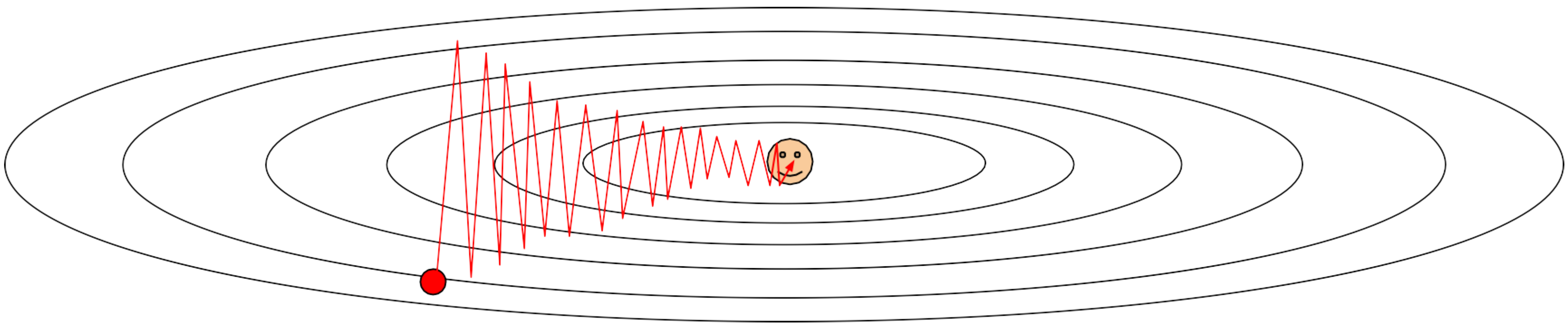
2021 Spring

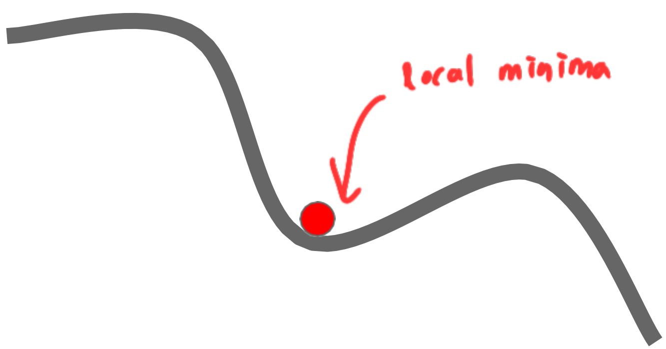
Artificial Intelligence & Deep Learning

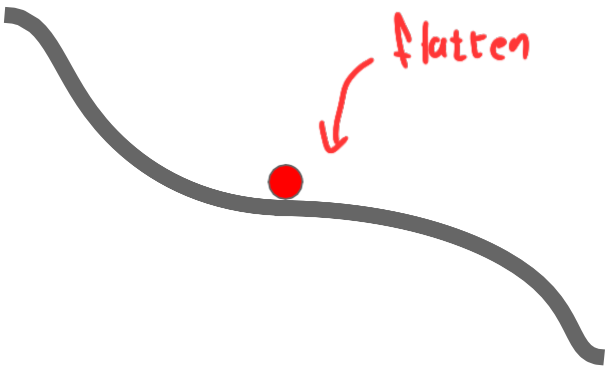
Prof. Minsuk Koo

Department of Computer Science & Engineering
Incheon National University

SGD의 문제점







5.2.3 모멘텀

그레이디언트의 잡음 현상

- 기계 학습은 훈련집합을 이용하여 그레이디언트를 추정하므로 잡음 가능성 높음
- 모멘텀은 그레이디언트에 스무딩을 가하여 잡음 효과 줄임 → 수렴 속도 향상

