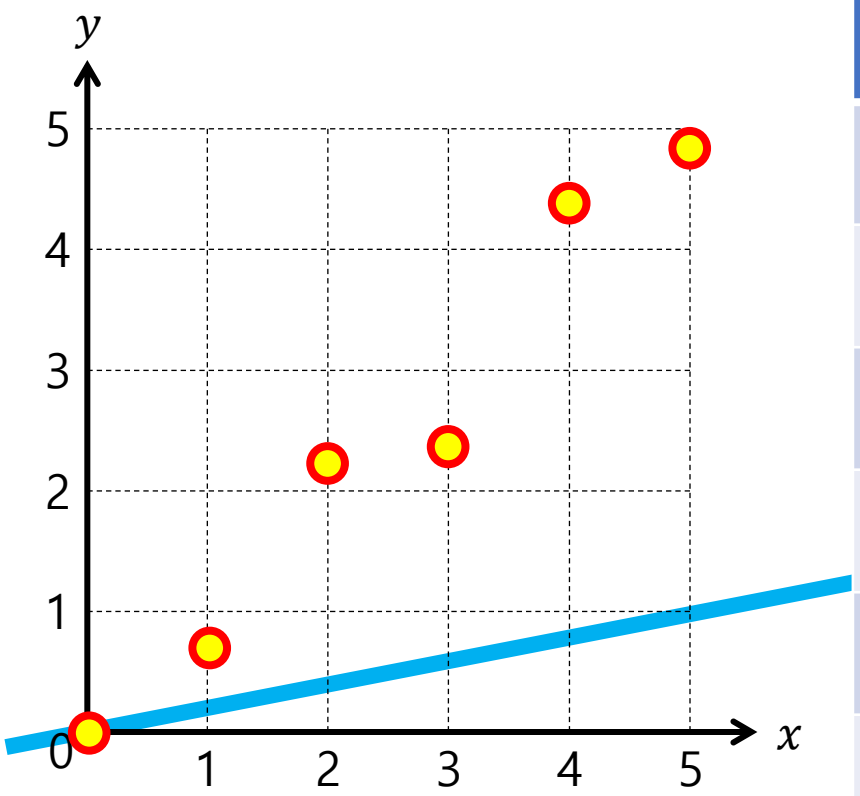
$$w^* = w - \alpha\frac{\partial L}{\partial w}$$



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

신박AI

체인룰 (연쇄법칙)에 의해서 식을 전개할 수 있고,



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

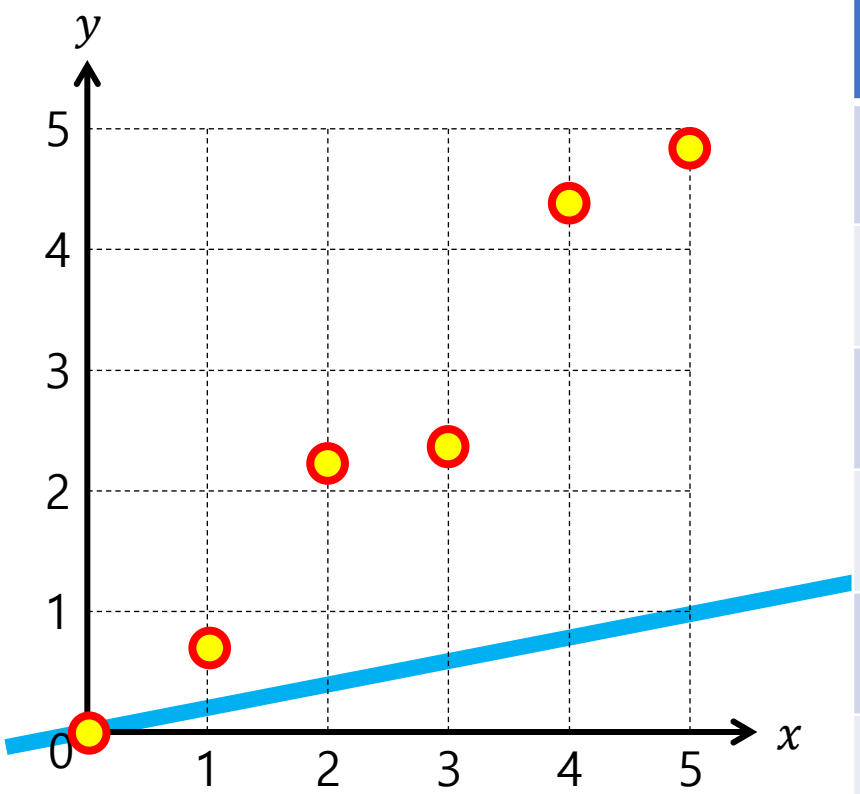
$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

체인룰

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

신박AI

이 부분을 편미분한 값은 다음과 같습니다



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

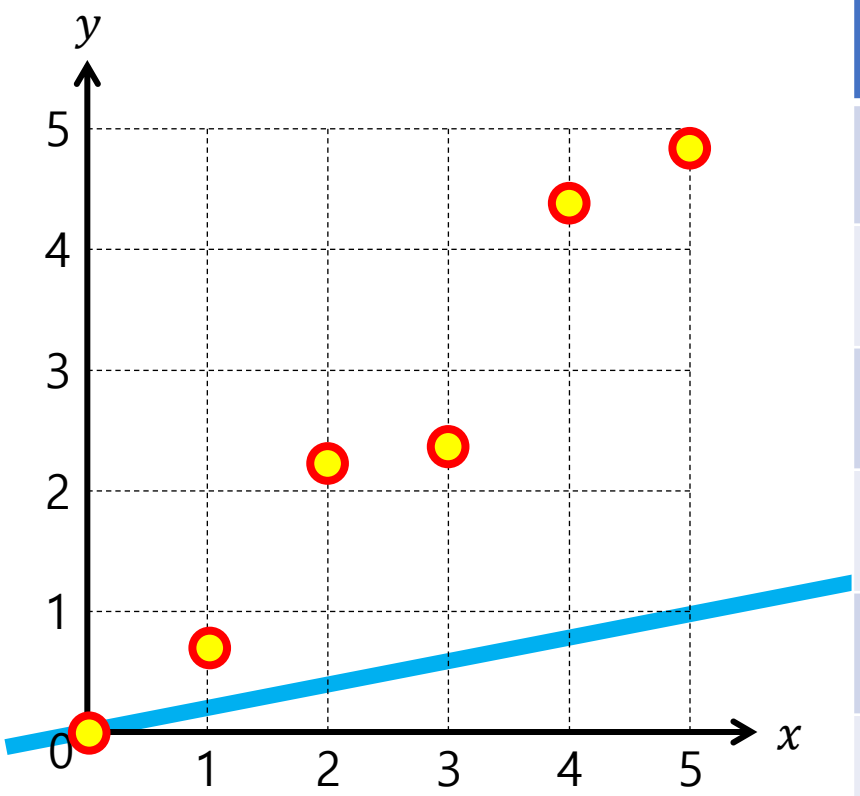
체인룰

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\right)\frac{\partial \hat{y}}{\partial w}$$

신박AI

그리고 이 부분은 편미분에 의해서 단순히 이렇게 됩니다



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

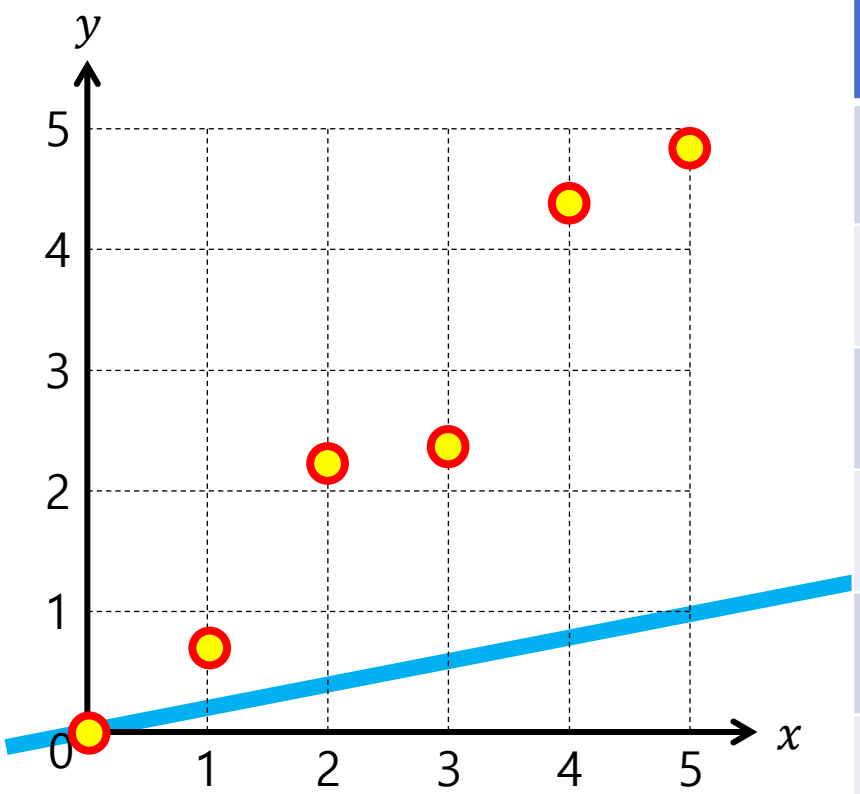
체인룰

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

신박AI

그리고 숫자를 넣어서 계산한다면, 다음과 같이 할 수 있습니다



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

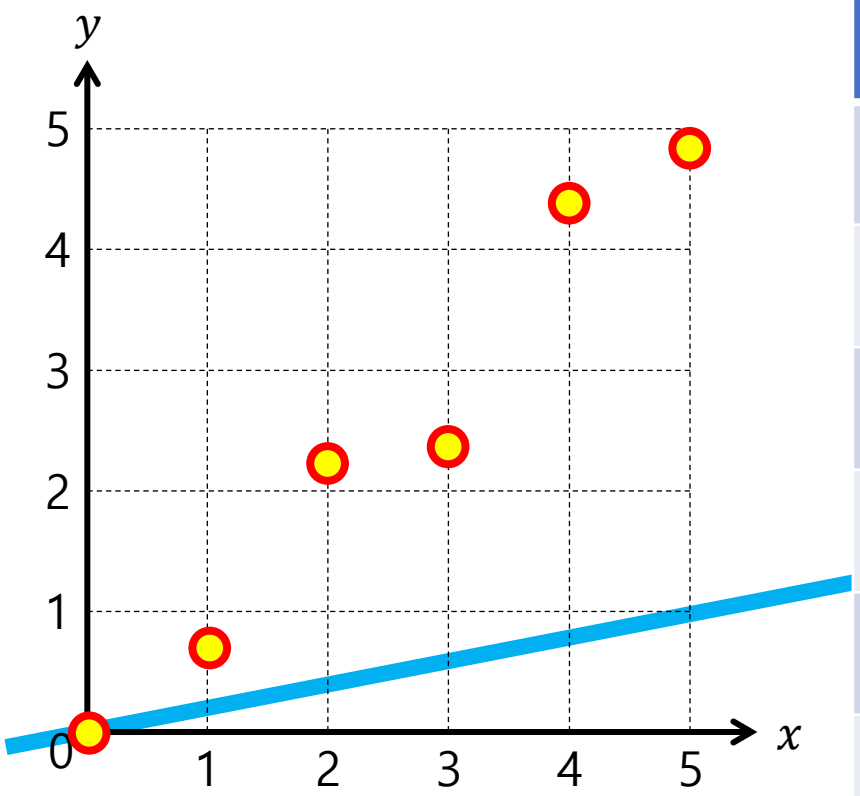
$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.2 - \alpha\left(\frac{1}{6}\cdot(-2)\cdot\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

신박AI

학습률을 0.01로 가정한다면,



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

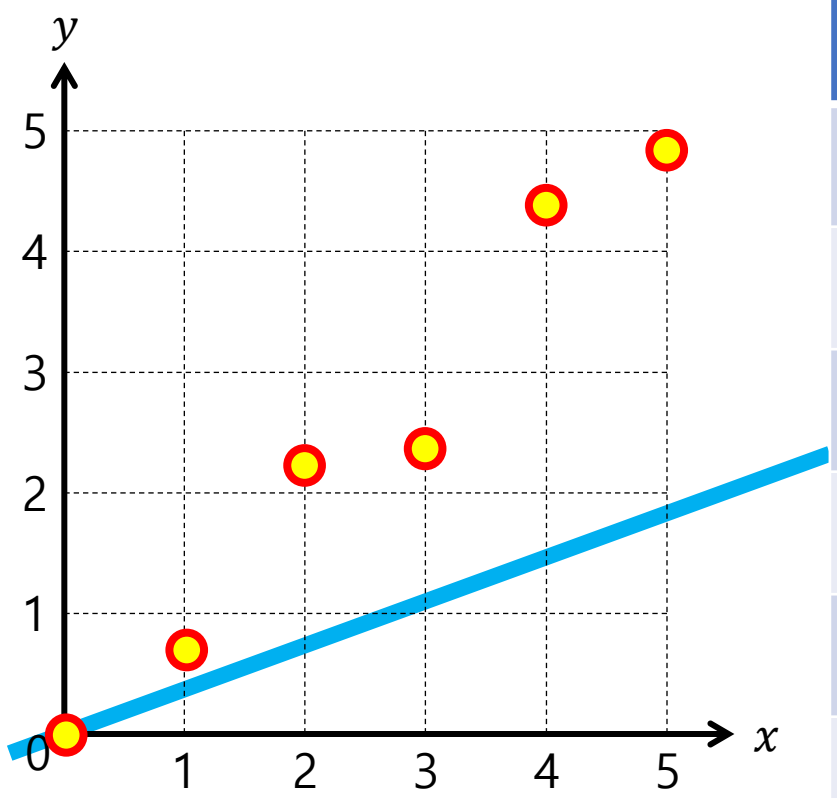
$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.2 - \alpha\left(\frac{1}{6}\cdot(-2)\cdot\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

$$= 0.2 - 0.01\left(\frac{1}{6}\cdot(-2)\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

신박AI

새 w는 0.343이 됩니다



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

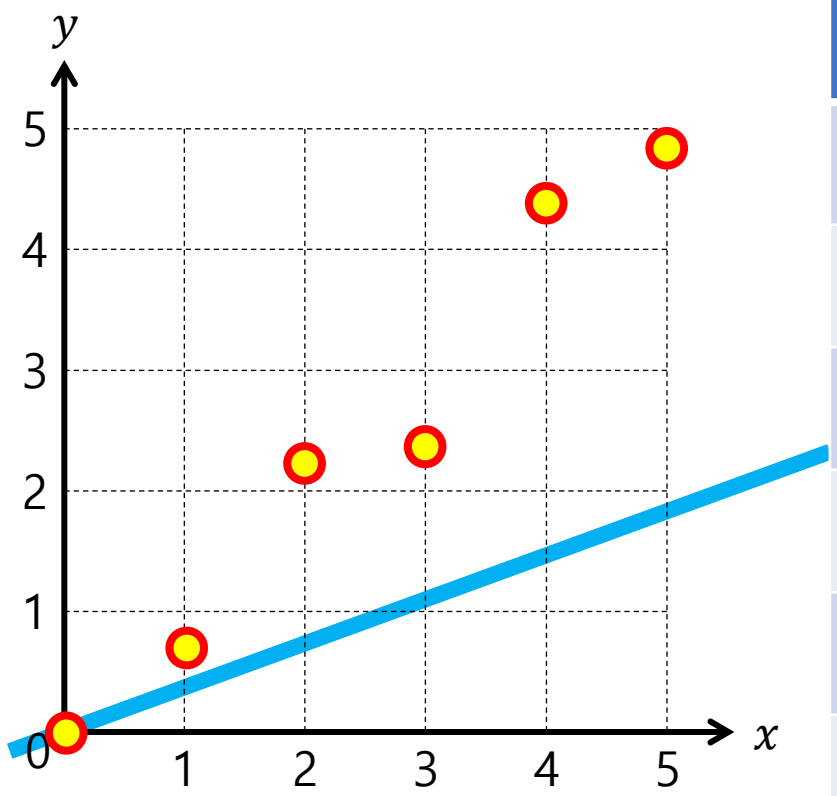
$$= 0.2 - \alpha\left(\frac{1}{6}\cdot(-2)\cdot\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

$$= 0.2 - 0.01\left(\frac{1}{6}\cdot(-2)\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

$$= 0.343$$

신박AI

자 여기까지만 보면 여느 경사하강법과 다를 바가 없어보입니다



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

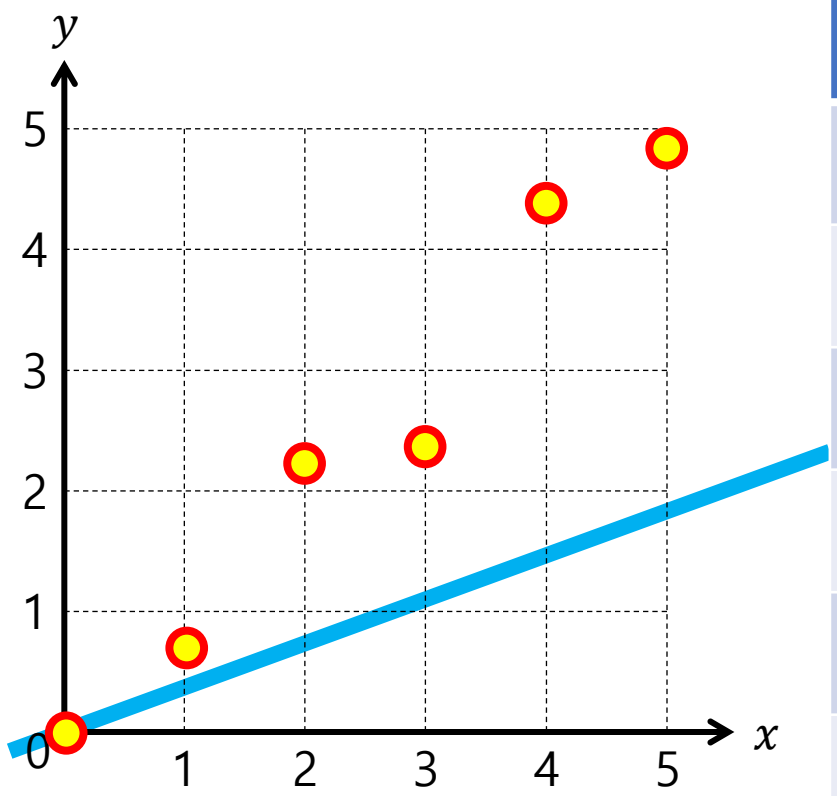
$$= 0.2 - \alpha\left(\frac{1}{6}\cdot(-2)\cdot\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

$$= 0.2 - 0.01\left(\frac{1}{6}\cdot(-2)\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

$$= 0.343$$

신박AI

그러나 우리가 오늘 주목할 부분은 알고리즘이 아니라 어떻게 데이터를 계산에 사용하는가 입니다.



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

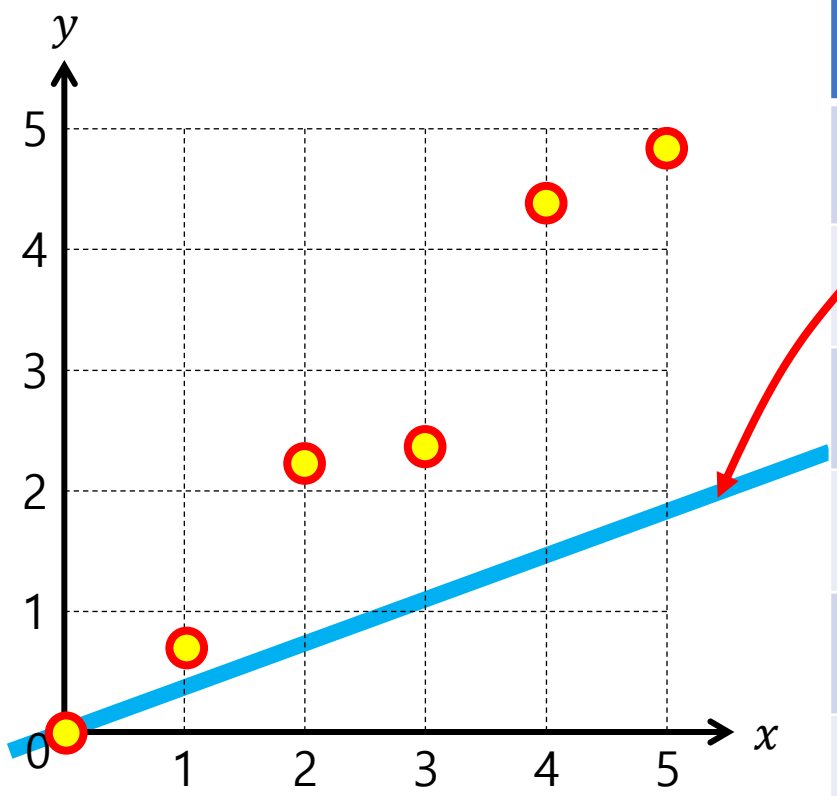
$$= 0.2 - \alpha\left(\frac{1}{6}\cdot(-2)\cdot\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

$$= 0.2 - 0.01\left(\frac{1}{6}\cdot(-2)\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

$$= 0.343$$

신박AI

즉, 기울기를 '한번' 계산하는데 모든 데이터를 다 사용하였다는 점에 유의해 주시기 바랍니다.



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

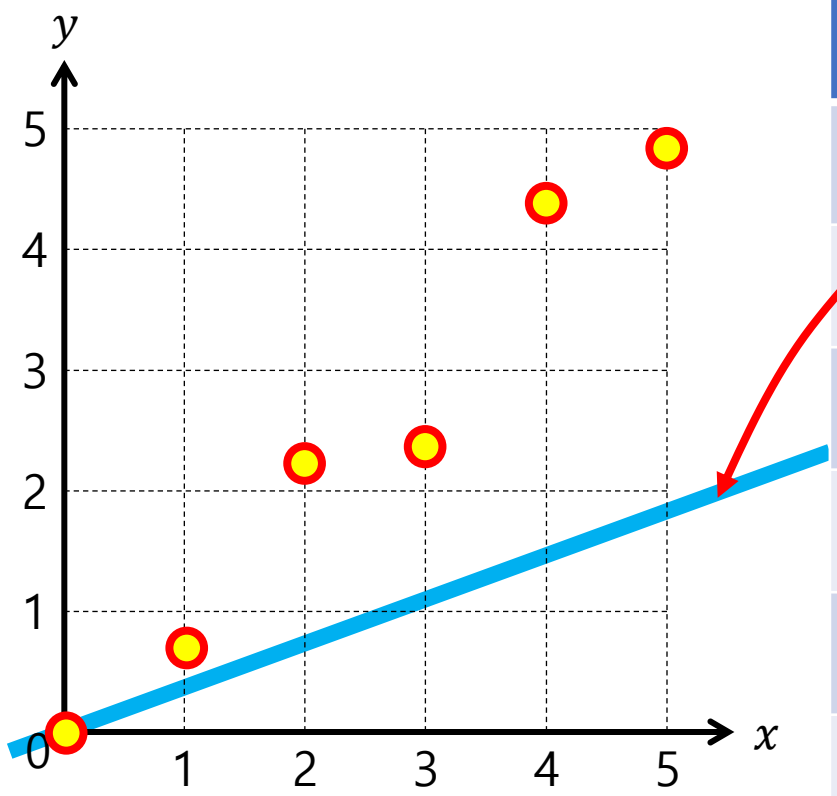
$$= 0.2 - \alpha\left(\frac{1}{6}\cdot(-2)\cdot\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

$$= 0.2 - 0.01\left(\frac{1}{6}\cdot(-2)\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

$$= 0.343$$

신박AI

이렇게 모든 데이터를 다 사용하여 신경망의 가중치를 한 번 업데이트 하는 것을 배치 경사하강법 Batch gradient descent라고 합니다