모멘텀을 적용한 가중치 갱신 수식

$$\left.\begin{array}{l} \mathbf{v} = \alpha\mathbf{v} - \rho\dfrac{\partial J}{\partial \mathbf{\Theta}} \\ \mathbf{\Theta} = \mathbf{\Theta} + \mathbf{v} \end{array}\right\} \qquad (5.12)$$

velocity

가중치

- 속도 벡터 $\mathbf{v}$는 이전 그레이디언트를 누적한 것에 해당함(처음에는 $\mathbf{v} = 0$로 출발)
- α의 효과
  - $\alpha = 0$이면 모멘텀이 적용 안 된 이전 공식과 같음
  - α가 1에 가까울수록 이전 그레이디언트 정보에 큰 가중치를 주는 셈 → $\mathbf{\Theta}$가 그리는 궤적이 매끄러움
  - 보통 0.5, 0.9, 또는 0.99 사용 (또는 0.5로 시작하여 세대가 지남에 따라 점점 키워 0.99에 도달하는 방법)

5.2.3 모멘텀

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x += learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

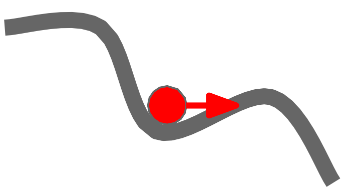
```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x += learning_rate * vx
```

누적

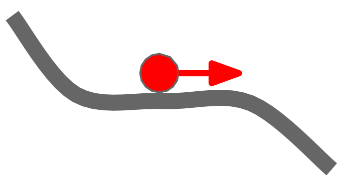
gradient 역할

- local minima 문제 해결
- flatten 문제 해결

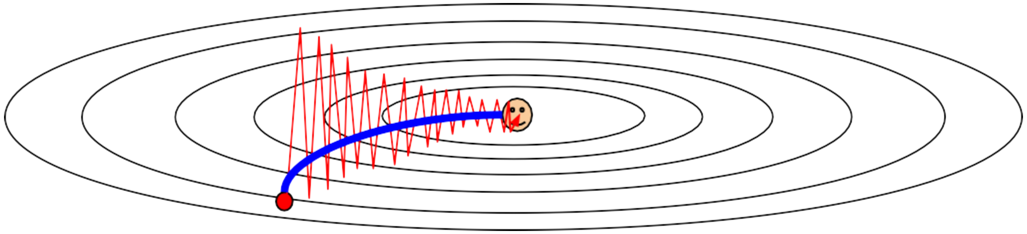
Local Minima



Saddle points



Poor Conditioning



5.2.3 모멘텀

- 모멘텀의 효과
  - 오버슈팅 현상 누그러뜨림

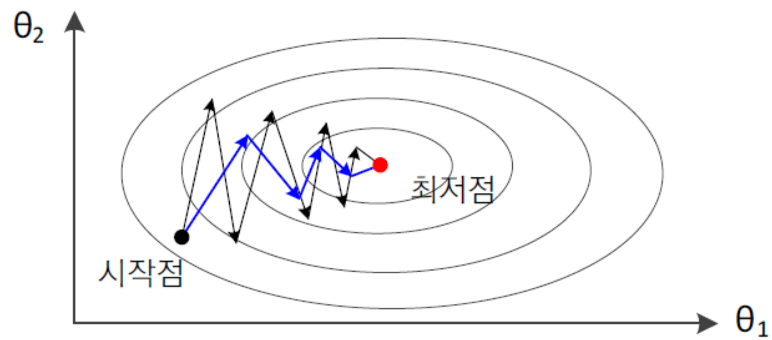


그림 5-10 모멘텀 효과

- 네스테로프 모멘텀
  - 현재 $\mathbf{v}$ 값으로 다음 이동할 곳 $\tilde{\mathbf{\Theta}}$를 예견한 후, 예견한 곳의 그레이디언트 $\left.\frac{\partial J}{\partial \mathbf{\Theta}}\right|_{\tilde{\mathbf{\Theta}}}$를 사용

$$\tilde{\mathbf{\Theta}} = \mathbf{\Theta} + \alpha \mathbf{v}$$
$$\mathbf{v} = \alpha \mathbf{v} - \rho \left.\frac{\partial J}{\partial \mathbf{\Theta}}\right|_{\tilde{\mathbf{\Theta}}} \qquad (5.13)$$
$$\mathbf{\Theta} = \mathbf{\Theta} + \mathbf{v}$$