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

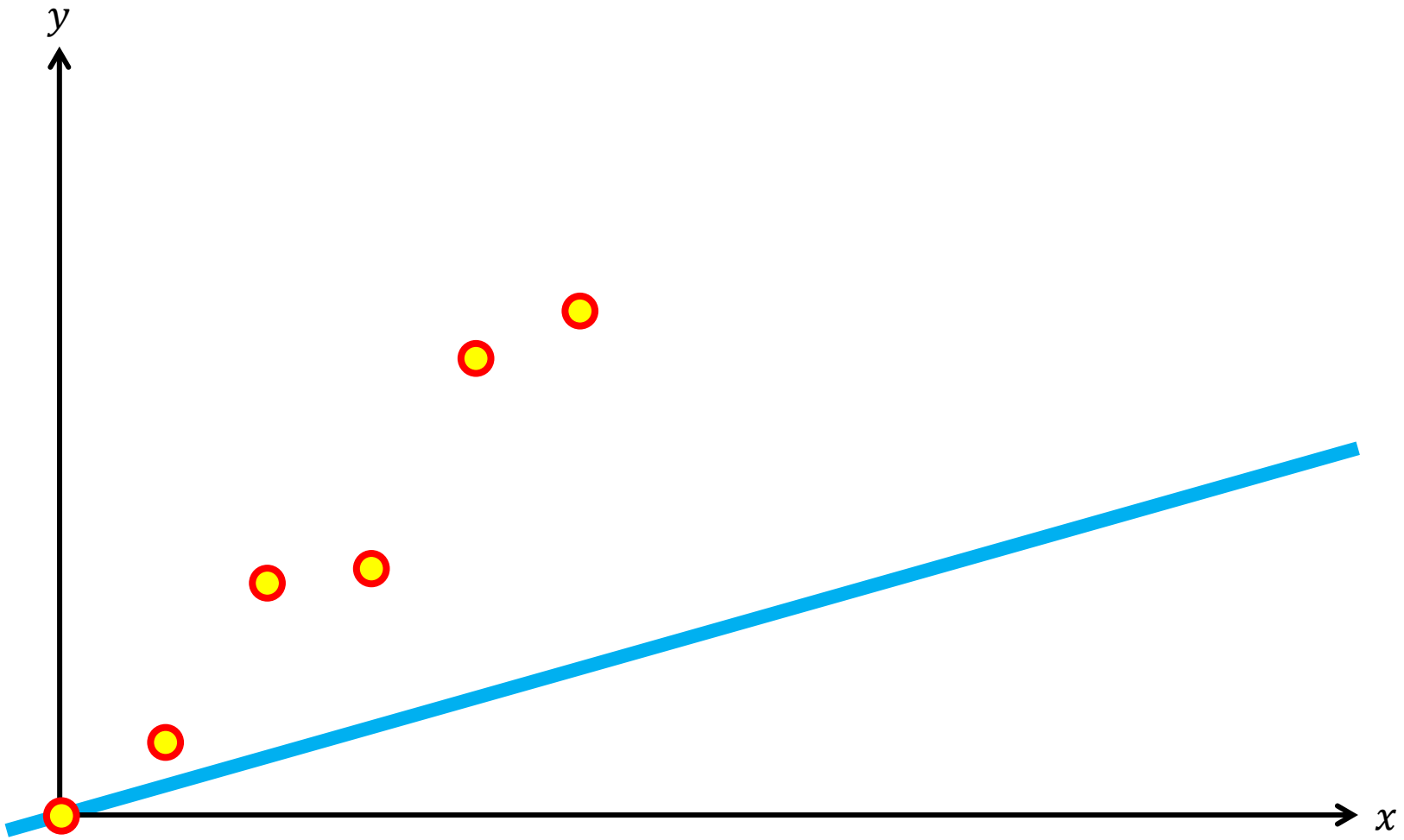
$$= 0.2 - \alpha\left(\frac{1}{6}\cdot(-2)\cdot\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

$$= 0.2 - 0.01\left(\frac{1}{6}\cdot(-2)\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

$$= 0.343$$

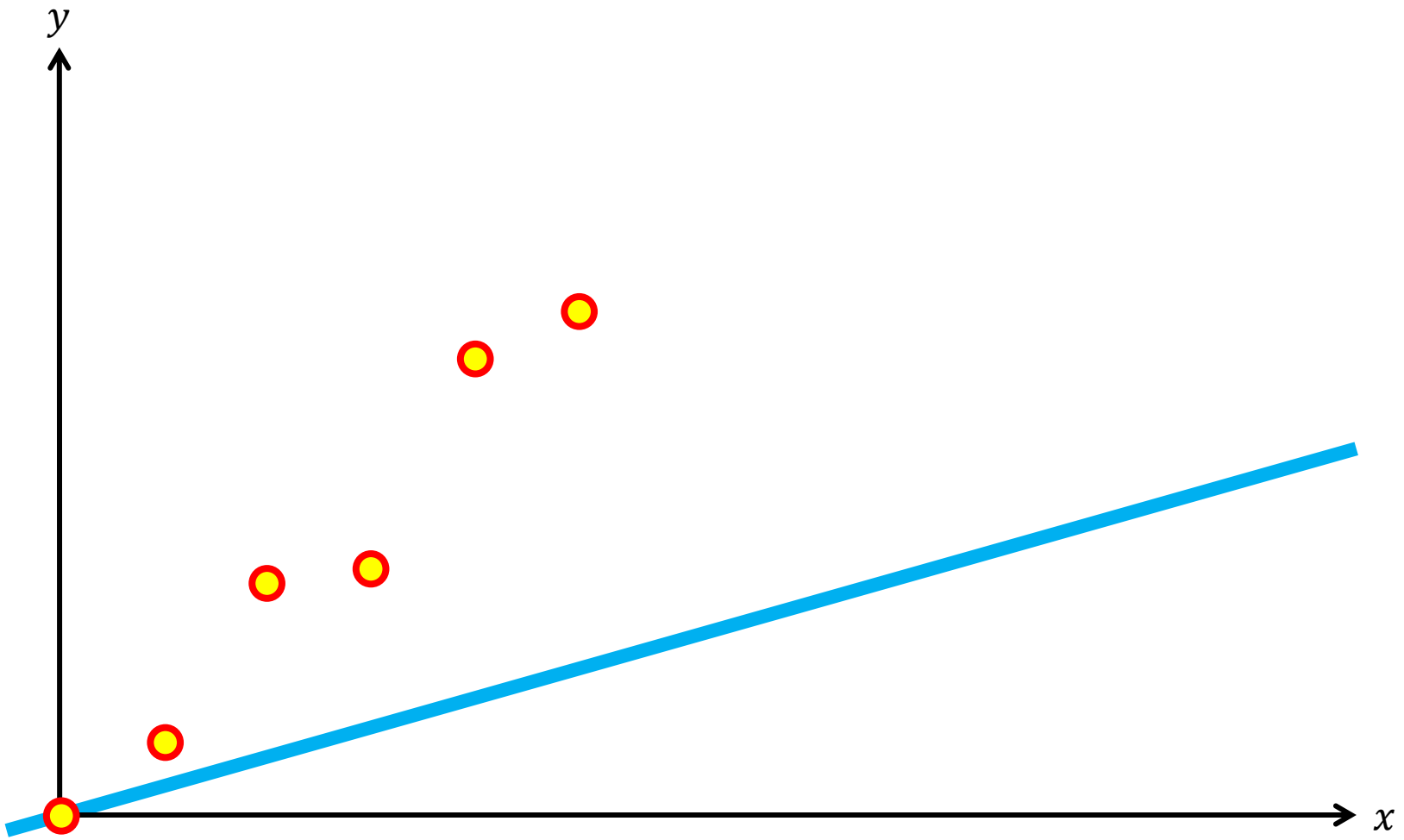
신박AI

보신바와 같이 배치 경사하강법은 한번의 가중치 계산에 모든 데이터들을 사용한다는 점에서



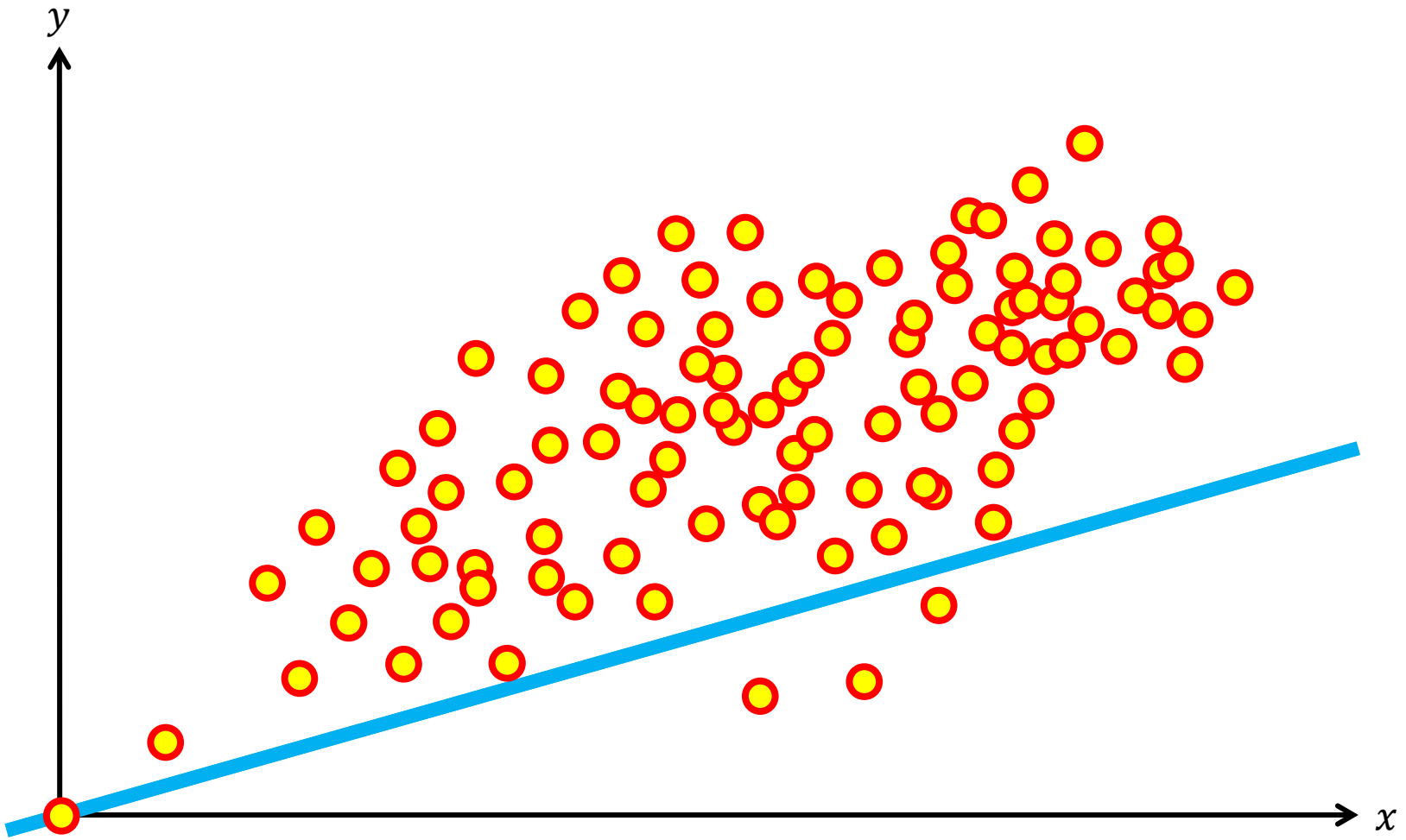
신박AI

상당히 안정적으로 가중치를 업데이트 할 수 있다는 장점이 있습니다



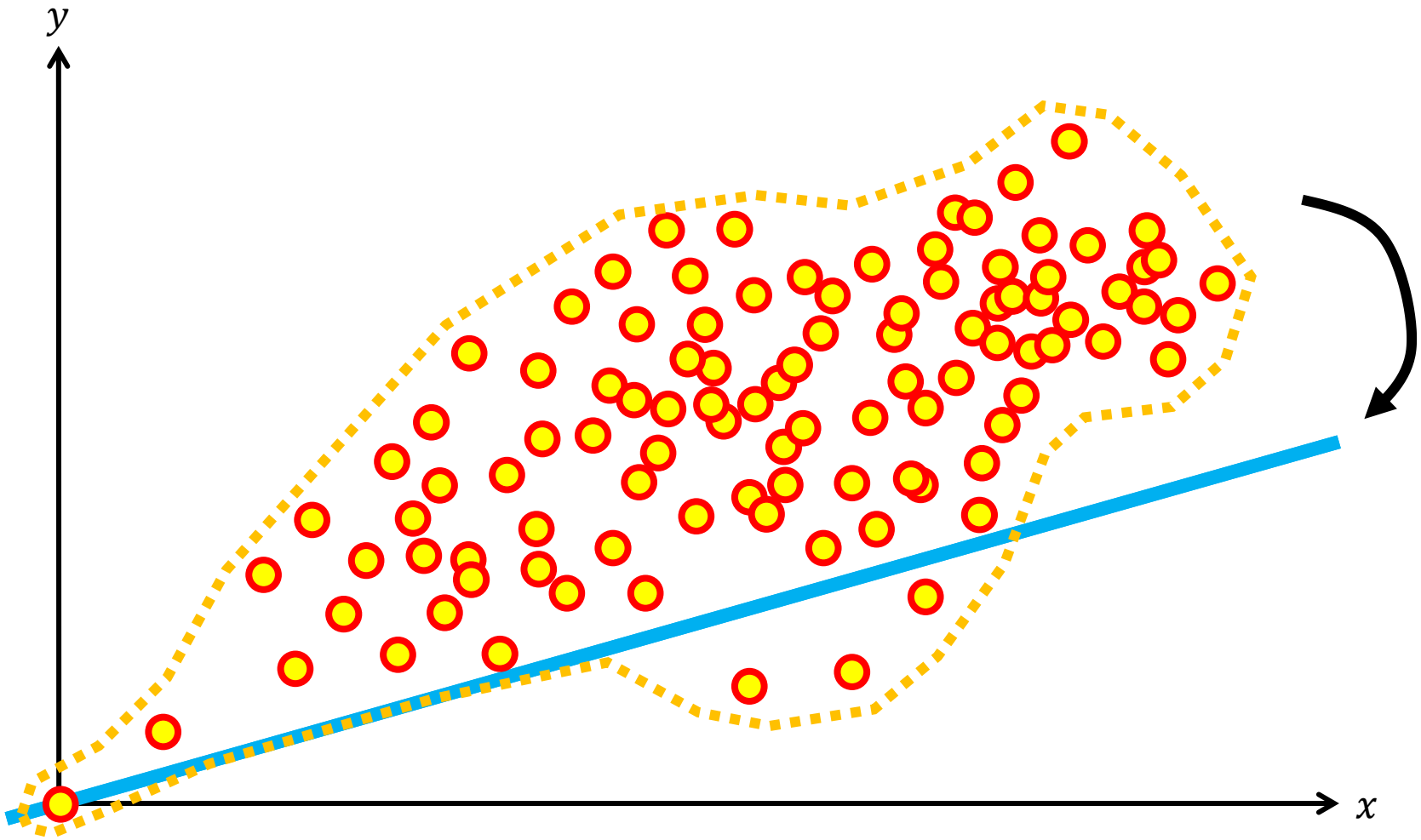
신박AI

하지만 데이터가 이렇게 많을 경우라면 어떨까요?



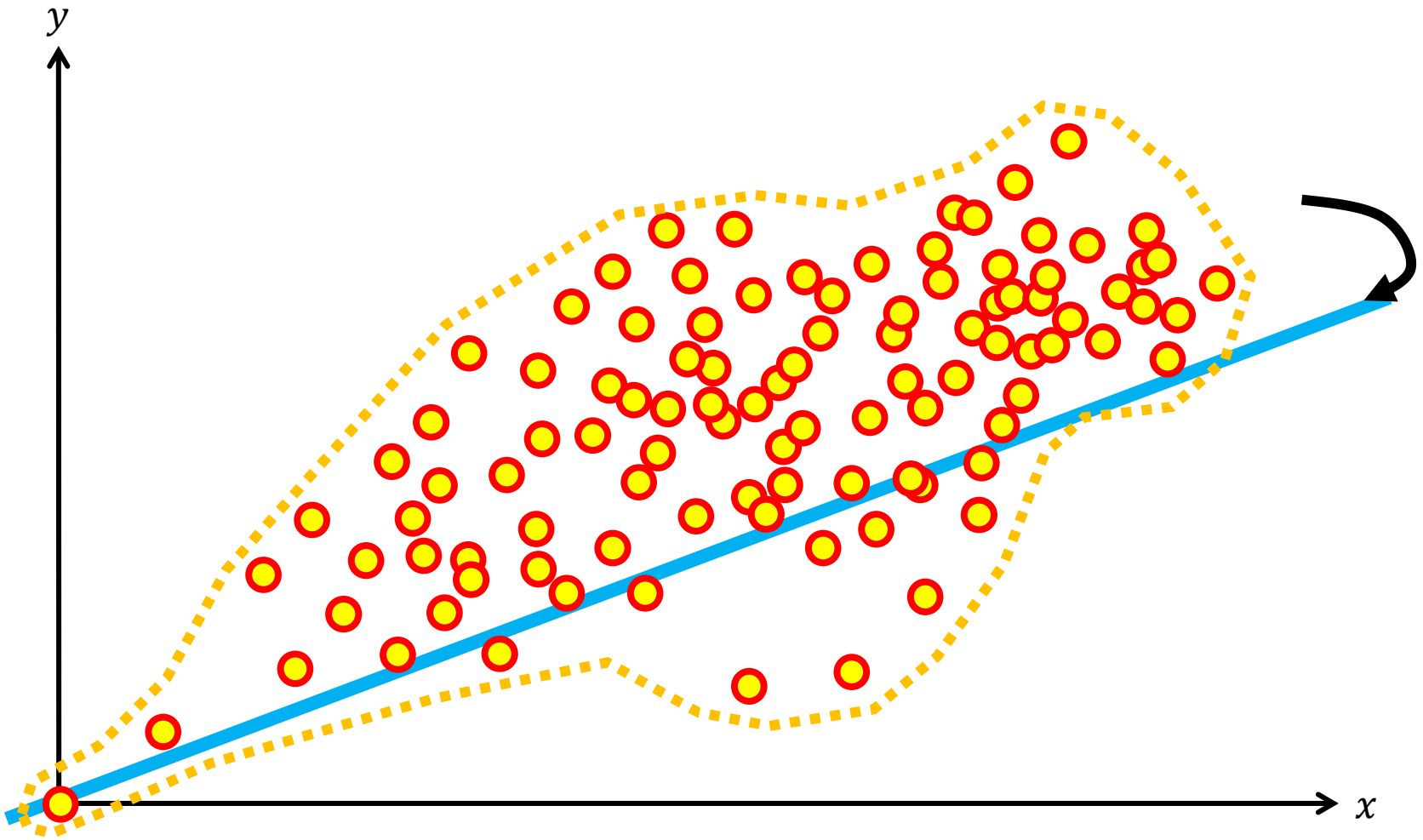
신박AI

가중치 한번 업데이트 하기 위해서 모든 데이터들의 에러를 계산해야 한다는 것은 상당히 비효율적으로 보입니다



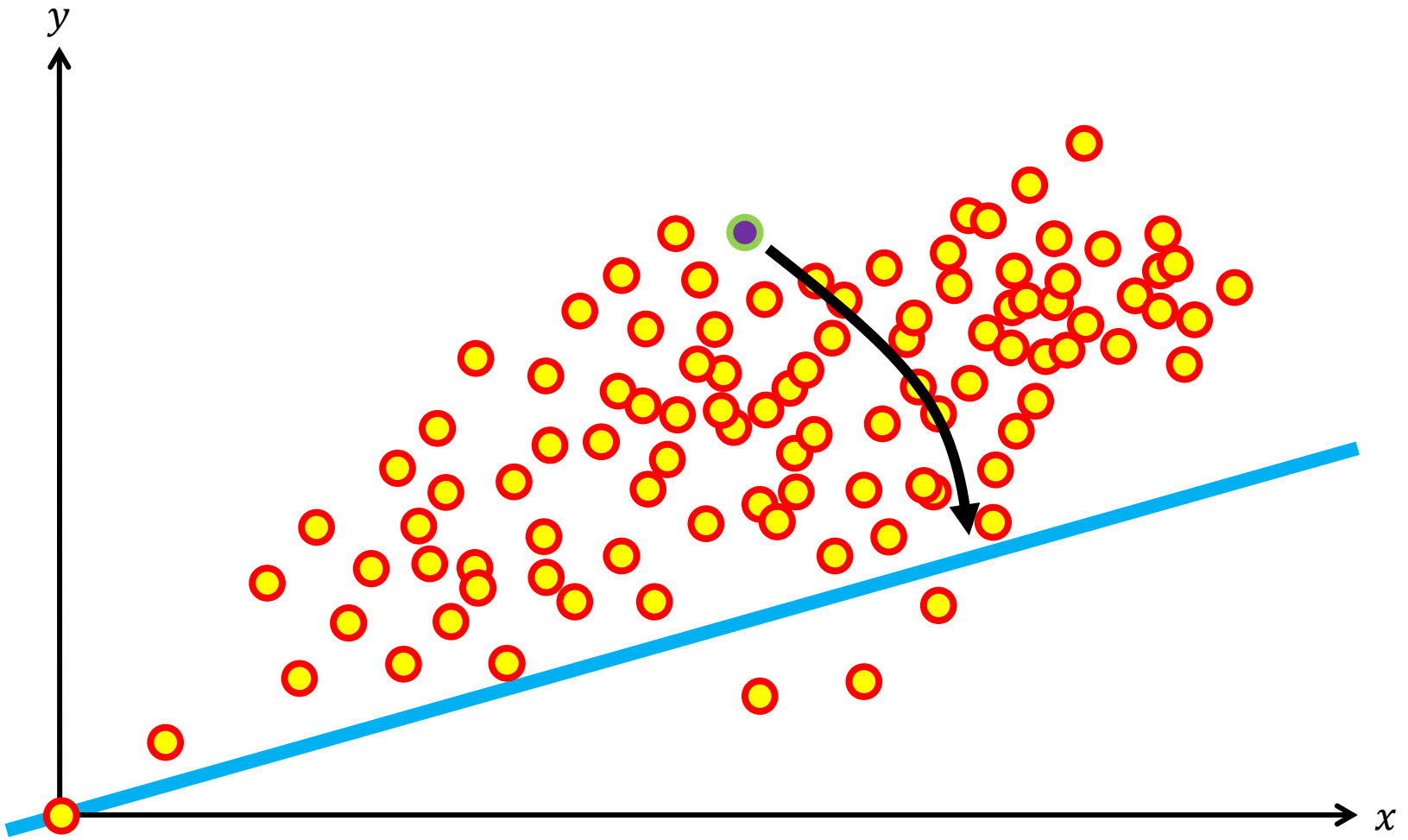
신박AI

가중치 한번 업데이트 하기 위해서 모든 데이터들의 에러를 계산해야 한다는 것은 상당히 비효율적으로 보입니다



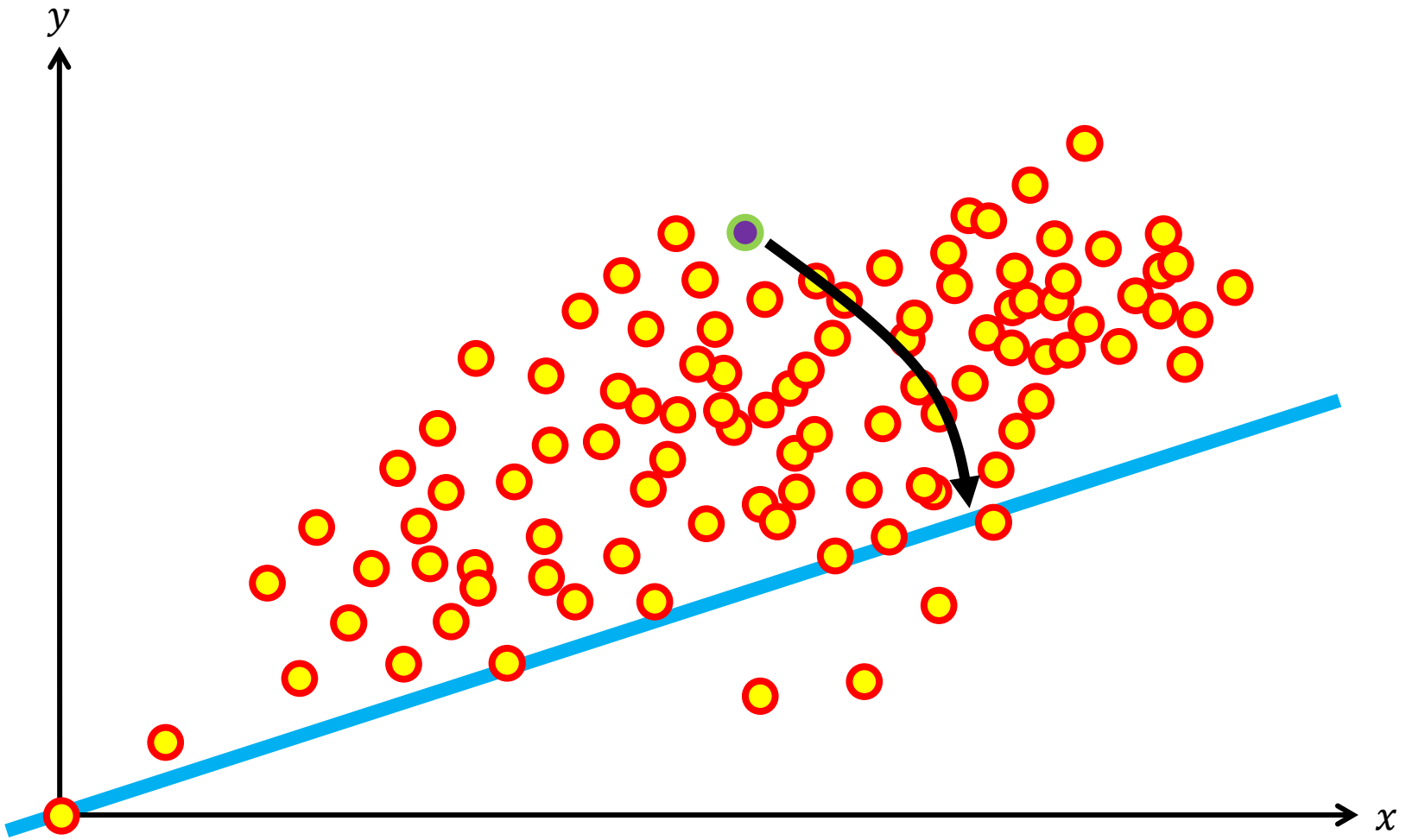
신박AI

그러지 말고, 이렇게 하나의 데이터에서 에러를 계산하여



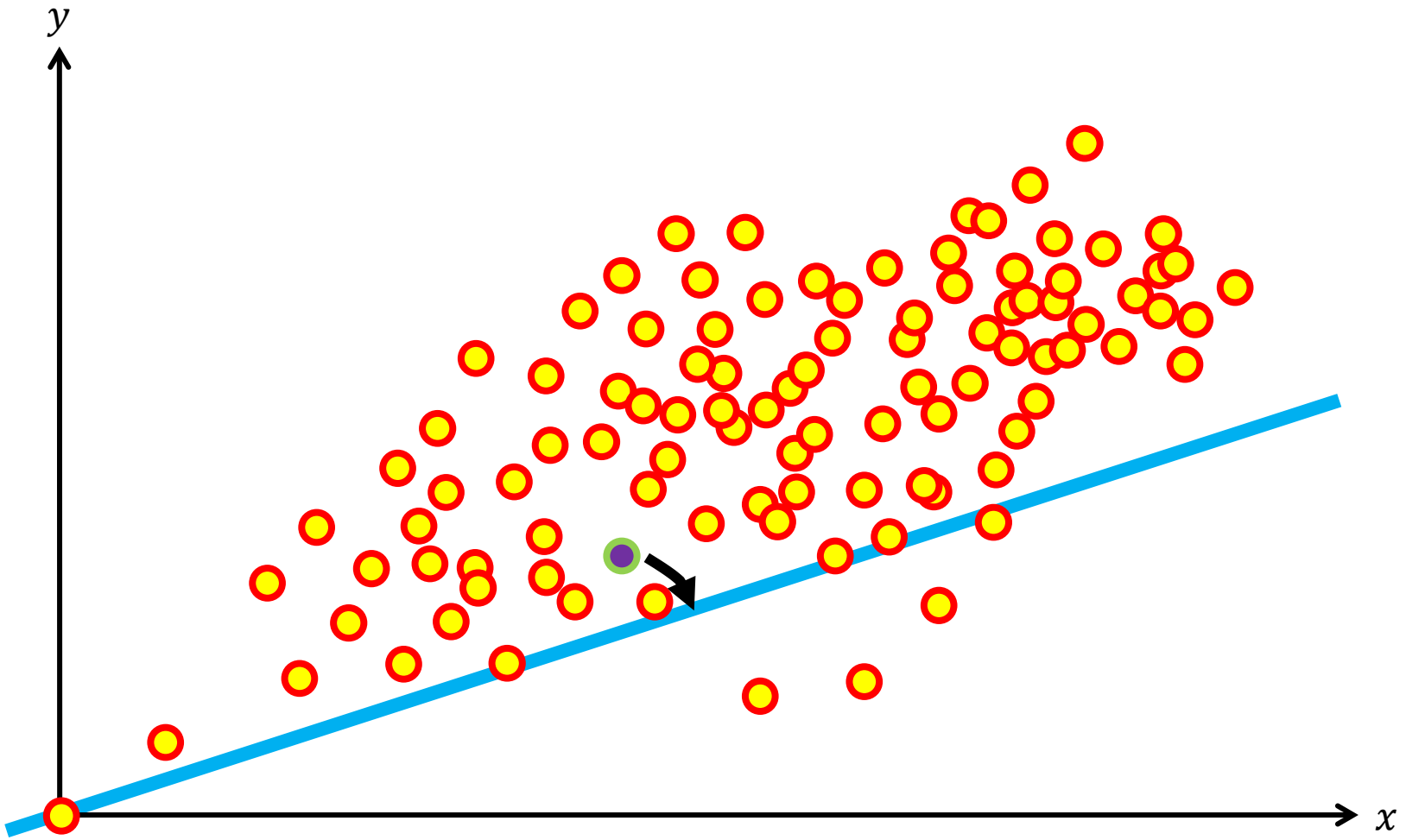
신박AI

가중치를 업데이트 하고,



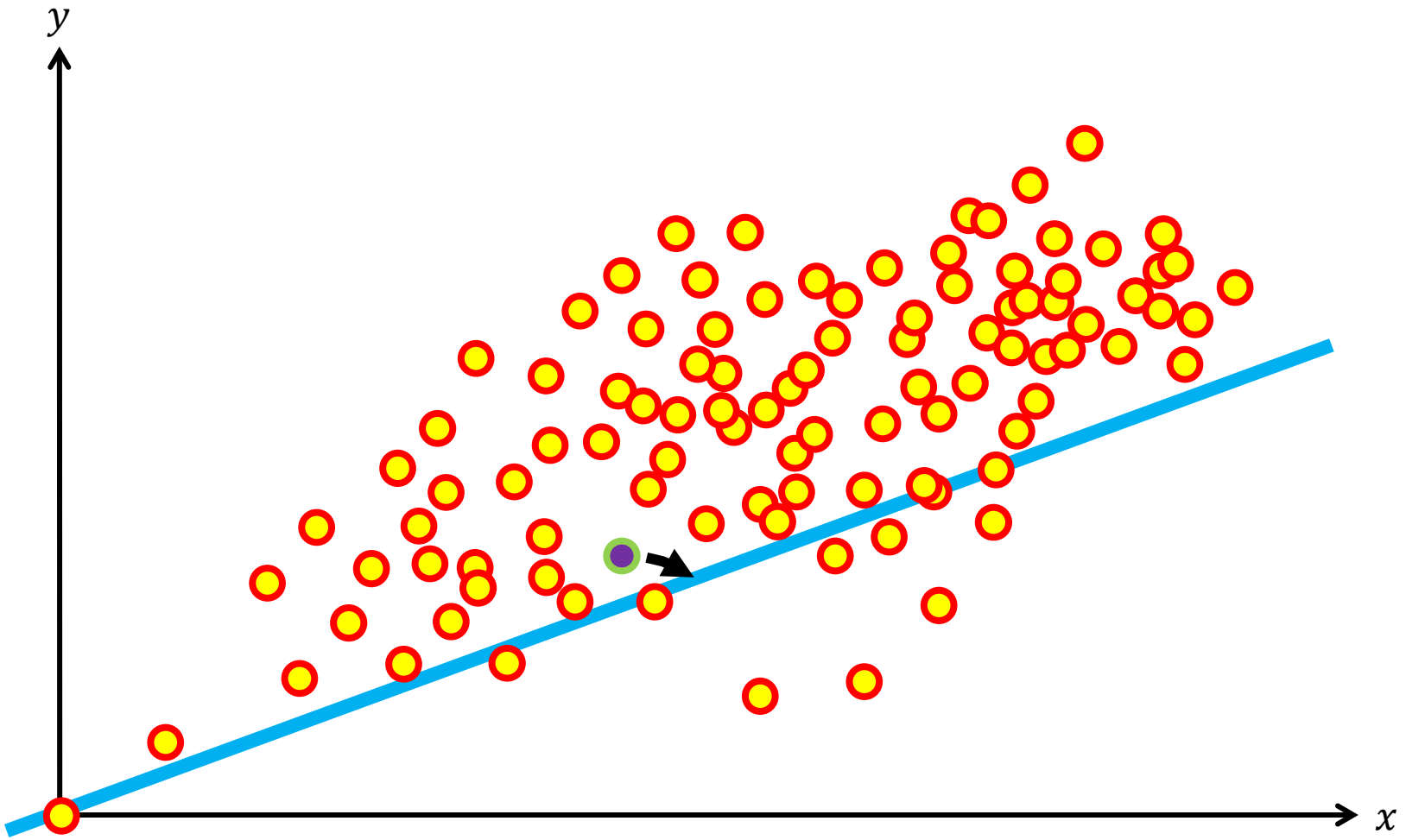
신박AI

또 다른 하나의 데이터에서 에러를 계산하여



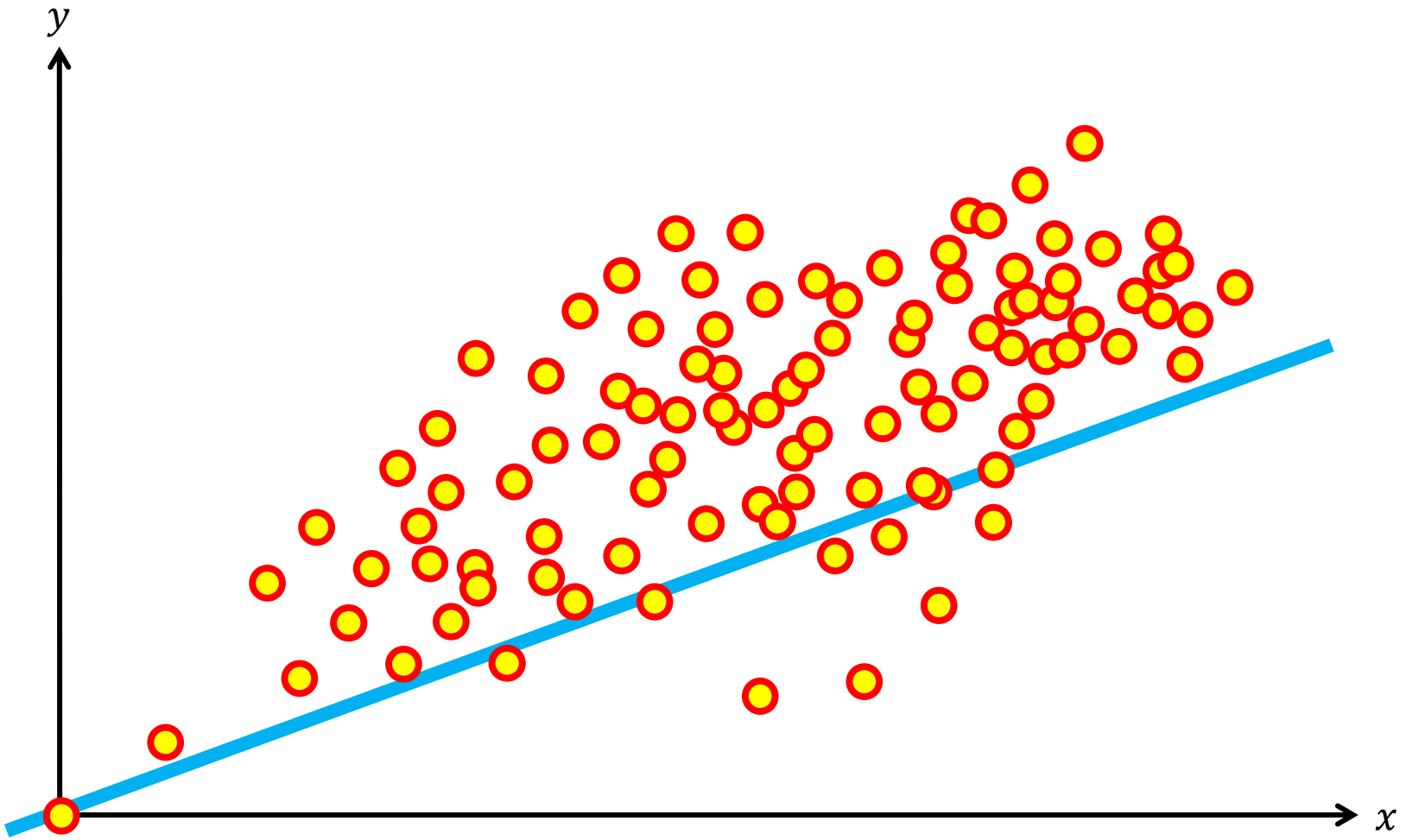
신박AI

가중치를 업데이트 하고,



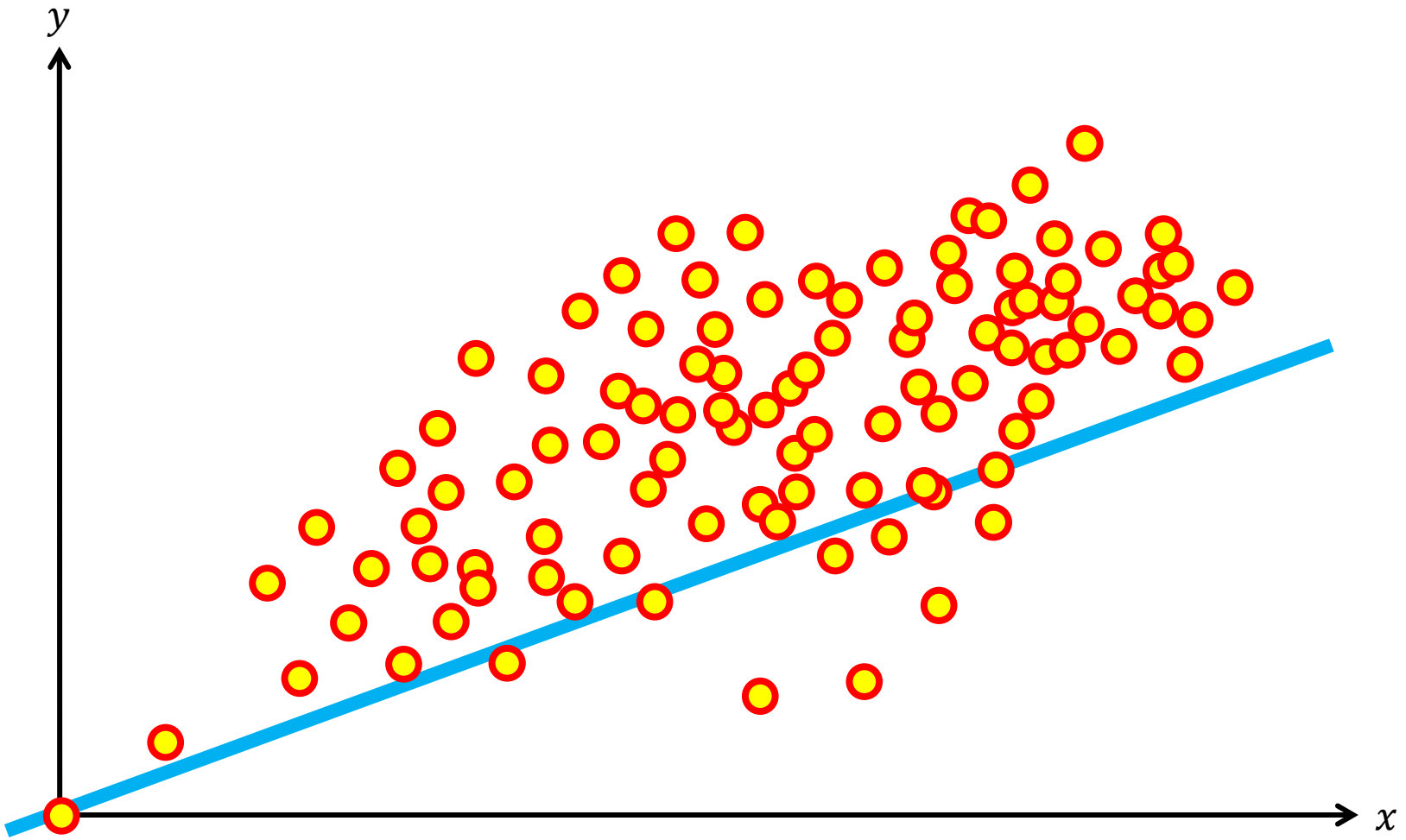
신박AI

이렇게 각각의 데이터 포인트마다 에러를 계산하여 가중치 업데이트 하는 학습 스케줄링을



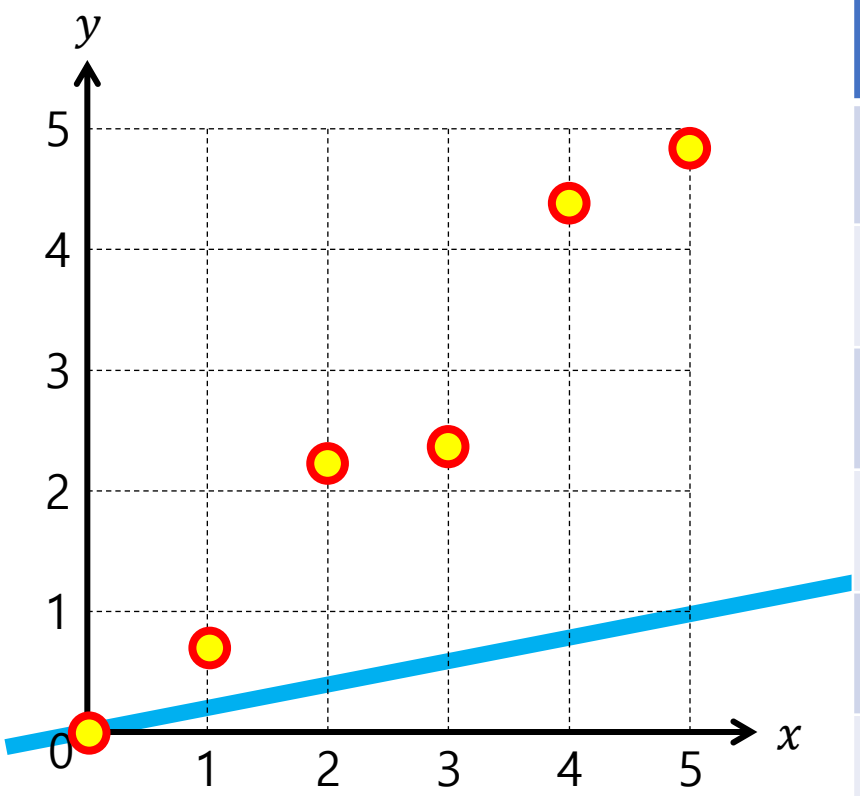
신박AI

확률적 경사하강법, Stochastic gradient descent라고 합니다



신박AI

우리가 아까 보았던 예제로 돌아가서 설명을 드리자면,



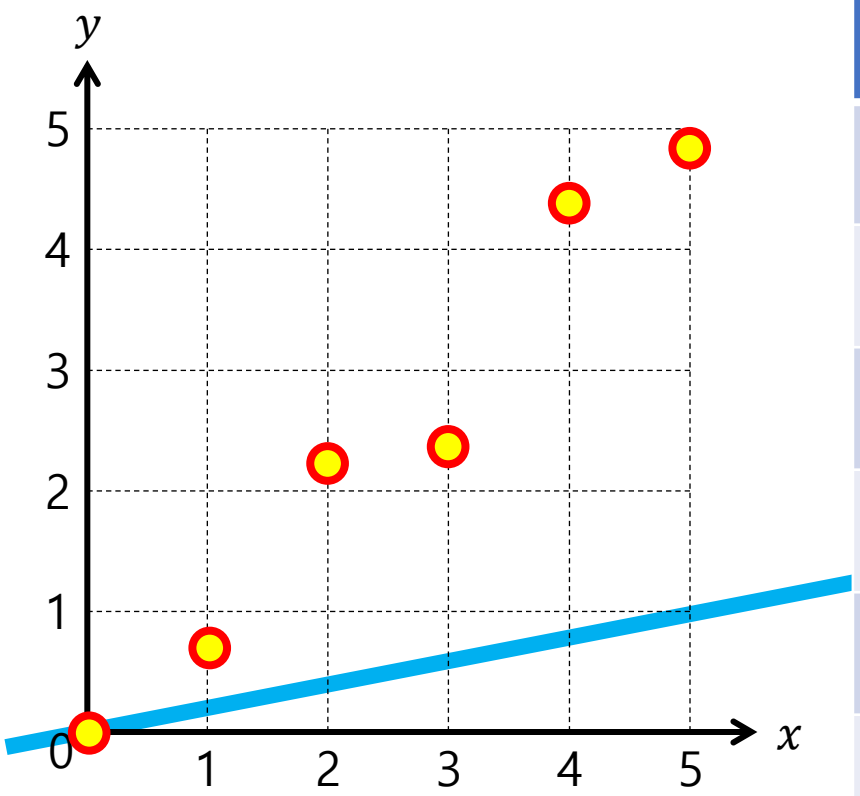
x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

신박AI

여기까지는 배치 경사하강법과 과정이 동일합니다.



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

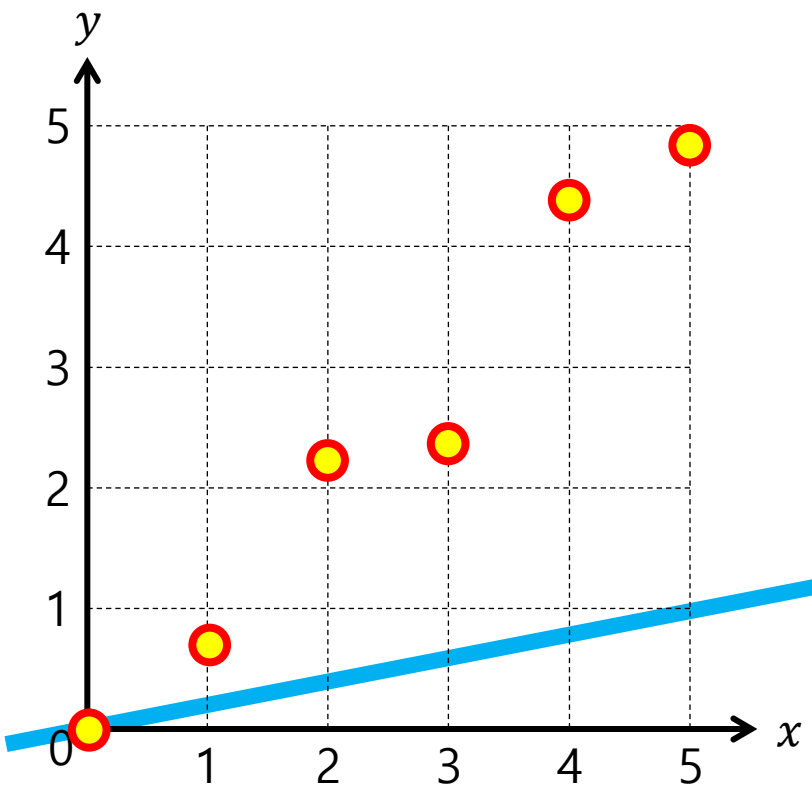
$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

신박AI

그러나 여기서부터는 랜덤으로 하나의 데이터만을 이용해 값을 구합니다.



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

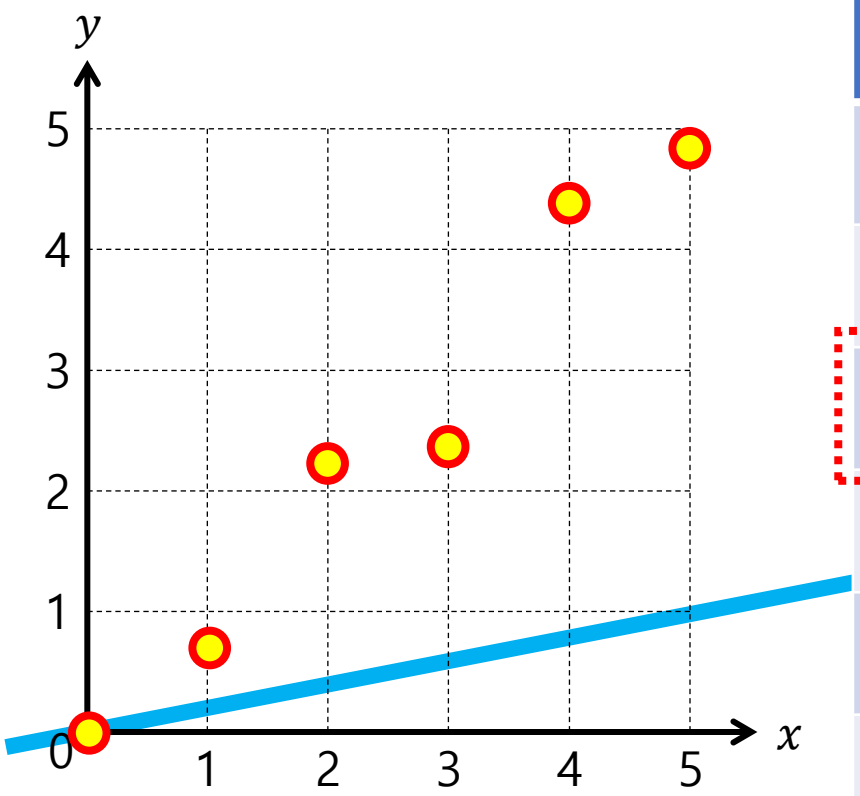
$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.2 - \alpha\left(\frac{1}{1}\cdot(-2)\cdot(2.2 - 0.4)\cdot 2\right)$$

신박AI

새로운 w는 0.272로 업데이트 되었습니다.