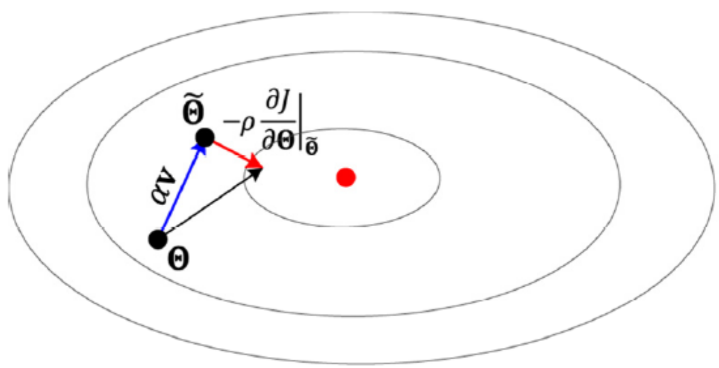


그림 5-11 네스테로프 모멘텀

5.2.3 모멘텀

- 일반적인 경사 하강 알고리즘

네스테로프 모멘텀을 적용한 경사 하강 알고리즘

gradient

$$v_{t+1} = \rho v_t - \alpha \boxed{\nabla f(\boxed{x_t + \rho v_t})}$$

$$x_{t+1} = x_t + v_{t+1}$$

$= \tilde{x}$

Annoying, usually we want update in terms of $x_t, \nabla f(x_t)$

Change of variables $\tilde{x}_t = x_t + \rho v_t$ and rearrange:

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$

$$\tilde{x}_{t+1} = \tilde{x}_t - \rho v_t + (1 + \rho) v_{t+1}$$

$\tilde{x}_{t+1} = x_{t+1} + v_{t+1}$

$$= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)$$

```
dx = compute_gradient(x)
old_v = v
v = rho * v - learning_rate * dx
x += -rho * old_v + (1 + rho) * v
```

5.2.4 적응적 학습률

- 학습률 ρ의 중요성
  - 너무 크면 오버슈팅에 따른 진자 현상, 너무 작으면 수렴이 느림

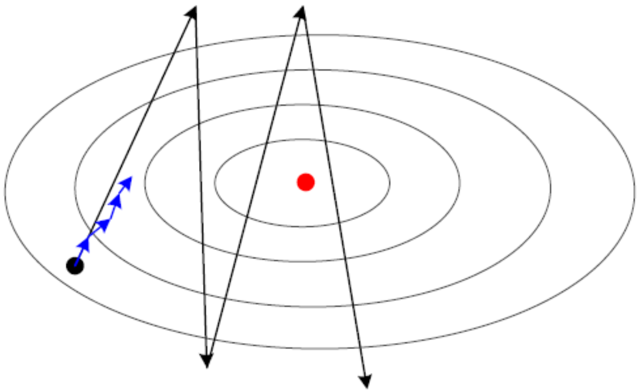


그림 5-12 학습률의 크기에 따른 최적화 알고리즘의 이동 궤적

- 적응적 학습률
  - 그레이디언트에 학습률 ρ를 곱하면, $\rho\frac{\partial J}{\partial \boldsymbol{\Theta}} = \left(\rho\frac{\partial J}{\partial \theta_1}, \rho\frac{\partial J}{\partial \theta_2}, \cdots, \rho\frac{\partial J}{\partial \theta_k}\right)^{\mathrm{T}}$. 즉 모든 매개변수가 같은 크기의 학습률을 사용하는 셈
  - 적응적 학습률은 매개변수마다 자신의 상황에 따라 학습률을 조절해 사용

5.2.4 적응적 학습률

- AdaGrad
  - 라인 5~7을 자세히 쓰면

update 얼마나 해 주냐

5. $(r_1, r_2, \cdots, r_k)^{\mathrm{T}} = (r_1 + g_1^2, r_2 + g_2^2, \cdots, r_k + g_k^2)^{\mathrm{T}}$

6. $(\Delta\theta_1, \Delta\theta_2, \cdots, \Delta\theta_k)^{\mathrm{T}} = \left(-\frac{\rho g_1}{\epsilon + \sqrt{r_1}}, -\frac{\rho g_2}{\epsilon + \sqrt{r_2}}, \cdots, -\frac{\rho g_k}{\epsilon + \sqrt{r_k}}\right)^{\mathrm{T}}$