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

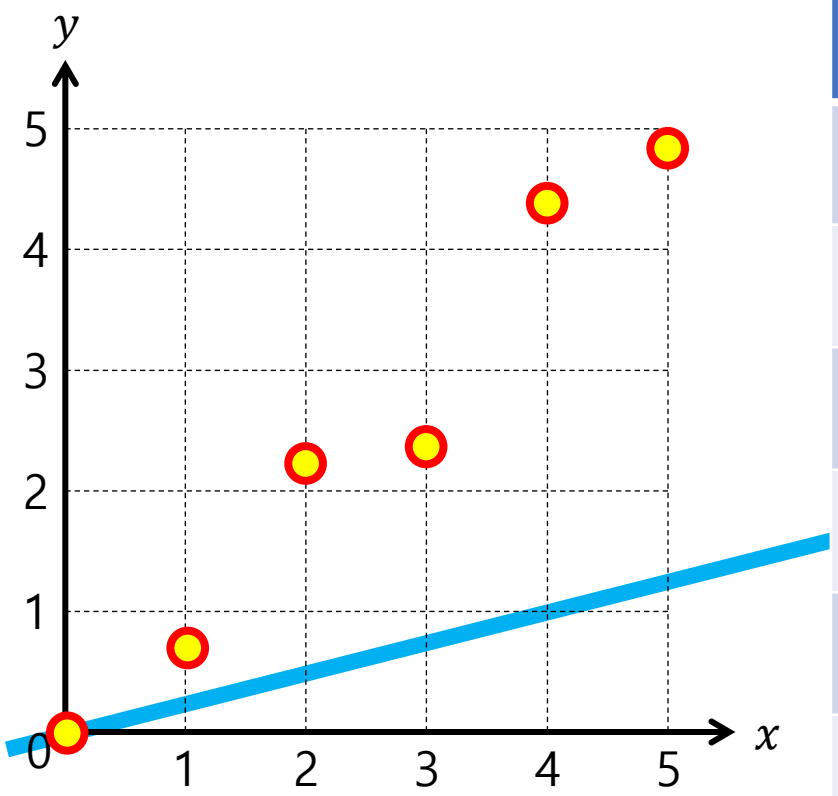
$$= 0.2 - \alpha\left(\frac{1}{1}\cdot(-2)\cdot(2.2 - 0.4)\cdot 2\right)$$

$$= 0.2 - 0.01\left(\frac{1}{1}\cdot(-2)\cdot(2.2 - 0.4)\cdot 2\right)$$

$$= 0.272$$

신박AI

그러면 이번엔 새로운 w를 사용하여 계산을 이어갑니다.



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

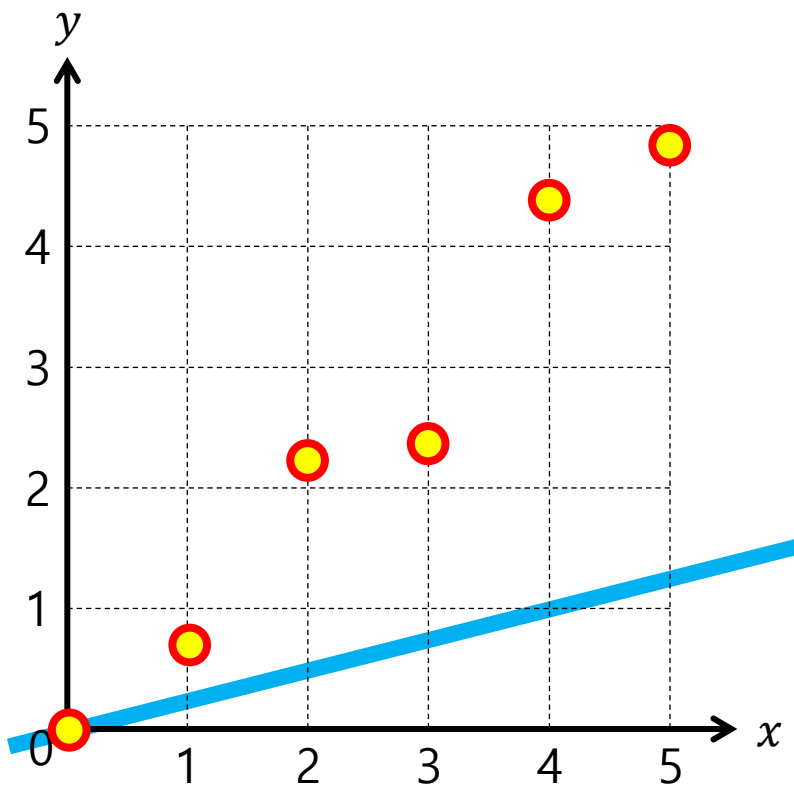
$$= 0.2 - \alpha\left(\frac{1}{1}\cdot(-2)\cdot(2.2 - 0.4)\cdot 2\right)$$

$$= 0.2 - 0.01\left(\frac{1}{1}\cdot(-2)\cdot(2.2 - 0.4)\cdot 2\right)$$

$$= 0.272$$

신박AI

그러면 w에 새로운 값을 대입하고 계산하면



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

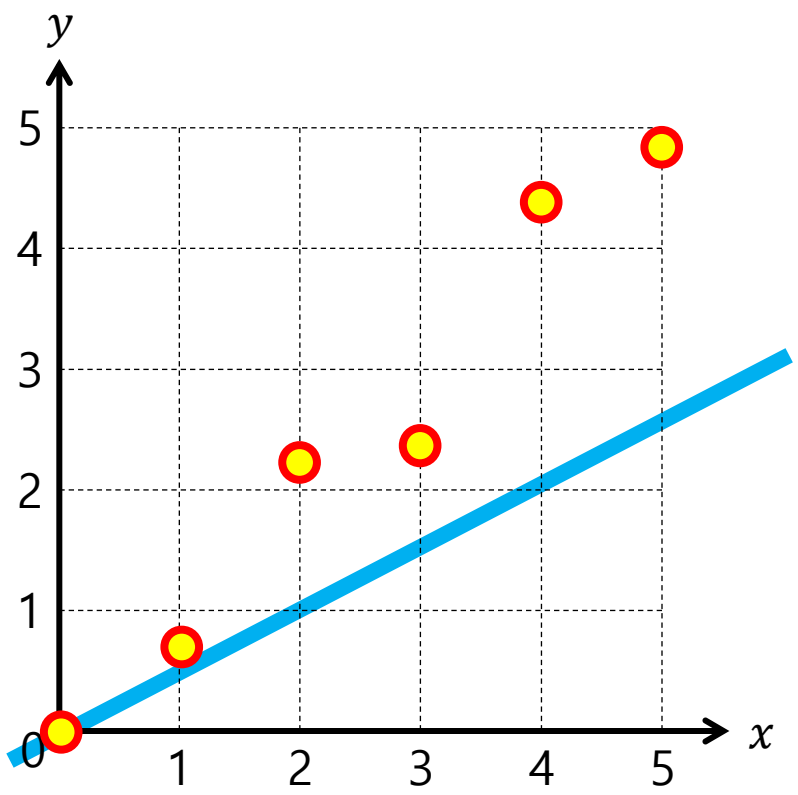
$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.272 - \alpha\left(\frac{1}{1}\cdot(-2)\cdot(4.3 - 0.8)\cdot 4\right)$$

신박AI

또 새로운 w를 계산할 수 있습니다



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

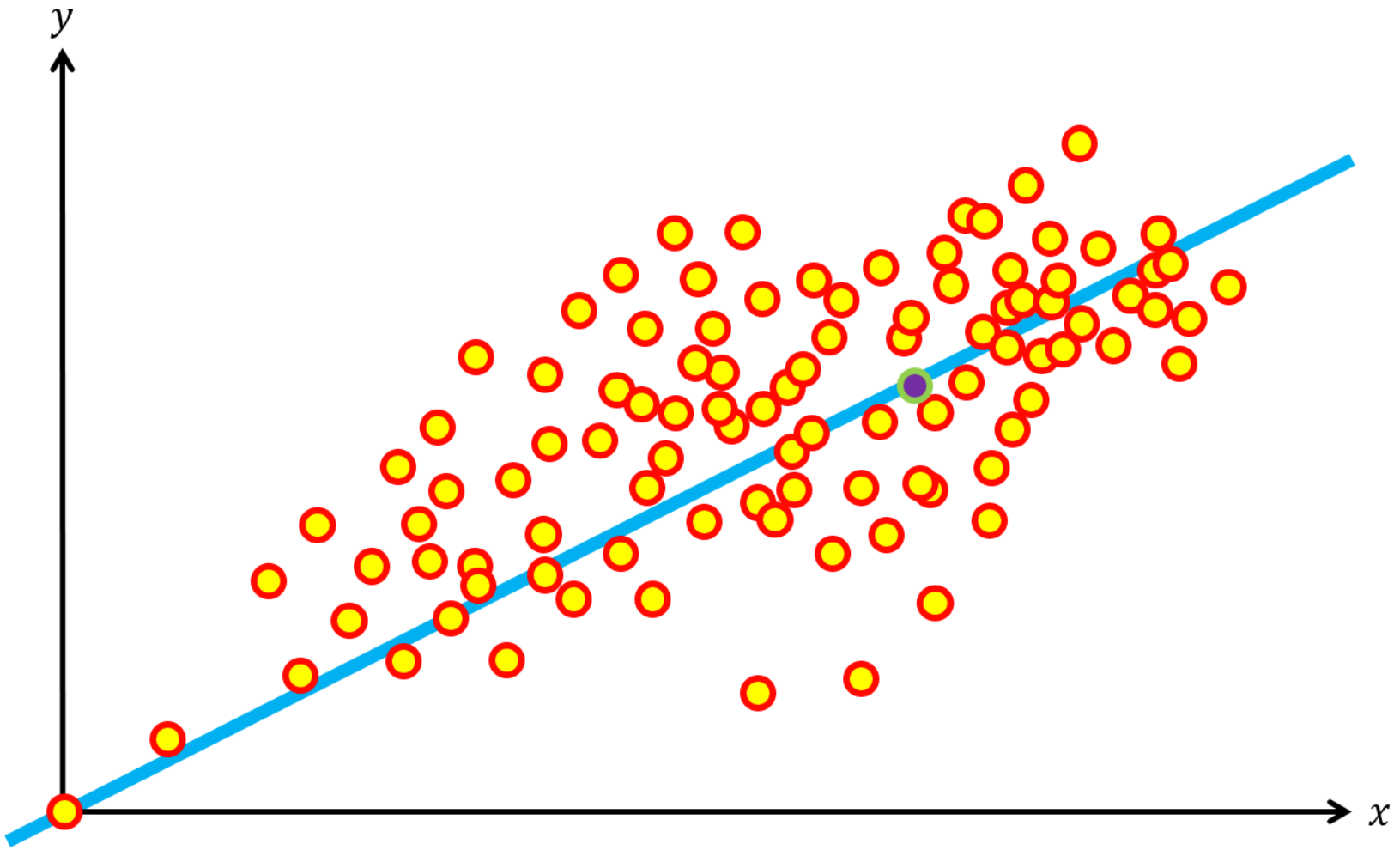
$$= 0.272 - \alpha\left(\frac{1}{1}\cdot(-2)\cdot(4.3 - 0.8)\cdot 4\right)$$

$$= 0.272 - 0.01\left(\frac{1}{1}\cdot(-2)\cdot(4.3 - 0.8)\cdot 4\right)$$

$$= 0.552$$

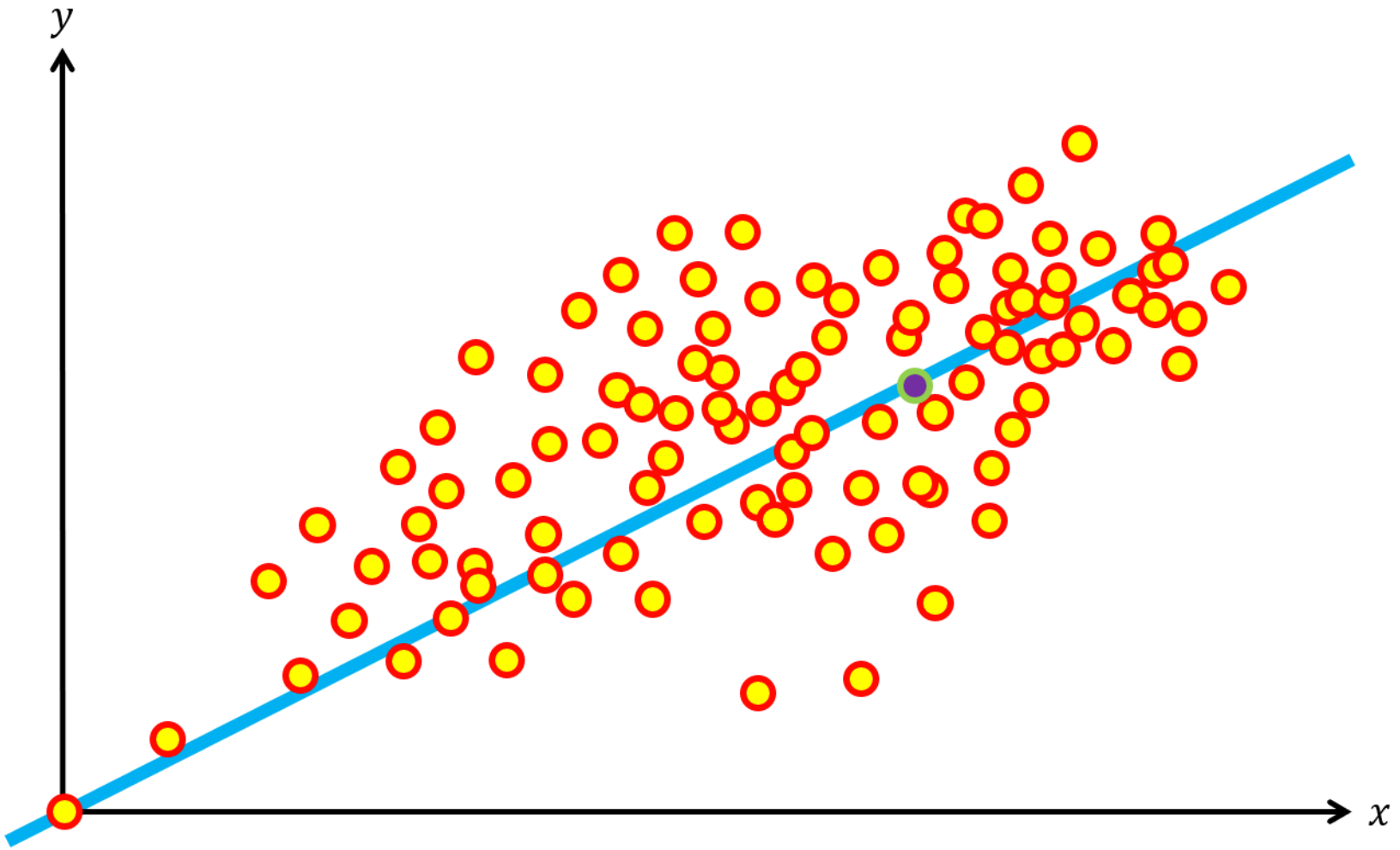
신박AI

이렇게 데이터 하나하나의 에러를 계산하여 가중치 w를 업데이트 하는 것은, 전체를 다 계산 한 뒤 '한번' 가중치 업데이트 하는 것 보다는



신박AI

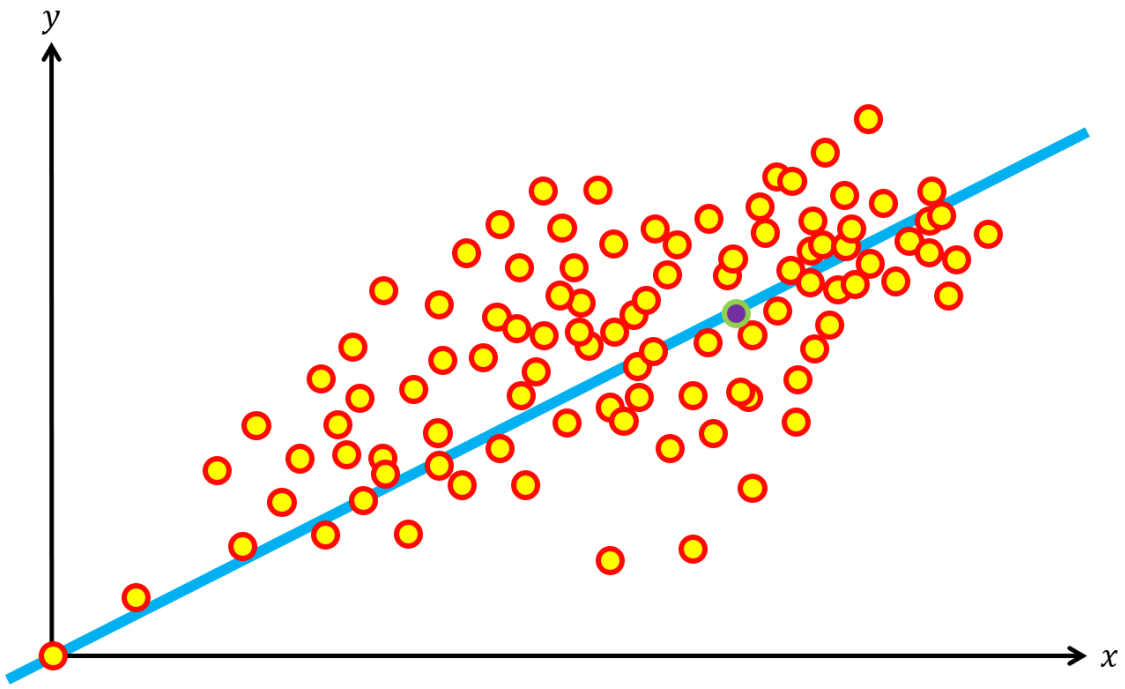
훨씬 효율적인 방법이라고 할 수 있습니다.



신박AI

그러나 확률적 경사하강법이 효율적인 방법이기는 하지만,

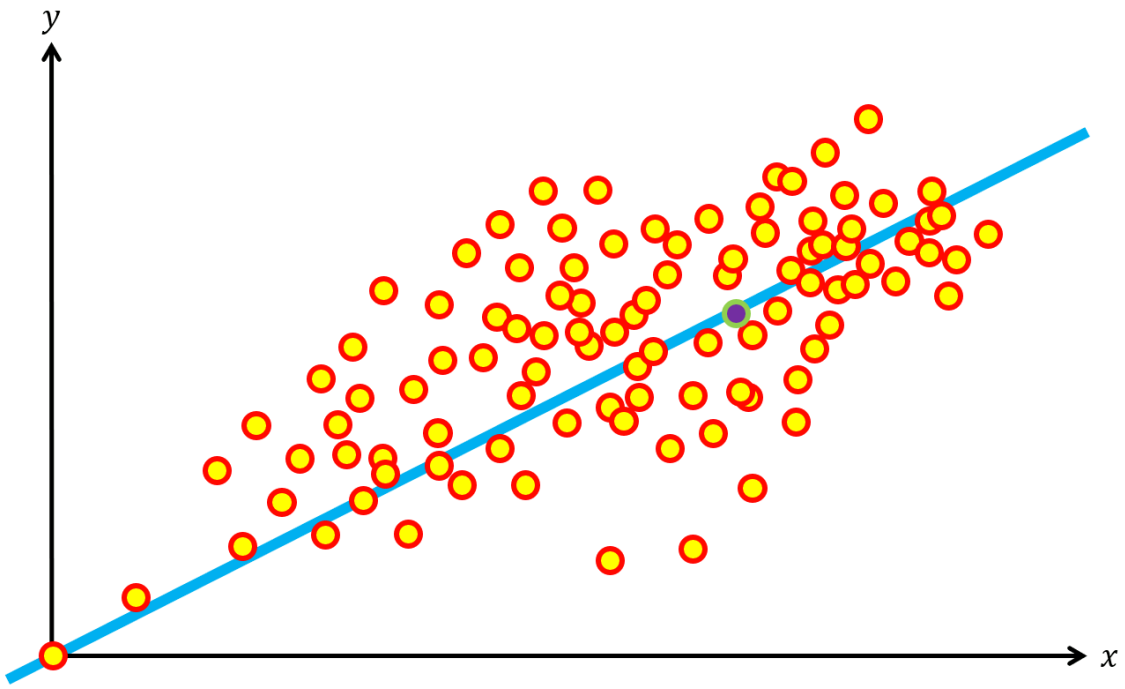
효율



신박AI

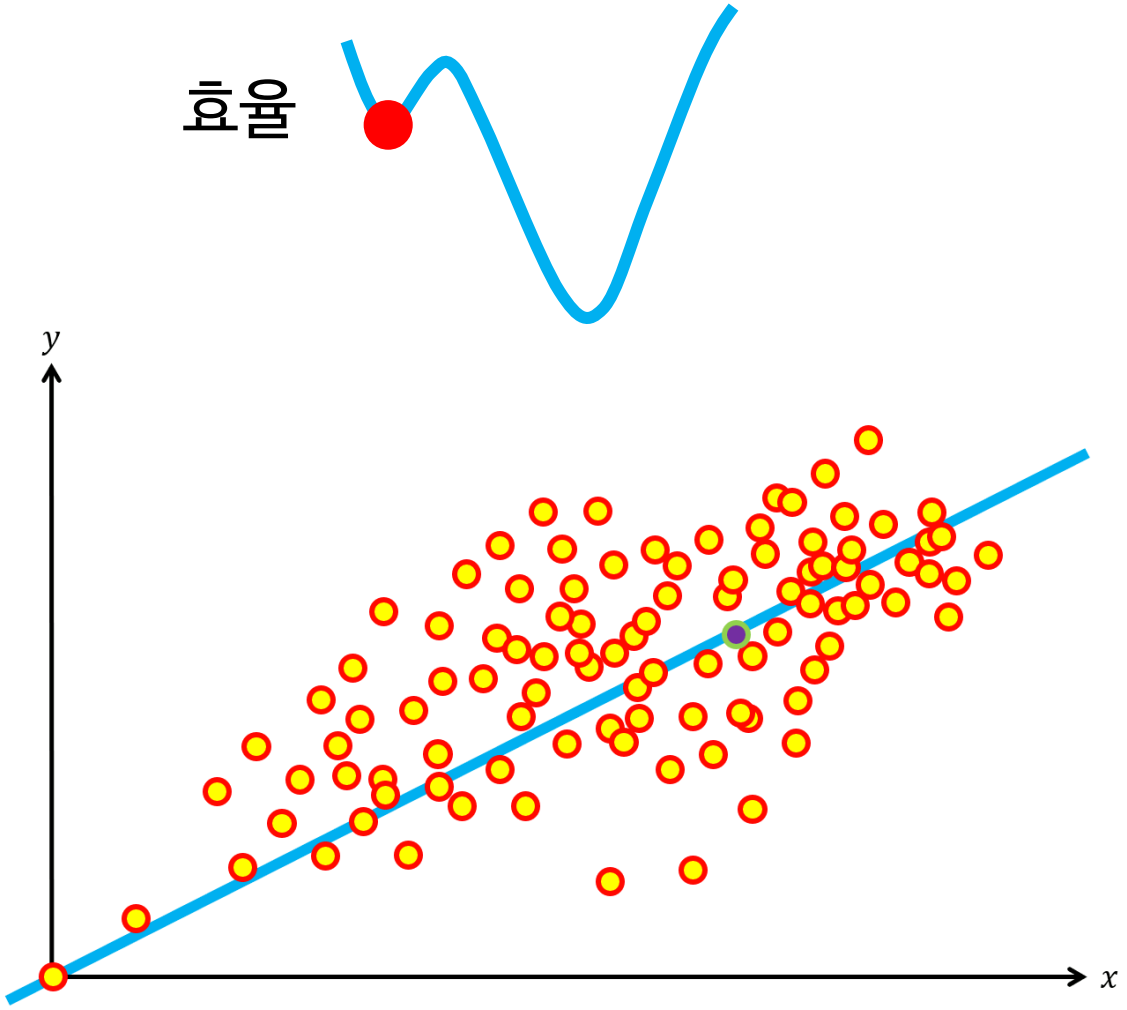
가중치 업데이트 하는데 데이터 하나하나에 영향을 받기 때문에

효율



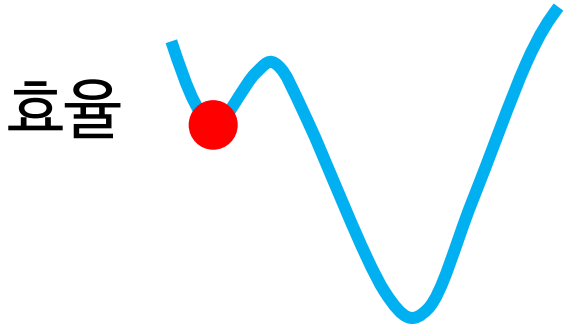
신박AI

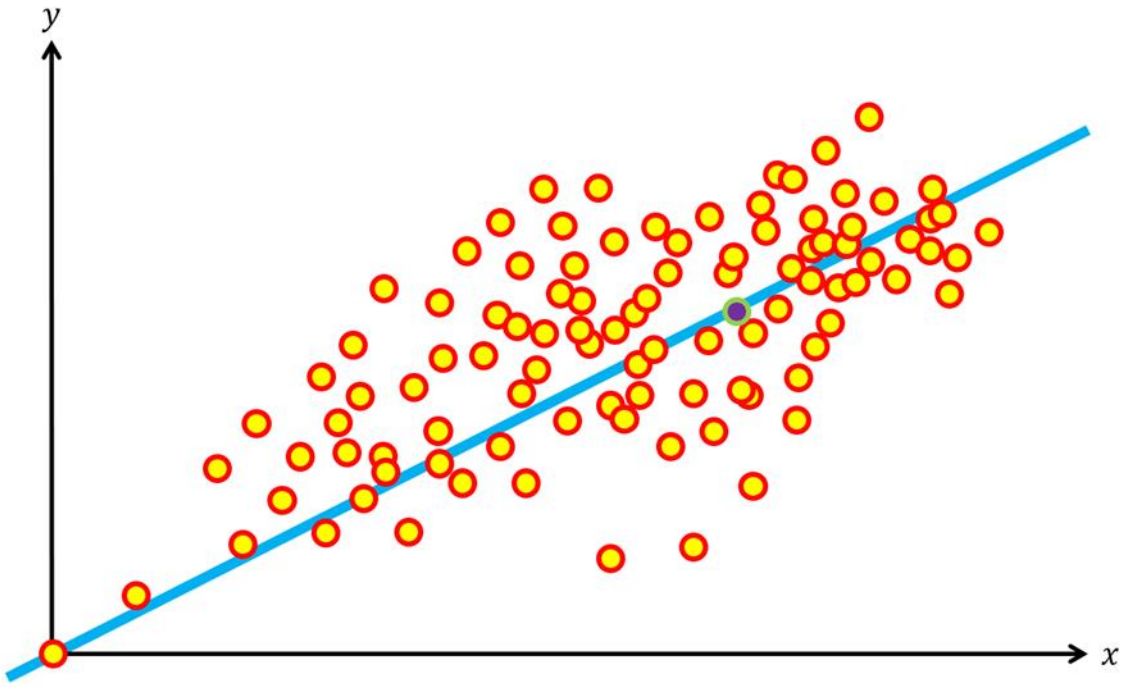
로컬 미니마에 빠질 확률이 높다고 합니다.

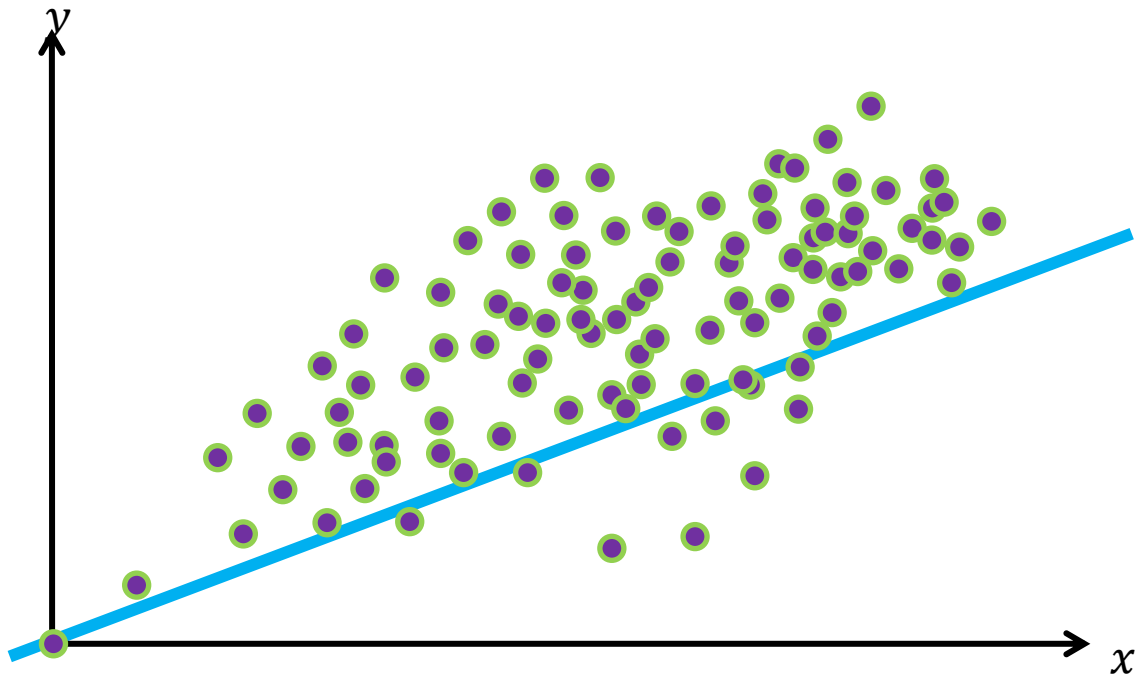


신박AI

그에 비해 배치 경사하강법은 가중치 한번 업데이트 하는데,

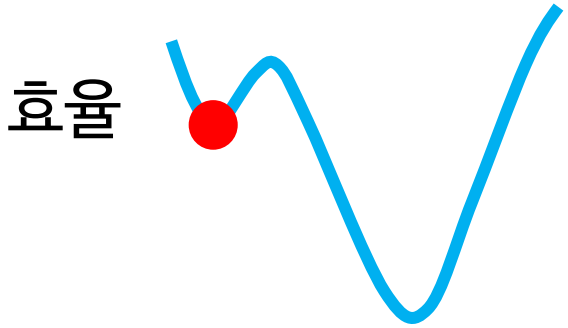


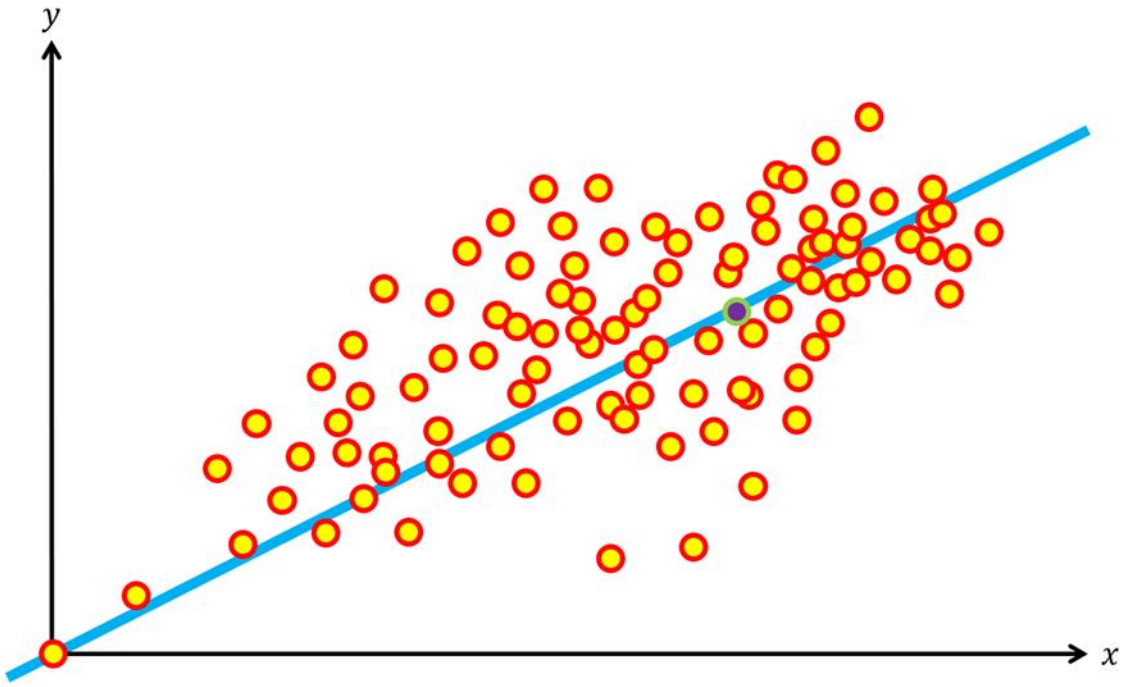


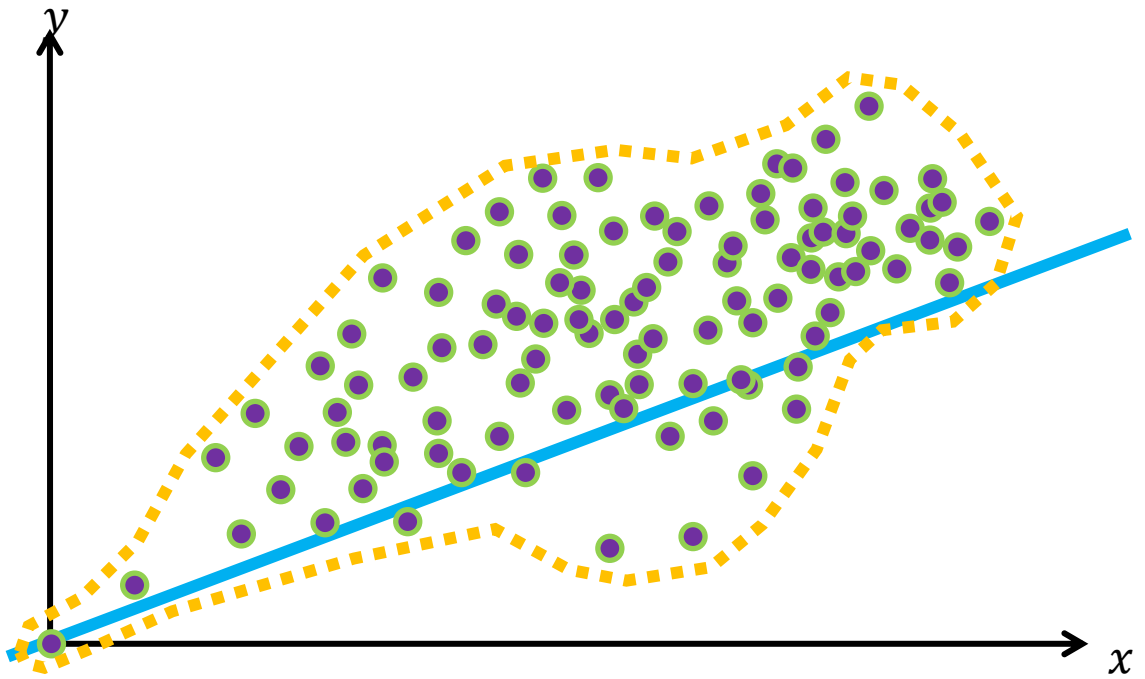


신박AI

모든 데이터의 일일이 고려해서 업데이트하기 때문에



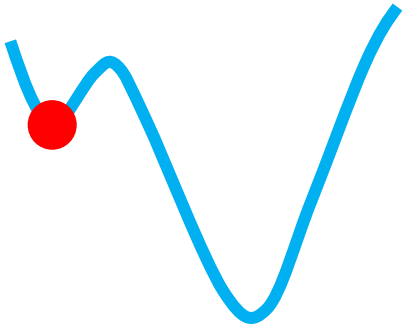




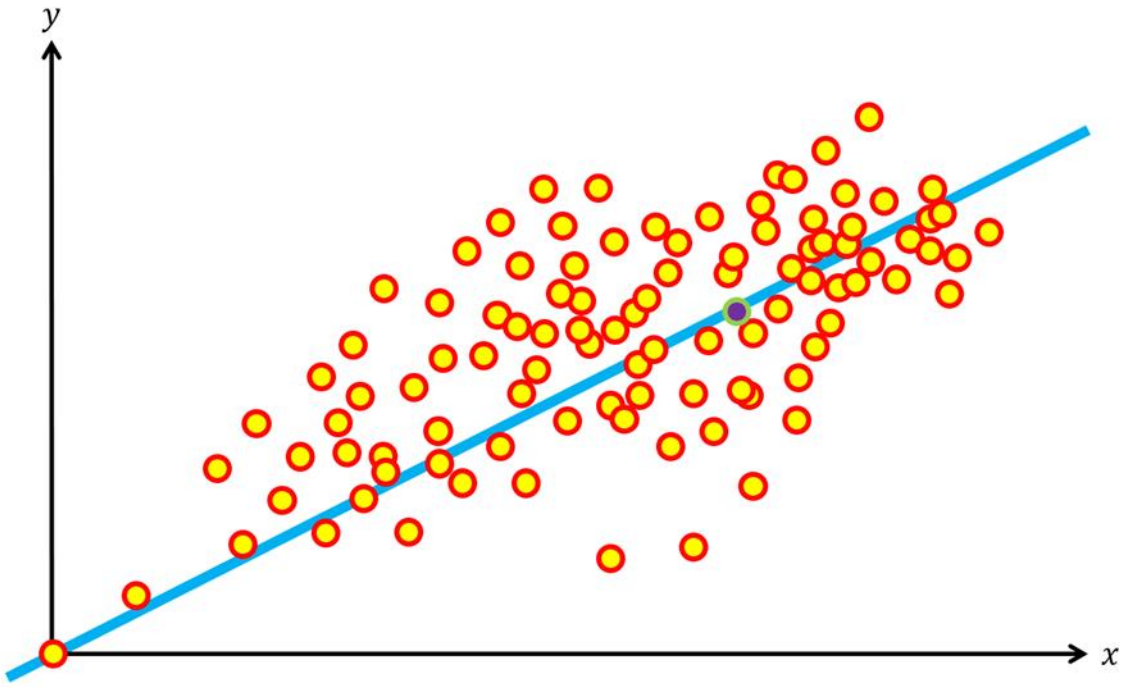
신박AI

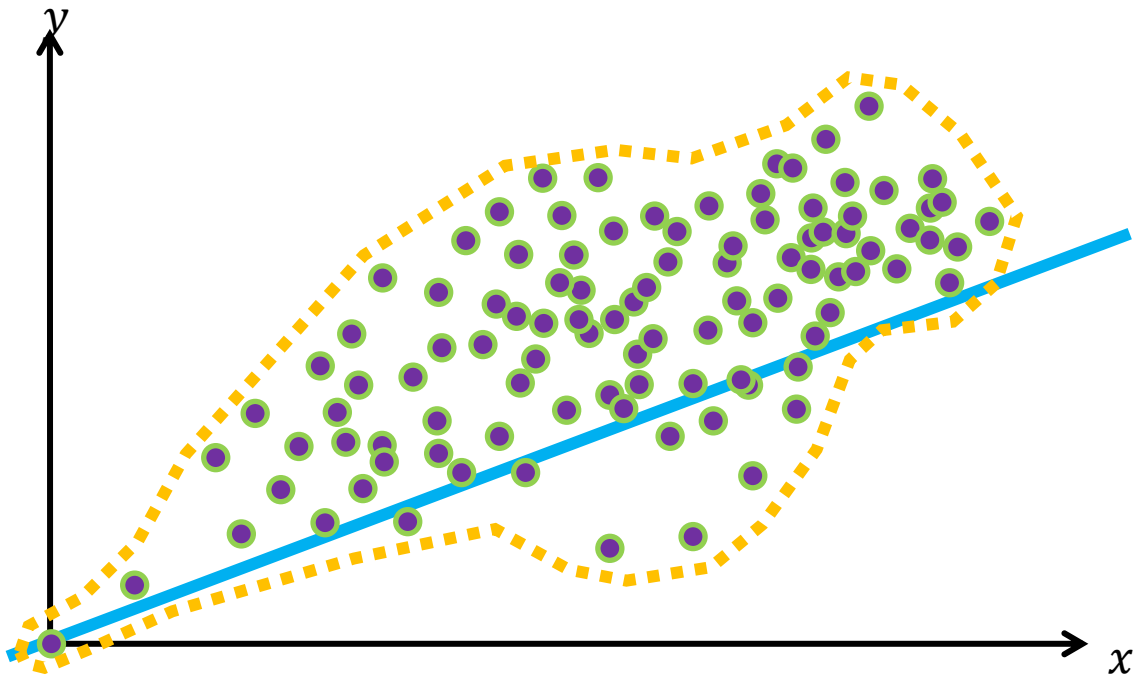
확률적 경사하강법에 비해 안정적이며,

효율



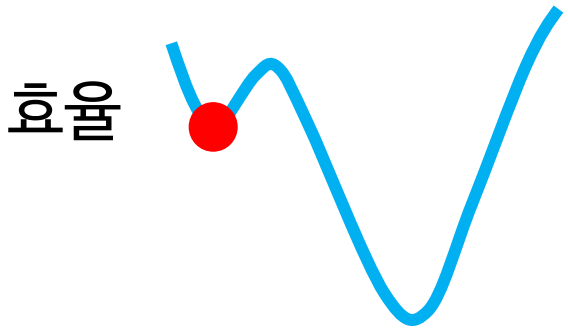
안정

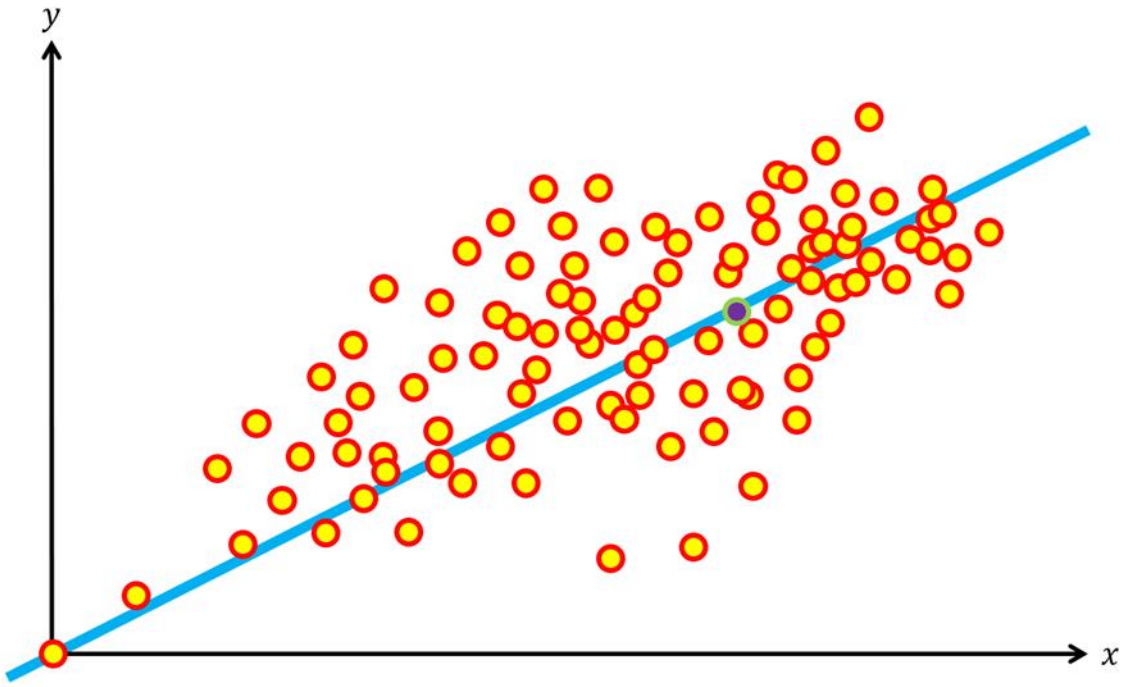


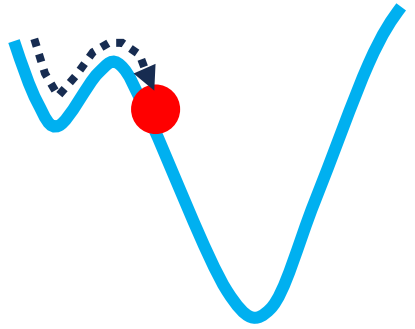


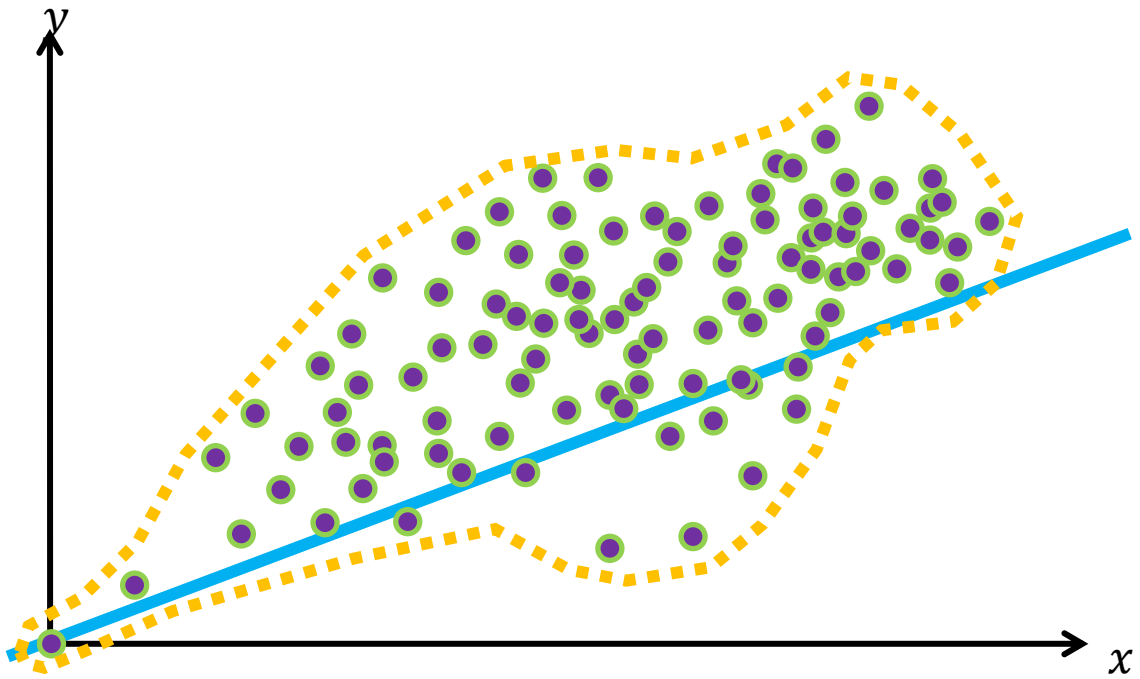
신박AI

확률적 경사하강법에 비해 로컬 미니마를 벗어날 확률도 높습니다



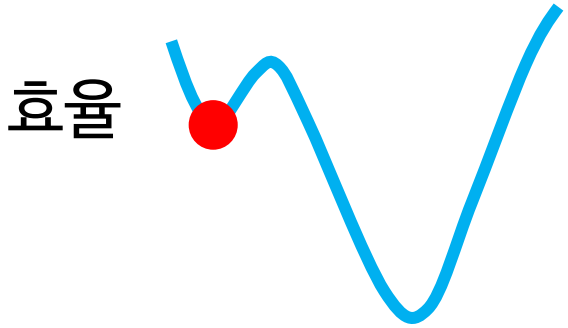


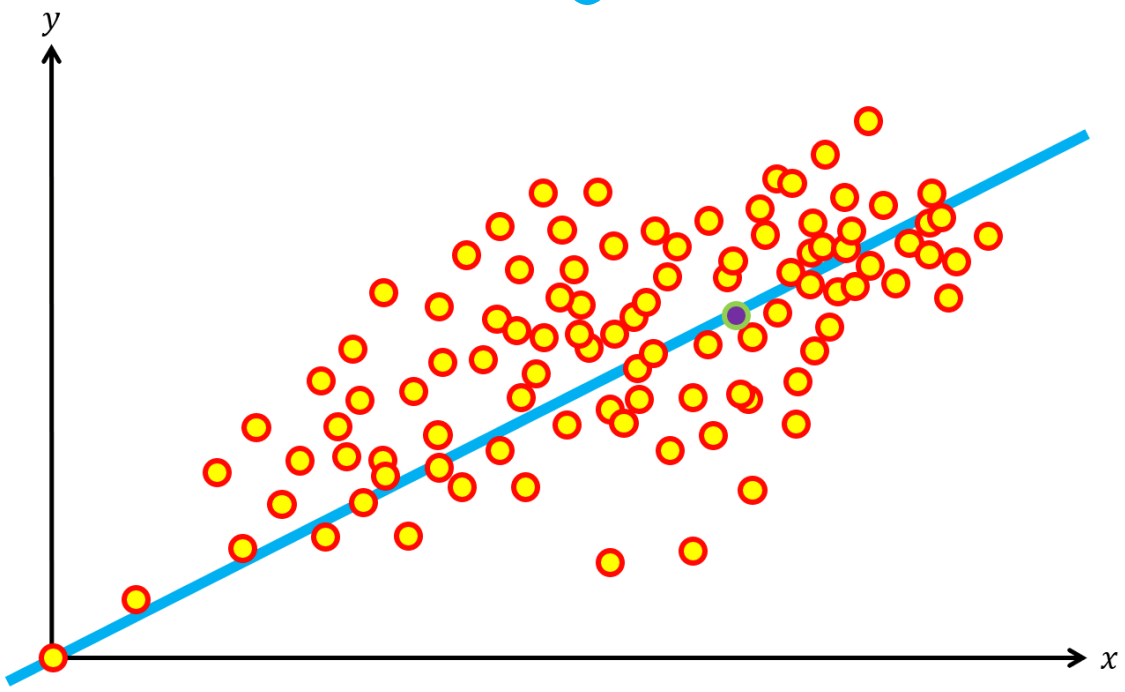


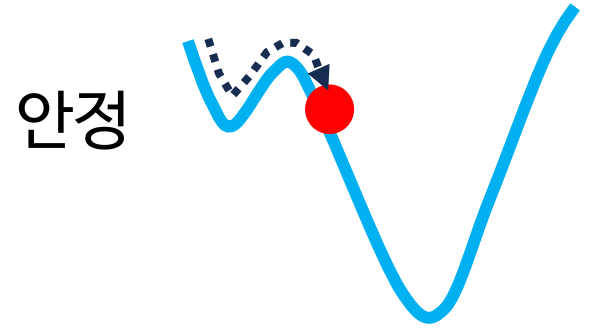


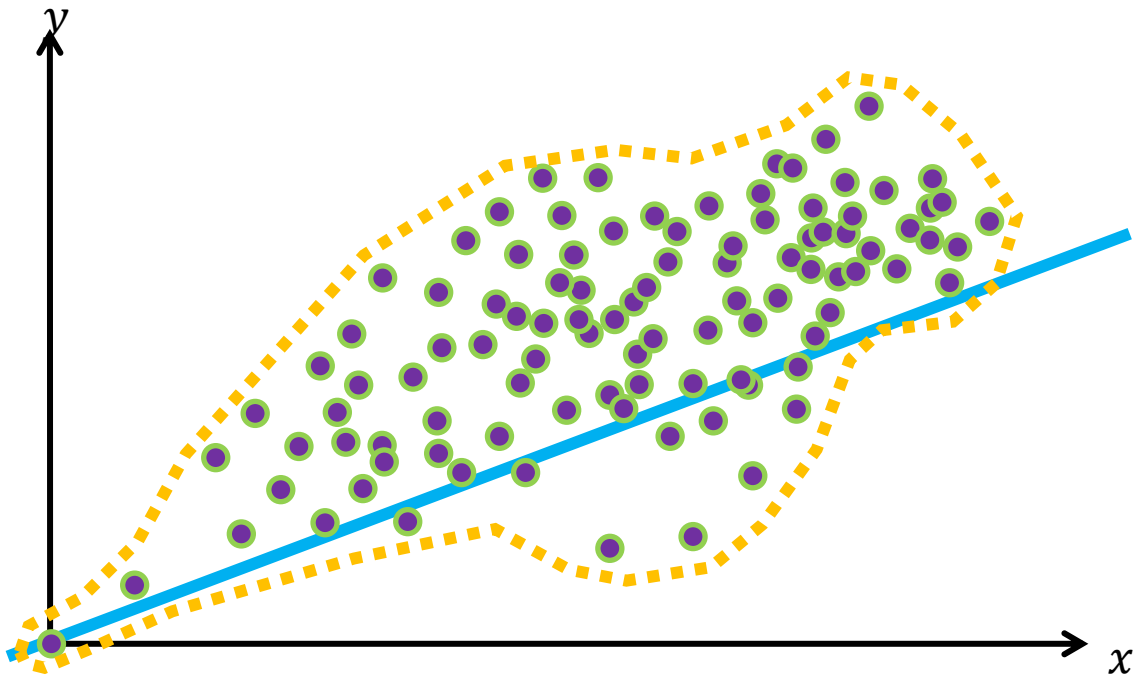
신박AI

그러면, 확률적 경사하강법의 효율과,



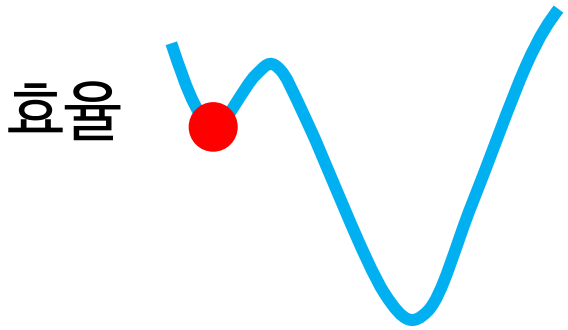


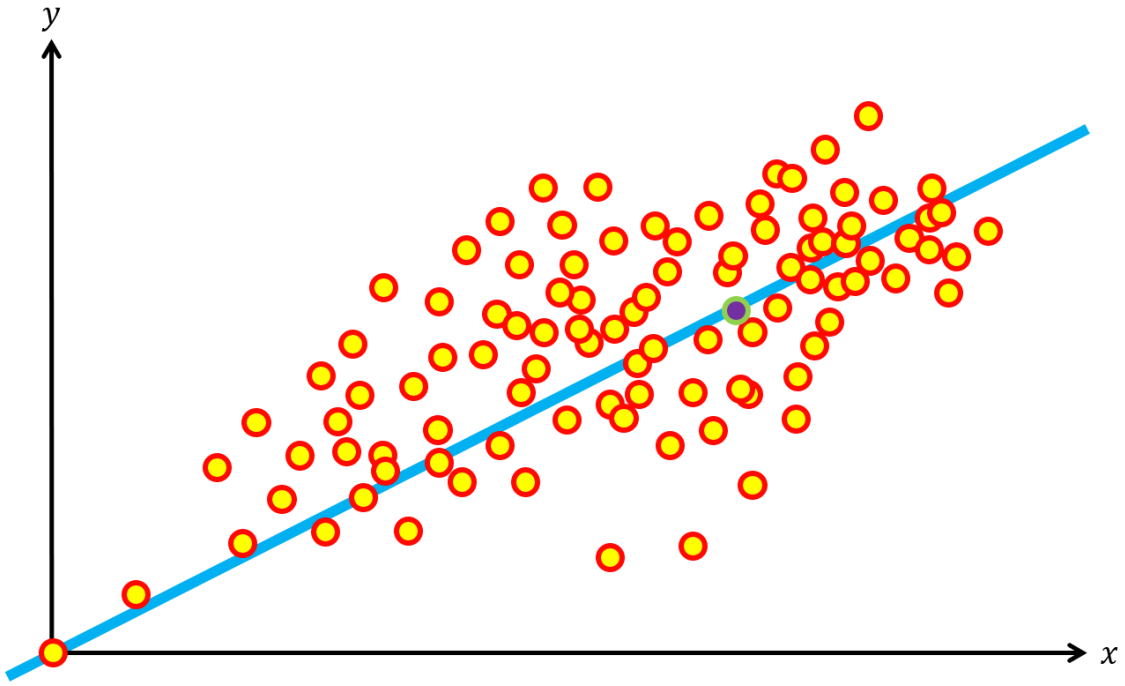


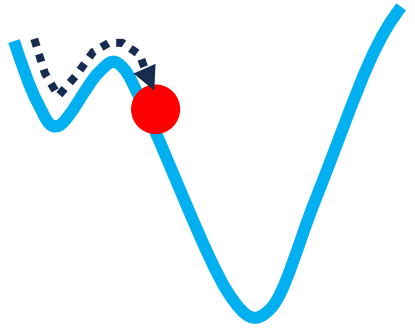


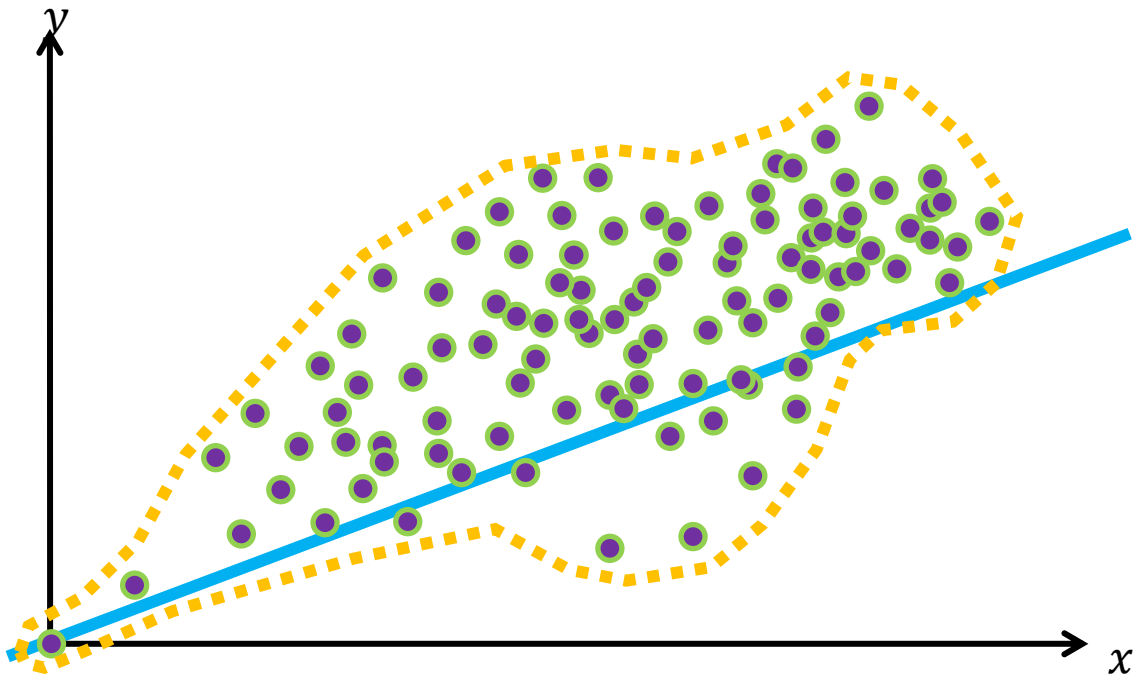
신박AI

배치 경사하강법의 안정적인 면을 골고루 갖춘 방법은 없을까요?



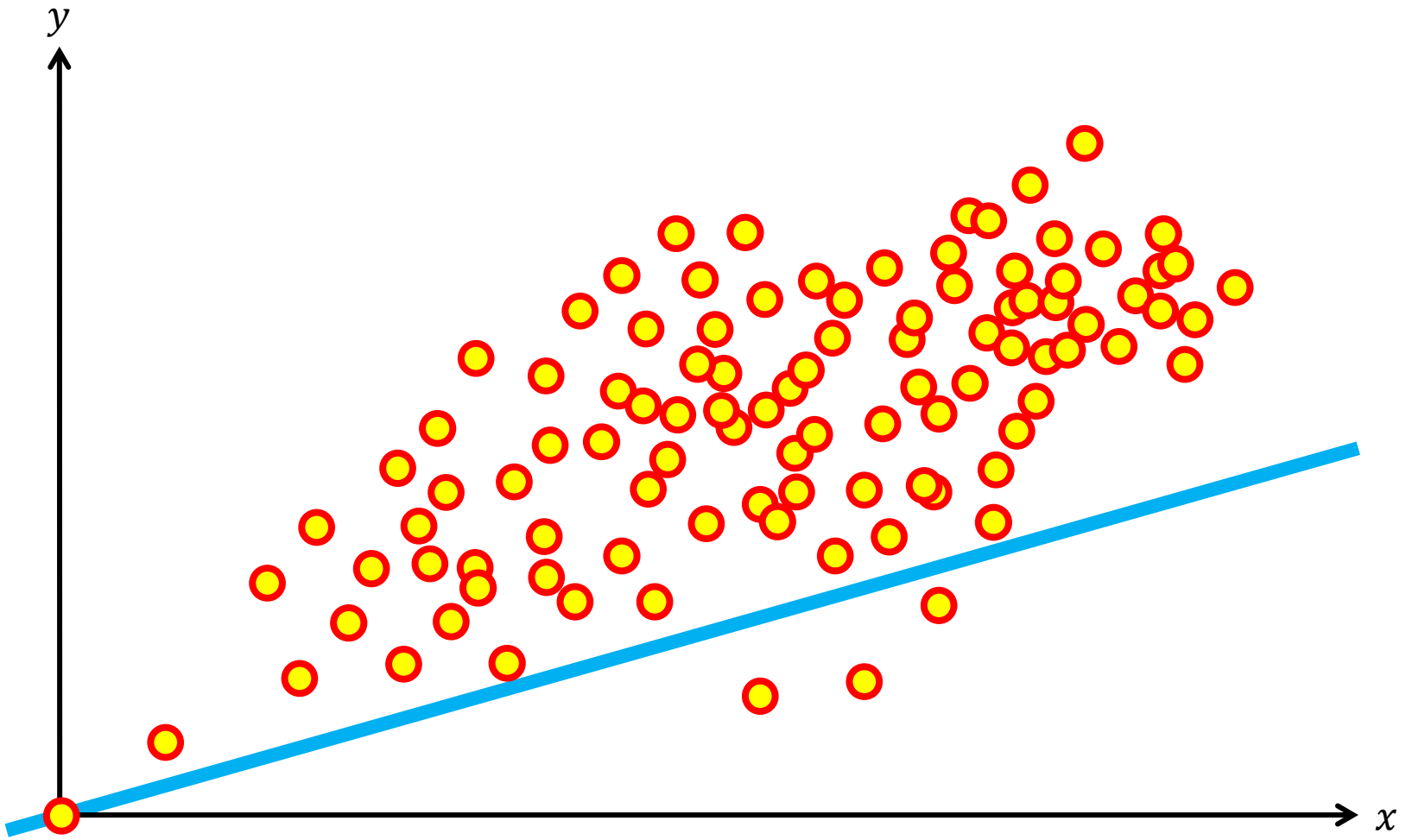






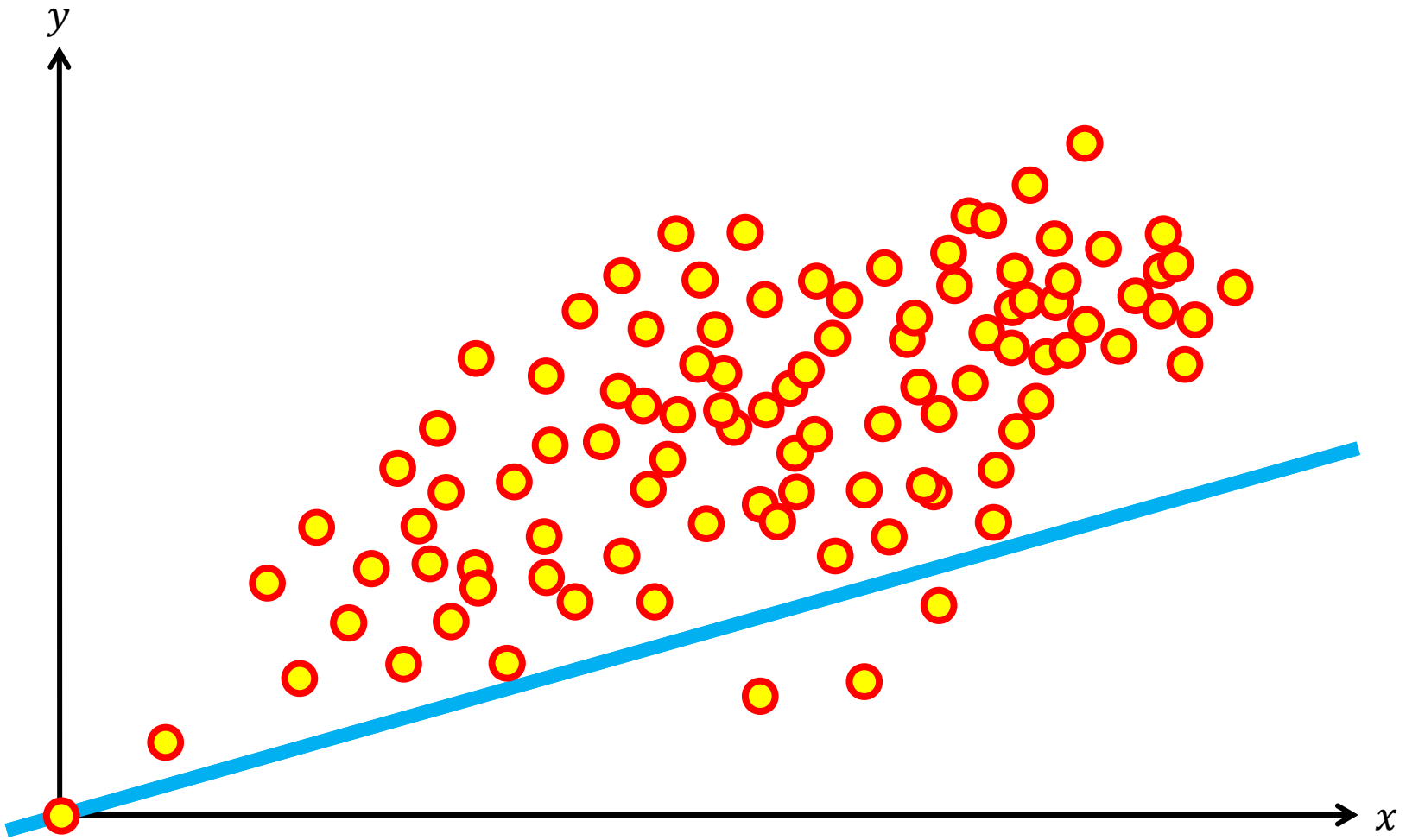
신박AI

둘의 장점을 고루 갖춘 경사하강법이 바로 미니-배치 경사하강법입니다
Mini-batch gradient descent



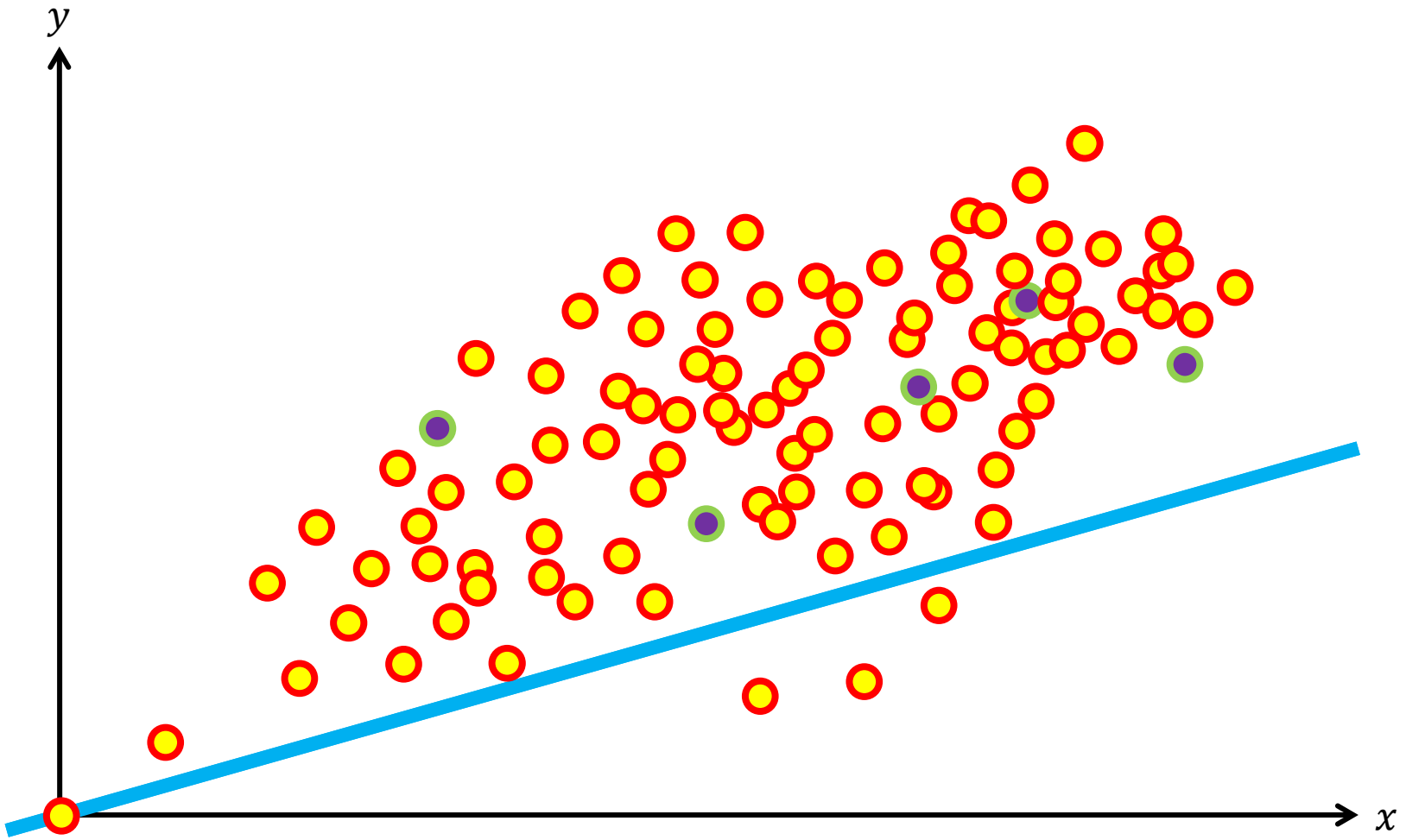
신박AI

미니-배치 경사하강법은 가중치를 계산하는데 일정 수의 배치 즉 미니배치 Mini-batch를 사용하는 경사하강법입니다



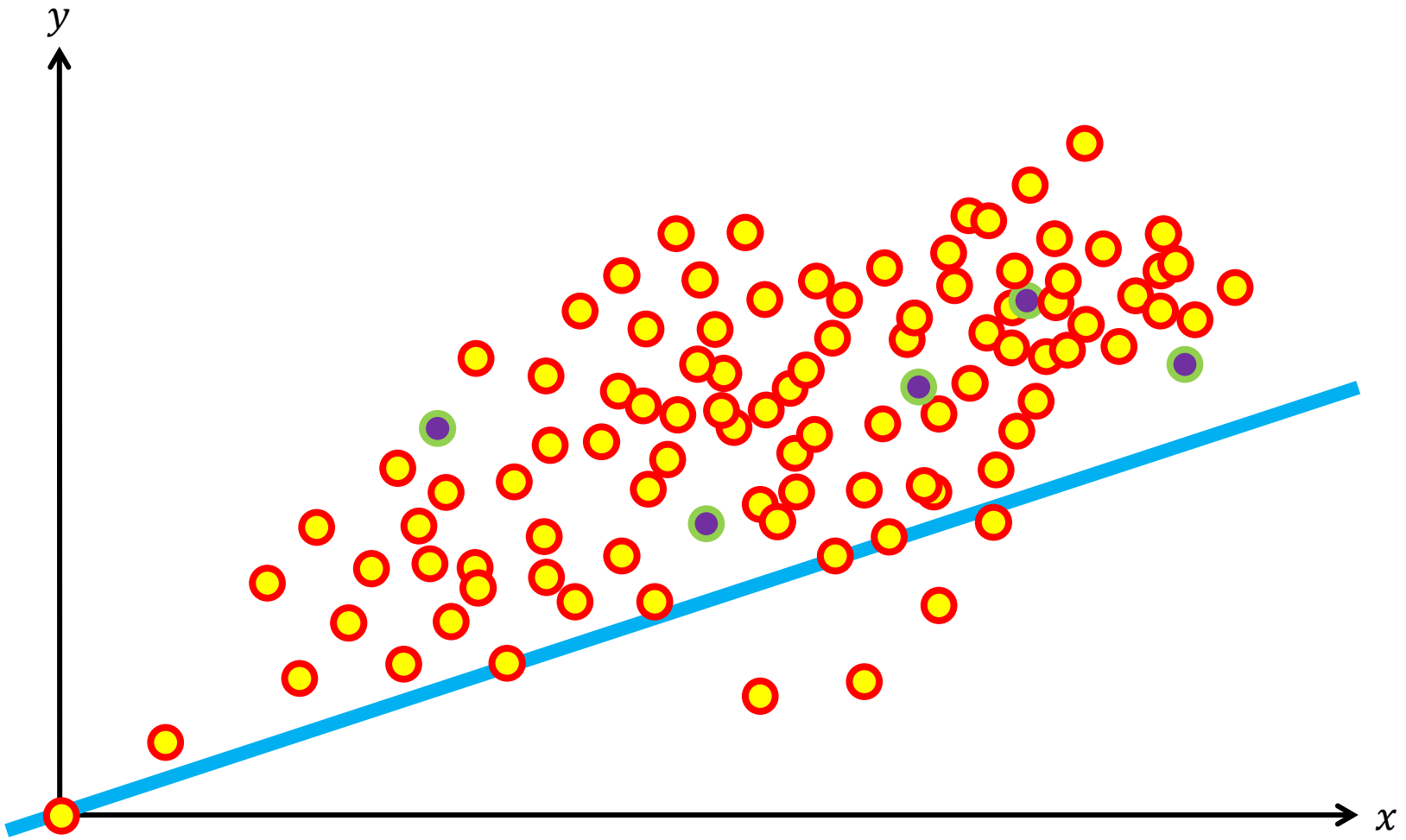
신박AI

즉 예를들어 랜덤하게 5개의 데이터를 선정해서 에러를 계산한 다음,



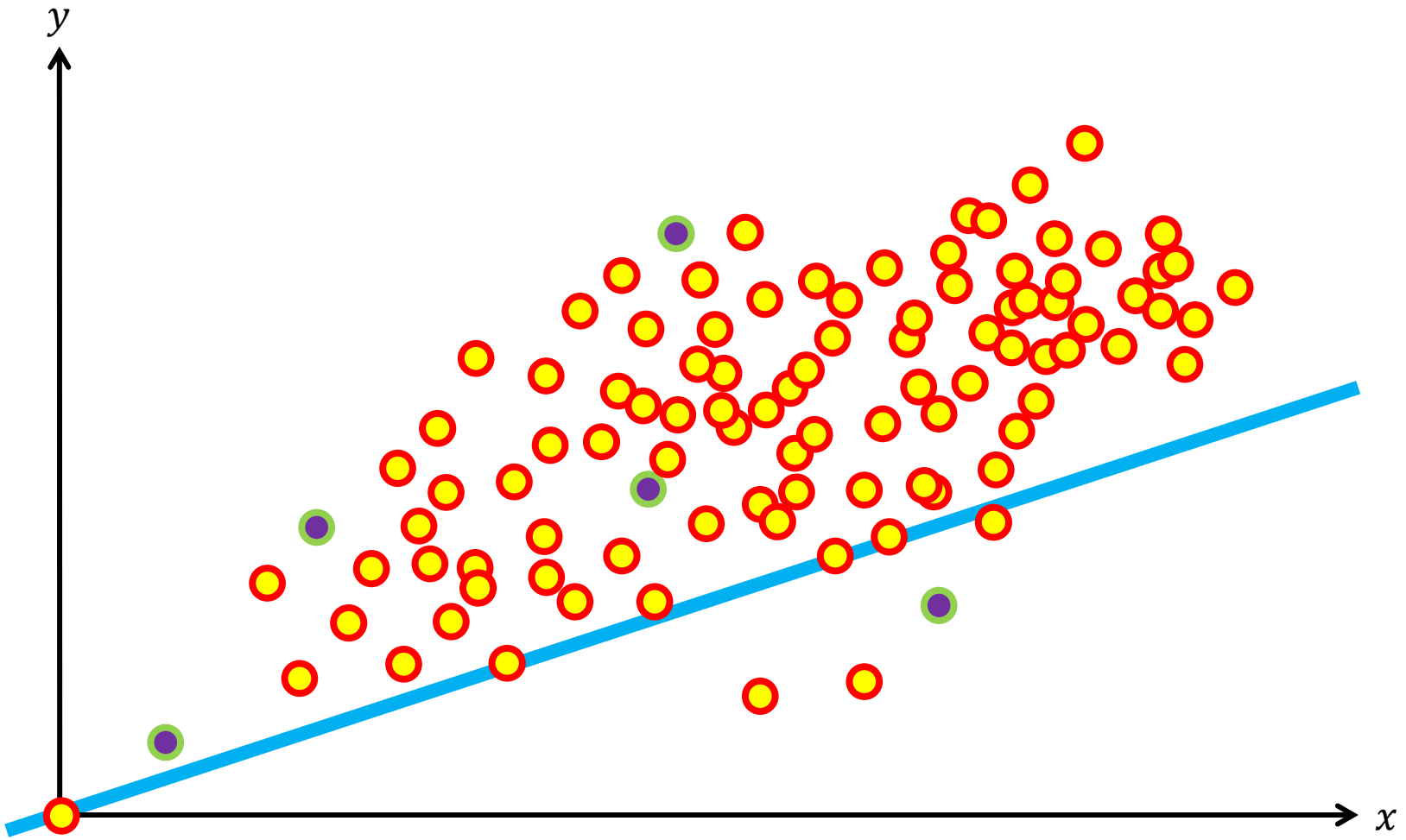
신박AI

5개의 평균 에러값을 사용하여 기울기를 업데이트할 수 있습니다



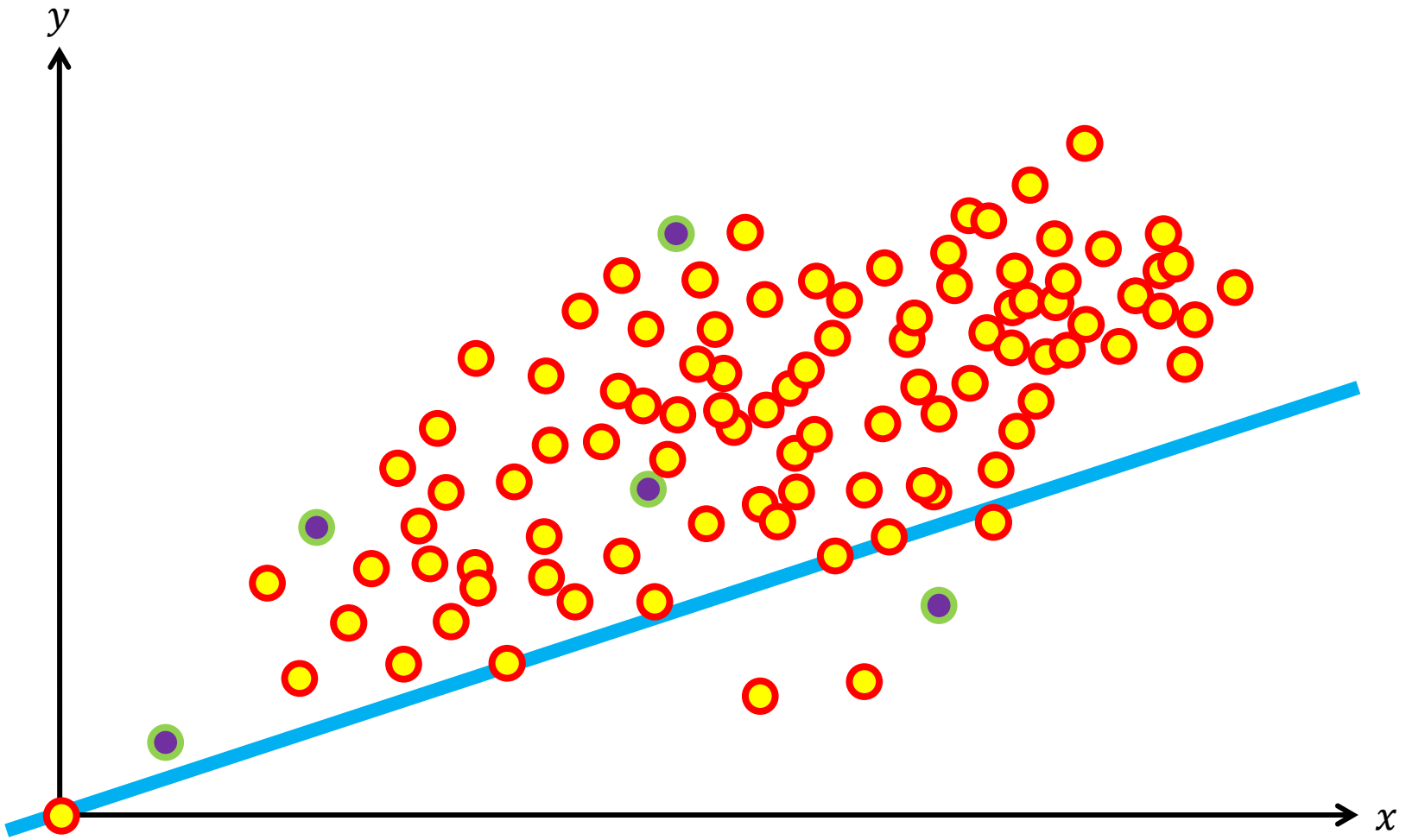
신박AI

또 랜덤하게 5개를 선정합니다.



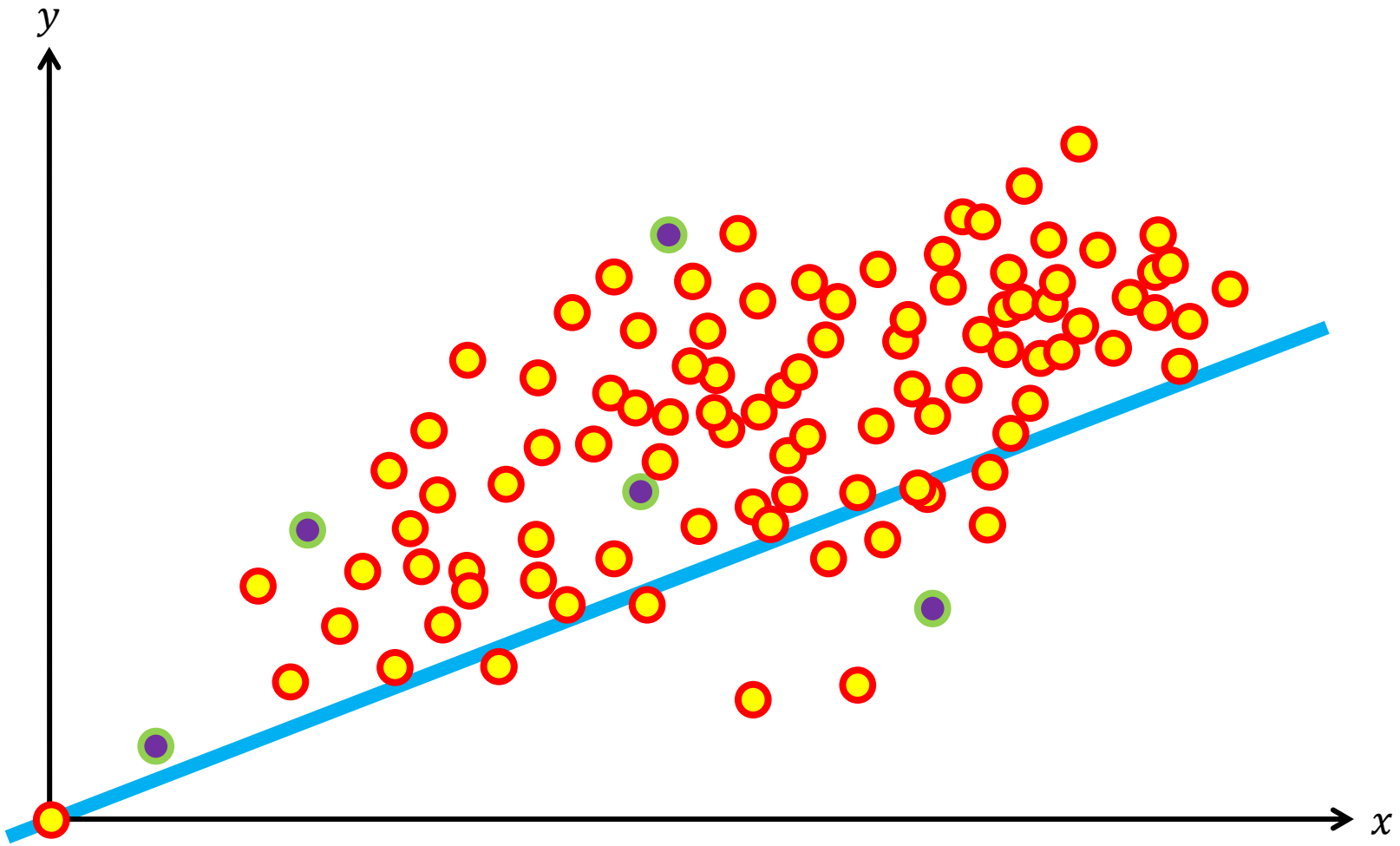
신박AI

이 5개의 데이터가 만들어내는 평균 에러값을 사용하여



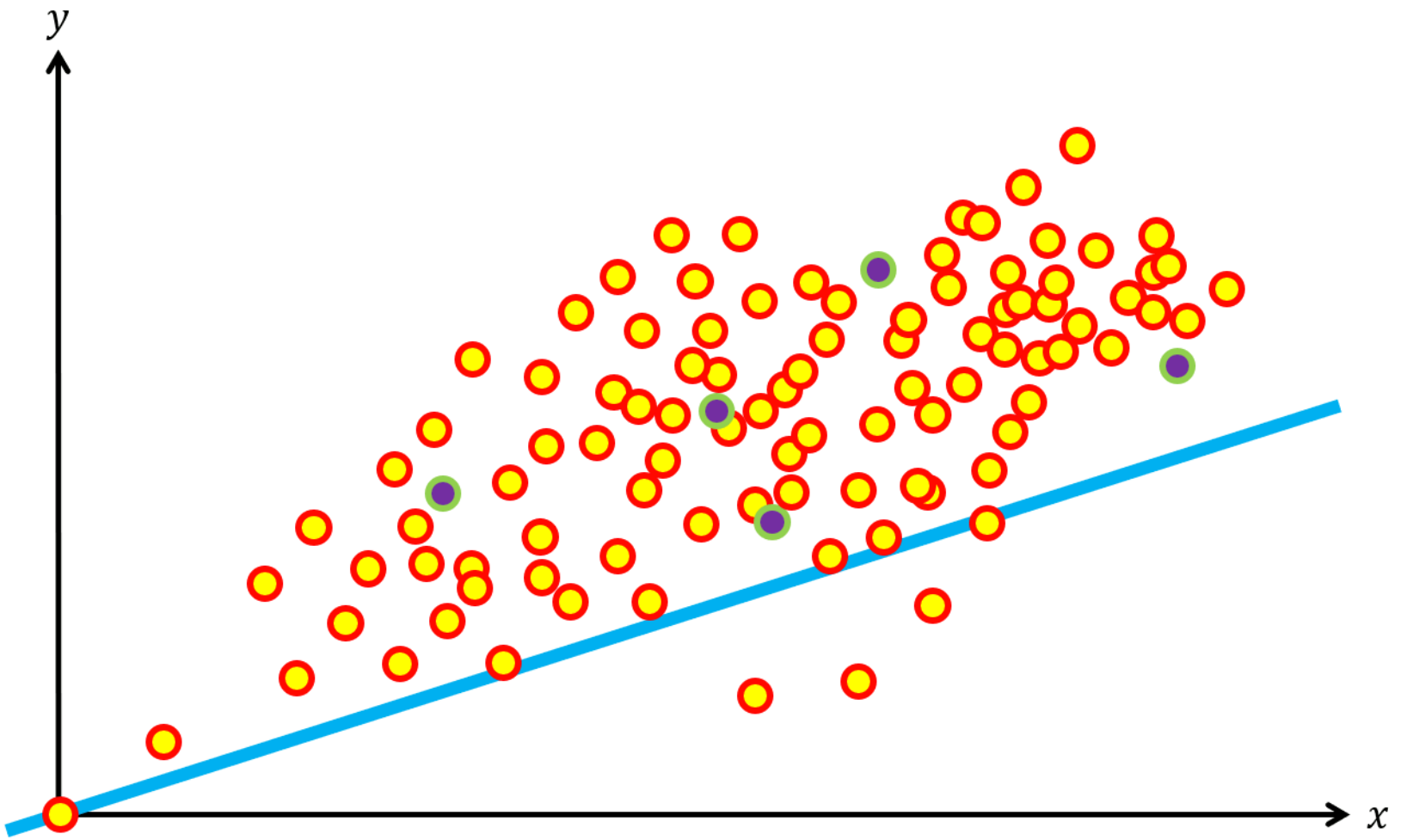
신박AI

또 다시 가중치를 업데이트 할 수 있습니다.



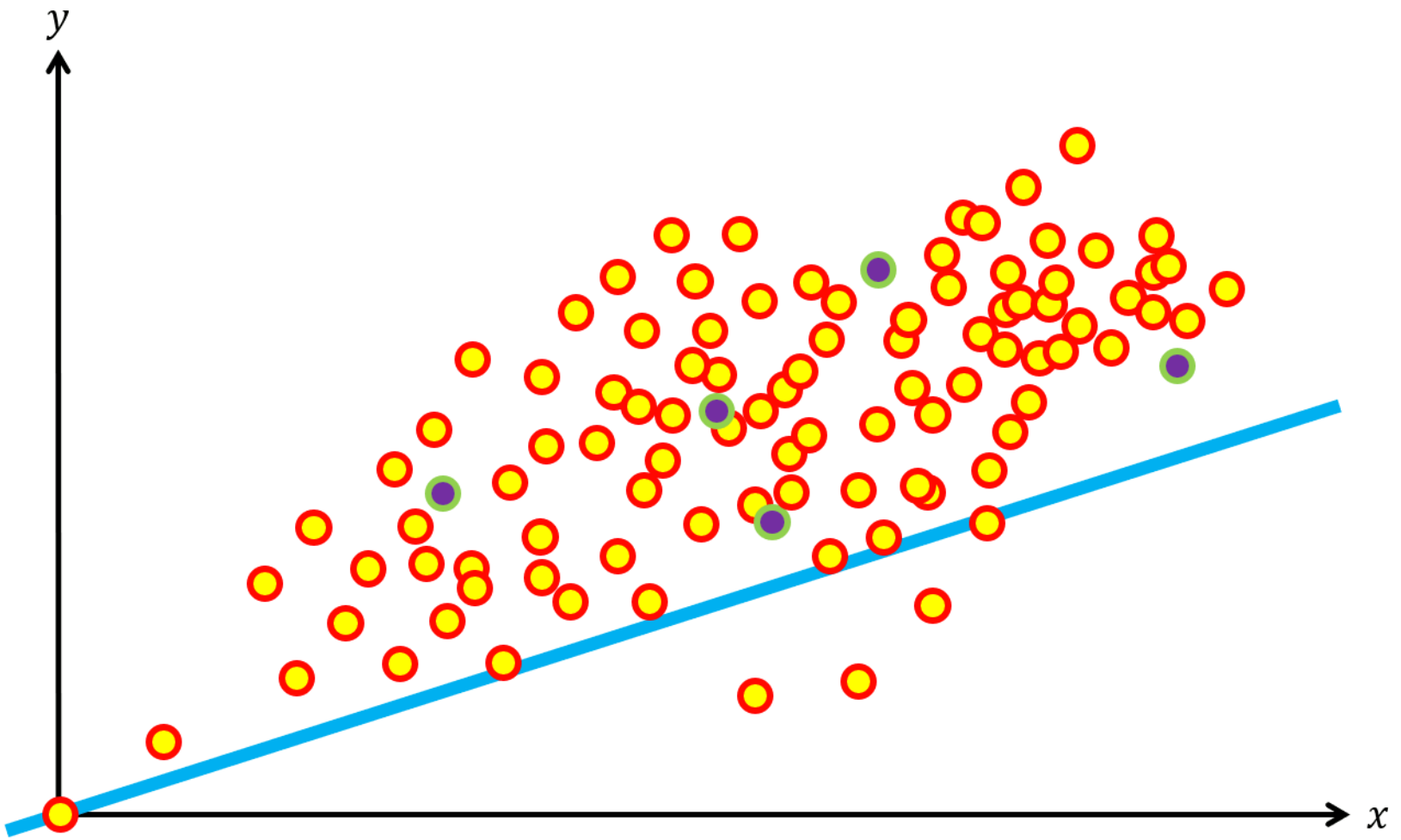
신박AI

이와 같이 특정 개수의 데이터만을 사용하여 가중치를 업데이트 하게 될 경우,



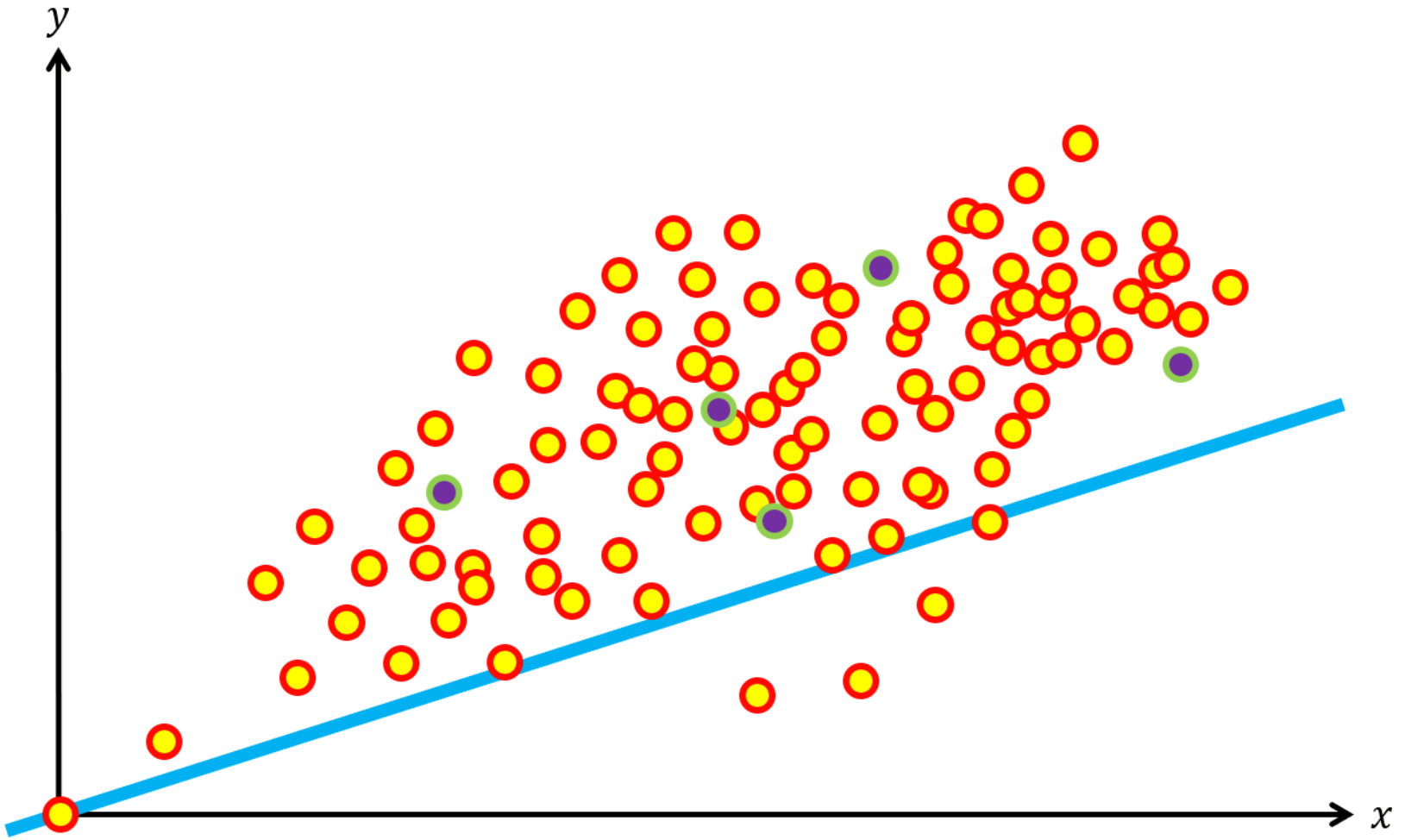
신박AI

데이터를 한 개만 사용하여 가중치를 업데이트하는 확률적 경사하강법보다는 안정적이면서,



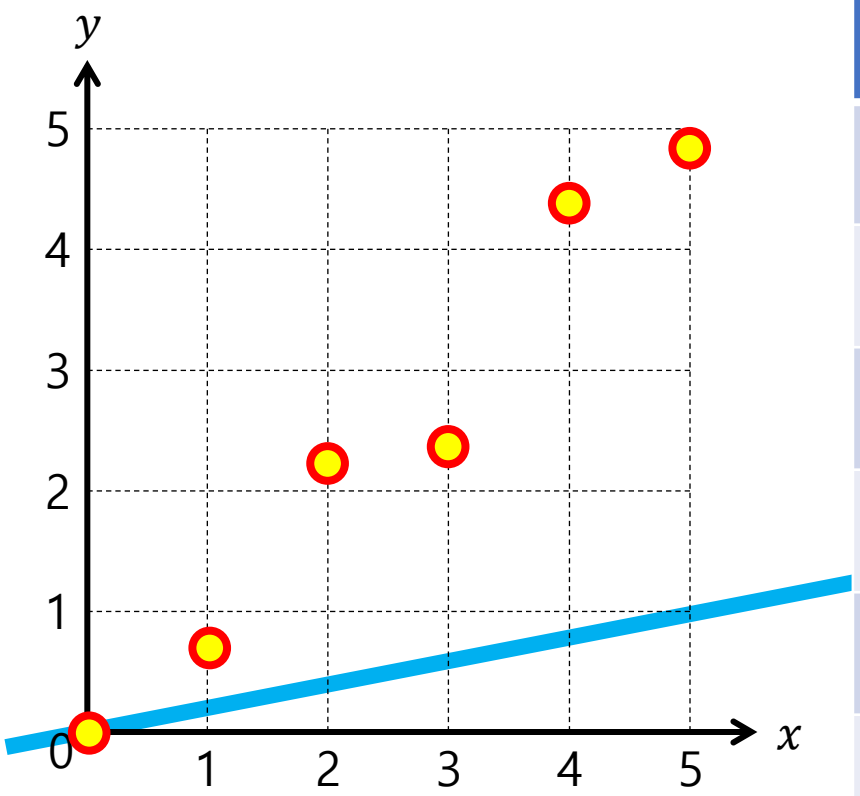
신박AI

데이터를 전부 사용하여 가중치를 '한번' 업데이트하는 배치 경사하강법 보다는 효율적인 경사하강법입니다.



신박AI

우리가 아까 보았던 예제로 돌아가서 설명을 드리자면,



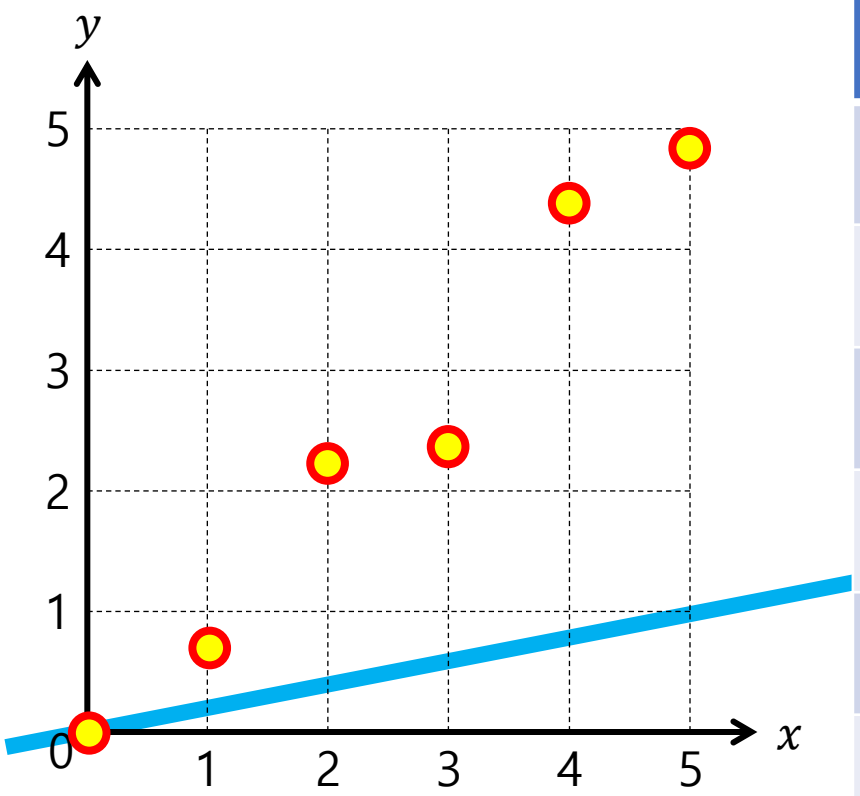
x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

신박AI

여기까지는 배치 경사하강법과 과정이 동일합니다.



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

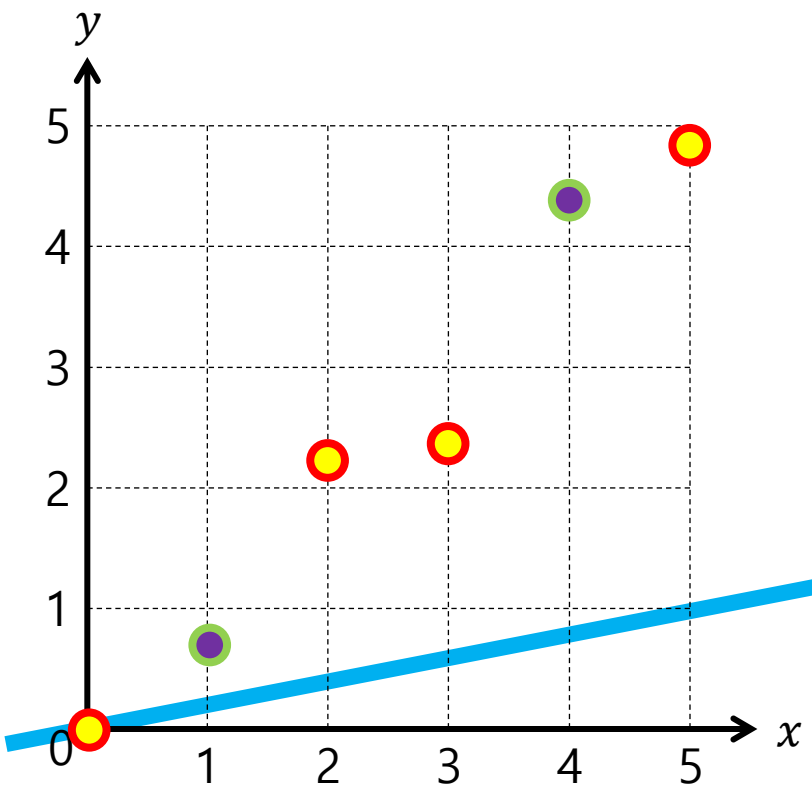
$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

신박AI

그러면 미니-배치 경사하강법의 경우라면 (n=2일 경우), 다음과 같은 두 개의 데이터를 넣어서 계산합니다



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

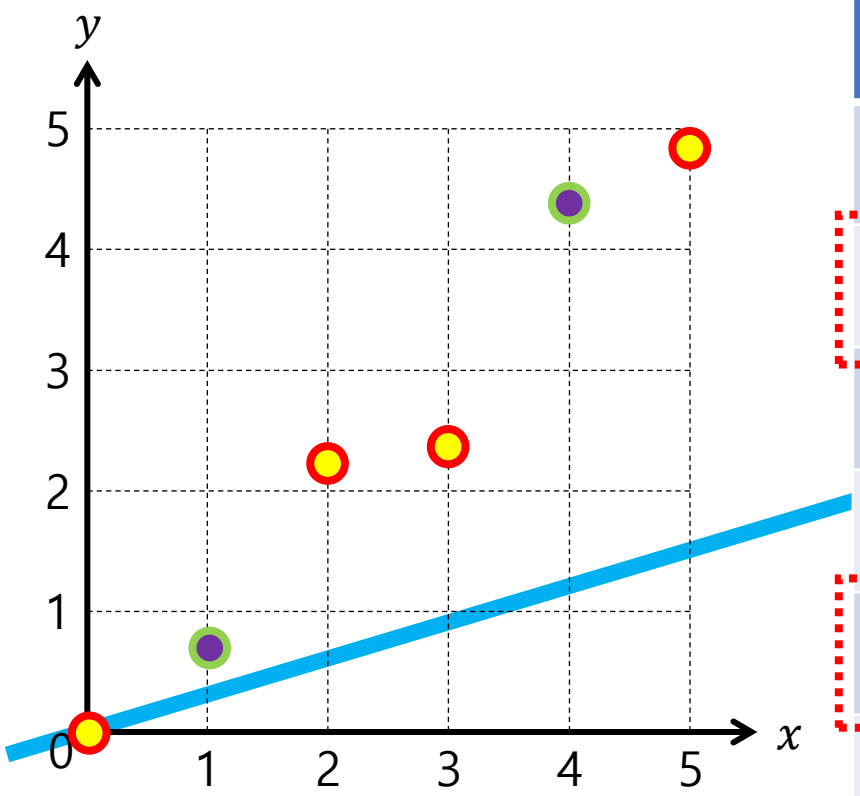
$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.2 - 0.01\left(\frac{1}{2}\cdot(-2)\cdot\begin{pmatrix}(0.8-0.2)\times 1\\ +(4.3-0.8)\times 4\end{pmatrix}\right)$$

신박AI

그러면 이렇게 새로운 가중치가 업데이트 됩니다.



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

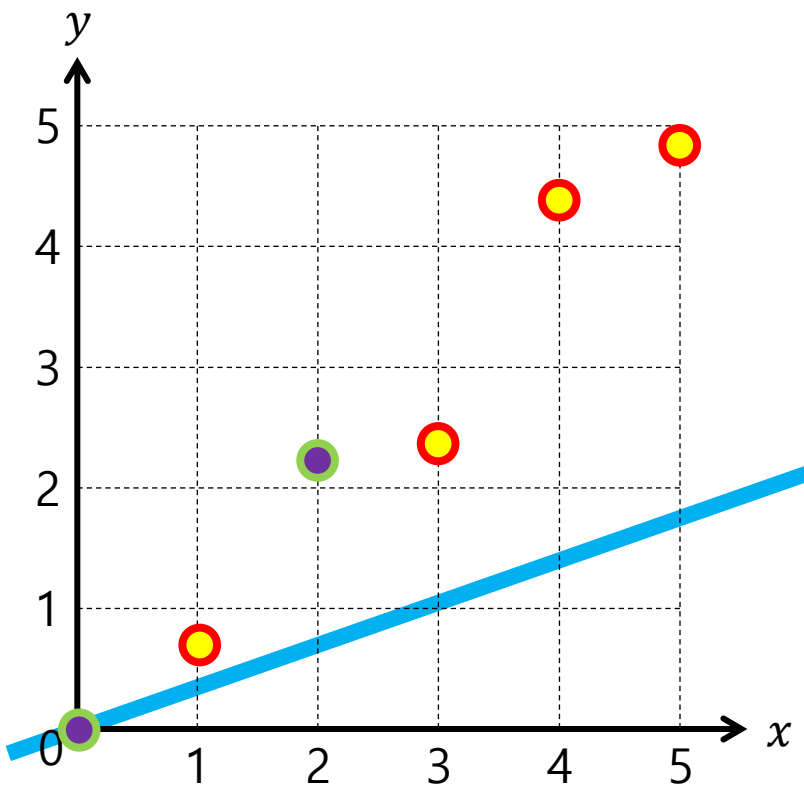
$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.2 - 0.01\left(\frac{1}{2}\cdot(-2)\cdot\begin{pmatrix}(0.8-0.2)\times 1\\ +(4.3-0.8)\times 4\end{pmatrix}\right)$$

$$= 0.346$$

신박AI

이렇게 계속 두 개씩 이용하여 가중치를 업데이트 하게 됩니다



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

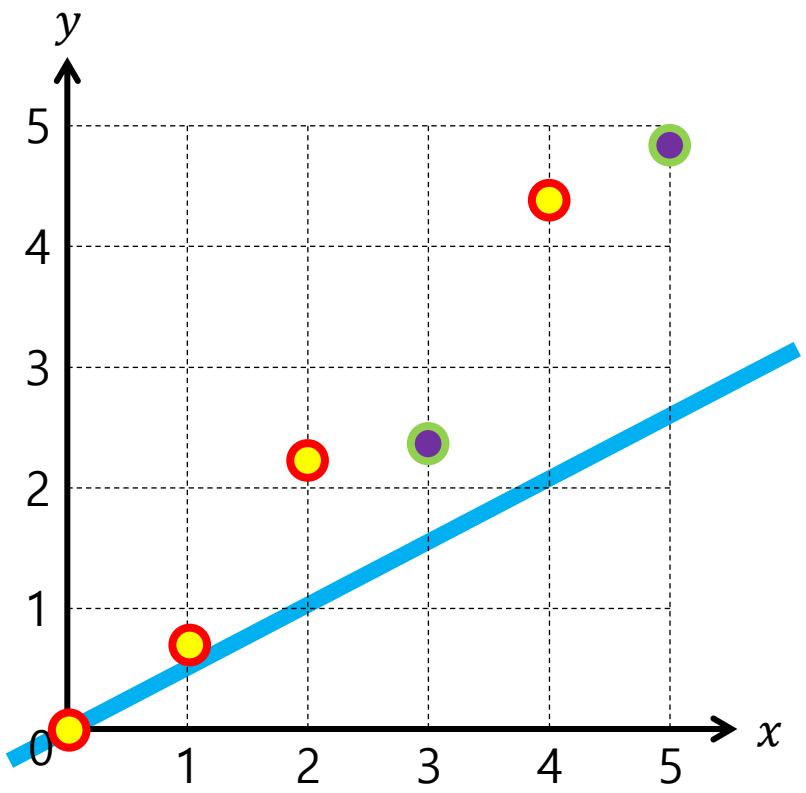
$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.346 - 0.01\left(\frac{1}{2}\cdot(-2)\cdot\begin{pmatrix}(0-0)\times 0\\ +(2.2-0.4)\times 2\end{pmatrix}\right)$$

$$= 0.382$$

신박AI

이렇게 계속 두 개씩 이용하여 가중치를 업데이트 하게 됩니다



x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.382 - 0.01\left(\frac{1}{2}\cdot(-2)\cdot\begin{pmatrix}(2.3-0.6)\times 3\\ +(5-4.9)\times 5\end{pmatrix}\right)$$

$$= 0.438$$

신박AI

이제 마지막으로, 각각의 경사하강법을 어떻게 실제로 코드로 작성하는지 살펴보도록 하겠습니다

다음 코드는 배치 경사하강법의 예제 코드입니다.

```
1 usage
def batch_gradient_descent(x, y_actual, w_init=0.5, lr=0.01, epochs=10):
    w = w_init
    mse_history = []
    w_history = [w_init]

    for epoch in range(epochs):
        # Predicted output
        y_hat = w * x

        # Compute gradient of the loss with respect to w
        gradient = -2 * np.mean(x * (y_actual - y_hat))
        # Update w using gradient descent
        w = w - lr * gradient
        # Store weight after update
        w_history.append(w)

        # Calculate Mean Squared Error (MSE)
        mse = np.mean((y_actual - y_hat) ** 2)
        mse_history.append(mse)
```

신박AI

다시 이 데이터셋을 예로 들어 설명하려고 합니다.

```
1 usage
def batch_gradient_descent(x, y_actual, w_init=0.5, lr=0.01, epochs=10):
    w = w_init
    mse_history = []
    w_history = [w_init]

    for epoch in range(epochs):
        # Predicted output
        y_hat = w * x

        # Compute gradient of the loss with respect to w
        gradient = -2 * np.mean(x * (y_actual - y_hat))
        # Update w using gradient descent
        w = w - lr * gradient
        # Store weight after update
        w_history.append(w)

        # Calculate Mean Squared Error (MSE)
        mse = np.mean((y_actual - y_hat) ** 2)
        mse_history.append(mse)
```

x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

신박AI

배치 경사하강법이기 때문에 $\hat{y}$에는 모든 예측값이 들어가게 됩니다

```
1 usage
def batch_gradient_descent(x, y_actual, w_init=0.5, lr=0.01, epochs=10):
    w = w_init
    mse_history = []
    w_history = [w_init]

    for epoch in range(epochs):
        # Predicted output
        y_hat = w * x

        # Compute gradient of the loss with respect to w
        gradient = -2 * np.mean(x * (y_actual - y_hat))
        # Update w using gradient descent
        w = w - lr * gradient
        # Store weight after update
        w_history.append(w)

        # Calculate Mean Squared Error (MSE)
        mse = np.mean((y_actual - y_hat) ** 2)
        mse_history.append(mse)
```

x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

신박AI

그리고 앞서 식에서 보았던 것 처럼, 모든 x 값과 $(y - \hat{y})$를 곱하여

```
1 usage
def batch_gradient_descent(x, y_actual, w_init=0.5, lr=0.01, epochs=10):
    w = w_init
    mse_history = []
    w_history = [w_init]

    for epoch in range(epochs):
        # Predicted output
        y_hat = w * x

        # Compute gradient of the loss with respect to w
        gradient = -2 * np.mean(x * (y_actual - y_hat))
        # Update w using gradient descent
        w = w - lr * gradient
        # Store weight after update
        w_history.append(w)

        # Calculate Mean Squared Error (MSE)
        mse = np.mean((y_actual - y_hat) ** 2)
        mse_history.append(mse)
```

x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.2 - \alpha\left(\frac{1}{6}\cdot(-2)\cdot\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

$$= 0.2 - 0.01\left(\frac{1}{6}\cdot(-2)\cdot\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

신박AI

-2를 곱하고 또 평균을 구하고

```
1 usage
def batch_gradient_descent(x, y_actual, w_init=0.5, lr=0.01, epochs=10):
    w = w_init
    mse_history = []
    w_history = [w_init]

    for epoch in range(epochs):
        # Predicted output
        y_hat = w * x

        # Compute gradient of the loss with respect to w
        gradient = -2 * np.mean(x * (y_actual - y_hat))
        # Update w using gradient descent
        w = w - lr * gradient
        # Store weight after update
        w_history.append(w)

        # Calculate Mean Squared Error (MSE)
        mse = np.mean((y_actual - y_hat) ** 2)
        mse_history.append(mse)
```

x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.2 - \alpha\left(\frac{1}{6}\cdot(-2)\cdot\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

$$= 0.2 - 0.01\left(\frac{1}{6}\cdot(-2)\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

신박AI

또 학습률을 곱하여 기존의 가중치값에서 빼주면 새로운 가중치가 나옵니다.

```
1 usage
def batch_gradient_descent(x, y_actual, w_init=0.5, lr=0.01, epochs=10):
    w = w_init
    mse_history = []
    w_history = [w_init]

    for epoch in range(epochs):
        # Predicted output
        y_hat = w * x

        # Compute gradient of the loss with respect to w
        gradient = -2 * np.mean(x * (y_actual - y_hat))
        # Update w using gradient descent
        w = w - lr * gradient
        # Store weight after update
        w_history.append(w)

        # Calculate Mean Squared Error (MSE)
        mse = np.mean((y_actual - y_hat) ** 2)
        mse_history.append(mse)
```

x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.2 - \alpha\left(\frac{1}{6}\cdot(-2)\cdot\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

$$= 0.2 - 0.01\left(\frac{1}{6}\cdot(-2)\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

신박AI

이와 같이 이 코드는 우리가 살펴본 배치 경사하강법 공식을 충실하게 구현하고 있습니다

```
1 usage
def batch_gradient_descent(x, y_actual, w_init=0.5, lr=0.01, epochs=10):
    w = w_init
    mse_history = []
    w_history = [w_init]

    for epoch in range(epochs):
        # Predicted output
        y_hat = w * x

        # Compute gradient of the loss with respect to w
        gradient = -2 * np.mean(x * (y_actual - y_hat))
        # Update w using gradient descent
        w = w - lr * gradient
        # Store weight after update
        w_history.append(w)

        # Calculate Mean Squared Error (MSE)
        mse = np.mean((y_actual - y_hat) ** 2)
        mse_history.append(mse)
```

x	y	$\hat{y}$
0	0	0
1	0.8	0.2
2	2.2	0.4
3	2.3	0.6
4	4.3	0.8
5	4.9	1

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.2 - \alpha\left(\frac{1}{6}\cdot(-2)\cdot\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

$$= 0.2 - 0.01\left(\frac{1}{6}\cdot(-2)\begin{pmatrix}0\times 0 + 0.6\times 1 + 1.8\times 2 + \\ 1.7\times 3 + 3.5\times 4 + 3.9\times 5\end{pmatrix}\right)$$

신박AI

이 코드는 확률적 경사하강법의 코드입니다

```
1 usage
def stochastic_gradient_descent(x, y_actual, w_init=0.2, lr=0.01, epochs=10):
    w = w_init
    mse_history = []
    w_history = [w_init]
    n = len(x)

    for epoch in range(epochs):
        for i in range(n):
            i = 2
            # Predicted output for a single data point
            y_hat = w * x[i]

            # Compute gradient of the loss with respect to w for a single data point
            gradient = -2 * x[i] * (y_actual[i] - y_hat)

            # Update w using gradient descent
            w = w - lr * gradient

            # Store weight after update
            w_history.append(w)

            # Calculate Mean Squared Error (MSE) for a single data point
            mse = (y_actual[i] - y_hat) ** 2
            mse_history.append(mse)
```

신박AI

보시는 바 처럼, 오직 하나의 데이터를 이용하여 기울기를 구하고 가중치를 업데이트 합니다

```python
1 usage
def stochastic_gradient_descent(x, y_actual, w_init=0.2, lr=0.01, epochs=10):
    w = w_init
    mse_history = []
    w_history = [w_init]
    n = len(x)

    for epoch in range(epochs):
        for i in range(n):
            i = 2
            # Predicted output for a single data point
            y_hat = w * x[i]

            # Compute gradient of the loss with respect to w for a single data point
            gradient = -2 * x[i] * (y_actual[i] - y_hat)

            # Update w using gradient descent
            w = w - lr * gradient

            # Store weight after update
            w_history.append(w)

            # Calculate Mean Squared Error (MSE) for a single data point
            mse = (y_actual[i] - y_hat) ** 2
            mse_history.append(mse)
```

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.2 - \alpha\left(\frac{1}{1}\cdot(-2)\cdot(2.2 - 0.4)\cdot 2\right)$$

$$= 0.2 - 0.01\left(\frac{1}{1}\cdot(-2)\cdot(2.2 - 0.4)\cdot 2\right)$$

$$= 0.272$$

신박AI

하나의 데이터만 사용하여 기울기를 구하는 점만 제외하면, 다른 가중치 업데이트 부분은 동일합니다.

```python
1 usage
def stochastic_gradient_descent(x, y_actual, w_init=0.2, lr=0.01, epochs=10):
    w = w_init
    mse_history = []
    w_history = [w_init]
    n = len(x)

    for epoch in range(epochs):
        for i in range(n):
            i = 2
            # Predicted output for a single data point
            y_hat = w * x[i]

            # Compute gradient of the loss with respect to w for a single data point
            gradient = -2 * x[i] * (y_actual[i] - y_hat)

            # Update w using gradient descent
            w = w - lr * gradient

            # Store weight after update
            w_history.append(w)

            # Calculate Mean Squared Error (MSE) for a single data point
            mse = (y_actual[i] - y_hat) ** 2
            mse_history.append(mse)
```

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.2 - \alpha\left(\frac{1}{1}\cdot(-2)\cdot(2.2 - 0.4)\cdot 2\right)$$

$$= 0.2 - 0.01\left(\frac{1}{1}\cdot(-2)\cdot(2.2 - 0.4)\cdot 2\right)$$

$$= 0.272$$

신박AI

이것은 미니-배치 경사하강법 코드입니다.

```python
def mini_batch_gradient_descent(x, y_actual, w_init=0.5, lr=0.01, epochs=10, batch_size=2):
    w = w_init
    mse_history = []
    w_history = [w_init]
    n = len(x)

    for epoch in range(epochs):
        indices = np.arange(n)
        np.random.shuffle(indices)
        x_shuffled = x[indices]
        y_shuffled = y_actual[indices]

        for i in range(0, n, batch_size):
            x_batch = x_shuffled[i:i + batch_size]
            y_batch = y_shuffled[i:i + batch_size]

            # Predicted output for the mini-batch
            y_hat = w * x_batch

            # Compute gradient of the loss with respect to w for the mini-batch
            gradient = -2 * np.mean(x_batch * (y_batch - y_hat))

            # Update w using gradient descent
            w = w - lr * gradient

            # Store weight after update
            w_history.append(w)
```

신박AI

여기서 주목하실 부분은, 바로 여기, 전체 데이터에서 배치사이즈 만큼의 데이터만 선별하여,

```python
def mini_batch_gradient_descent(x, y_actual, w_init=0.5, lr=0.01, epochs=10, batch_size=2):
    w = w_init
    mse_history = []
    w_history = [w_init]
    n = len(x)

    for epoch in range(epochs):
        indices = np.arange(n)
        np.random.shuffle(indices)
        x_shuffled = x[indices]
        y_shuffled = y_actual[indices]

        for i in range(0, n, batch_size):
            x_batch = x_shuffled[i:i + batch_size]
            y_batch = y_shuffled[i:i + batch_size]

            # Predicted output for the mini-batch
            y_hat = w * x_batch

            # Compute gradient of the loss with respect to w for the mini-batch
            gradient = -2 * np.mean(x_batch * (y_batch - y_hat))

            # Update w using gradient descent
            w = w - lr * gradient

            # Store weight after update
            w_history.append(w)
```

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.2 - 0.01\left(\frac{1}{2}\cdot(-2)\cdot\begin{pmatrix}(0.8-0.2)\times 1\\ +(4.3-0.8)\times 4\end{pmatrix}\right)$$

신박AI

학습에 사용할 수 있도록 정하는 부분입니다.

```python
def mini_batch_gradient_descent(x, y_actual, w_init=0.5, lr=0.01, epochs=10, batch_size=2):
    w = w_init
    mse_history = []
    w_history = [w_init]
    n = len(x)

    for epoch in range(epochs):
        indices = np.arange(n)
        np.random.shuffle(indices)
        x_shuffled = x[indices]
        y_shuffled = y_actual[indices]

        for i in range(0, n, batch_size):
            x_batch = x_shuffled[i:i + batch_size]
            y_batch = y_shuffled[i:i + batch_size]

            # Predicted output for the mini-batch
            y_hat = w * x_batch

            # Compute gradient of the loss with respect to w for the mini-batch
            gradient = -2 * np.mean(x_batch * (y_batch - y_hat))

            # Update w using gradient descent
            w = w - lr * gradient

            # Store weight after update
            w_history.append(w)
```

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.2 - 0.01\left(\frac{1}{2}\cdot(-2)\cdot\begin{pmatrix}(0.8-0.2)\times 1\\ +(4.3-0.8)\times 4\end{pmatrix}\right)$$

신박AI

가중치 학습에 사용되는 데이터를 선정하는 부분만 다를 뿐, 다른 부분은 기존과 동일합니다.

```python
def mini_batch_gradient_descent(x, y_actual, w_init=0.5, lr=0.01, epochs=10, batch_size=2):
    w = w_init
    mse_history = []
    w_history = [w_init]
    n = len(x)

    for epoch in range(epochs):
        indices = np.arange(n)
        np.random.shuffle(indices)
        x_shuffled = x[indices]
        y_shuffled = y_actual[indices]

        for i in range(0, n, batch_size):
            x_batch = x_shuffled[i:i + batch_size]
            y_batch = y_shuffled[i:i + batch_size]

            # Predicted output for the mini-batch
            y_hat = w * x_batch

            # Compute gradient of the loss with respect to w for the mini-batch
            gradient = -2 * np.mean(x_batch * (y_batch - y_hat))

            # Update w using gradient descent
            w = w - lr * gradient

            # Store weight after update
            w_history.append(w)
```

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.2 - 0.01\left(\frac{1}{2}\cdot(-2)\cdot\begin{pmatrix}(0.8-0.2)\times 1\\ +(4.3-0.8)\times 4\end{pmatrix}\right)$$

신박AI

이 코드들을 Github 페이지에도 올려놓도록 하겠습니다 ☺

```python
def mini_batch_gradient_descent(x, y_actual, w_init=0.5, lr=0.01, epochs=10, batch_size=2):
    w = w_init
    mse_history = []
    w_history = [w_init]
    n = len(x)

    for epoch in range(epochs):
        indices = np.arange(n)
        np.random.shuffle(indices)
        x_shuffled = x[indices]
        y_shuffled = y_actual[indices]

        for i in range(0, n, batch_size):
            x_batch = x_shuffled[i:i + batch_size]
            y_batch = y_shuffled[i:i + batch_size]

            # Predicted output for the mini-batch
            y_hat = w * x_batch

            # Compute gradient of the loss with respect to w for the mini-batch
            gradient = -2 * np.mean(x_batch * (y_batch - y_hat))

            # Update w using gradient descent
            w = w - lr * gradient

            # Store weight after update
            w_history.append(w)
```

$$\hat{y} = 0.2x \quad \rightarrow \hat{y} = wx$$

$$MSE(L) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

$$w^* = w - \alpha\frac{\partial L}{\partial w} = w - \alpha\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w}$$

$$= w - \alpha\left(\frac{1}{n}\cdot(-2)\sum_{i}^{n}(y_i - \hat{y}_i)\cdot x_i\right)$$

$$= 0.2 - 0.01\left(\frac{1}{2}\cdot(-2)\cdot\begin{pmatrix}(0.8-0.2)\times 1\\ +(4.3-0.8)\times 4\end{pmatrix}\right)$$

신박AI

자 여기까지가 오늘 제가 준비한
확률적 경사하강법에 관한 영상입니다

신박AI

확률적 경사하강법 Stochastic Gradient Decent라는 단어가 어려워서 처음에는 왠지 어려워 보이지만

실상 그렇게 어렵지 않고 쉽게 이해할 수 있는
최적화 기법임을 알 수 있었습니다

신박AI

지금까지 시청해 주셔서 감사합니다

신박AI

다음시간에는 더 유익하고 재미있는 영상으
로 찾아뵙겠습니다

신박AI

그때까지 안녕히 계세요

신박AI

감사합니다!

좋은 하루 되세요!!

신박AI

이 채널은 여러분의 관심과 사랑이 필요합니다

신박AI

‘좋아요’와 ‘구독’버튼은 강의 준비에 큰 힘이 됩니다!

신박AI

그리고 영상 자료를 사용하실때는
출처 '신박AI'를 밝혀주세요

신박AI

신박AI