7. $(\theta_1, \theta_2, \cdots, \theta_k)^{\mathrm{T}} = (\theta_1 + \Delta\theta_1, \theta_2 + \Delta\theta_2, \cdots, \theta_k + \Delta\theta_k)^{\mathrm{T}}$

  - $\mathbf{r}$은 이전 그레이디언트를 누적한 벡터
    - r_i가 크면 $|\Delta\theta_i|$는 작아서 조금만 이동
    - r_i가 작으면 $|\Delta\theta_i|$는 커서 많이 이동
    - 예) [그림 5-10]에서 θ_1은 θ_2보다 보폭이 큼
  - 라인 6의 $\frac{\rho}{\epsilon + \sqrt{r_i}}$는 상황에 따라 보폭을 정해주는 적응적 학습률로 볼 수 있음

알고리즘 5-3 AdaGrad

입력: 훈련집합 $\mathbb{X}$, $\mathbb{Y}$, 학습률 ρ
출력: 최적의 매개변수 $\widehat{\mathbf{\Theta}}$

1 난수를 생성하여 초기해 $\mathbf{\Theta}$를 설정한다.
2 $\mathbf{r} = \mathbf{0}$ // 그레이디언트 누적 벡터 초기화
3 repeat
4 그레이디언트 $\mathbf{g} = \left.\frac{\partial J}{\partial \mathbf{\Theta}}\right|_{\mathbf{\Theta}}$를 구한다.
5 $\mathbf{r} = \mathbf{r} + \mathbf{g} \odot \mathbf{g}$ // $\odot$는 요소별 곱
6 $\Delta\mathbf{\Theta} = -\frac{\rho}{\epsilon + \sqrt{\mathbf{r}}} \odot \mathbf{g}$
7 $\mathbf{\Theta} = \mathbf{\Theta} + \Delta\mathbf{\Theta}$
8 until (멈춤 조건)
9 $\widehat{\mathbf{\Theta}} = \mathbf{\Theta}$

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

5.2.4 적응적 학습률

- RMSProp
  - AdaGrad의 단점
    - [알고리즘 5-3]의 라인 5는 단순히 제곱을 더함
    - 따라서 오래된 그레이디언트와 최근 그레이디언트는 같은 비중의 역할 → $\mathbf{r}$이 점점 커져 수렴 방해할 가능성
  - RMSProp은 가중 이동 평균 기법 적용

$$\mathbf{r} = \alpha\mathbf{r} + (1-\alpha)\mathbf{g}\odot\mathbf{g} \qquad (5.14)$$

    - α가 작을수록 최근 것에 비중을 둠
    - 보통 α로 0.9, 0.99, 0.999를 사용

알고리즘 5-4 RMSProp

입력: 훈련집합 $\mathbb{X}$, $\mathbb{Y}$, 학습률 ρ, 가중 이동 평균 계수 α

출력: 최적의 매개변수 $\hat{\mathbf{\Theta}}$

난수를 생성하여 초기해 $\mathbf{\Theta}$를 설정한다.

$\mathbf{r} = \mathbf{0}$ // 그레이디언트 누적 벡터 초기화

repeat

  그레이디언트 $\mathbf{g} = \left.\frac{\partial J}{\partial \mathbf{\Theta}}\right|_{\mathbf{\Theta}}$를 구한다.

  $\mathbf{r} = \alpha\mathbf{r} + (1-\alpha)\mathbf{g}\odot\mathbf{g}$ // $\odot$는 요소별 곱

  $\Delta\mathbf{\Theta} = -\frac{\rho}{\epsilon+\sqrt{\mathbf{r}}}\odot\mathbf{g}$

  $\mathbf{\Theta} = \mathbf{\Theta} + \Delta\mathbf{\Theta}$

until (멈춤 조건)

$\hat{\mathbf{\Theta}} = \mathbf{\Theta}$

5.2.4 적응적 학습률

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

0.9, 0.99 ...

5.2.4 적응적 학습률

Adam

- RMSProp에 식 (5-12)의 모멘텀을 추가로 적용한 알고리즘

알고리즘 5-5 Adam

입력: 훈련집합 $\mathbb{X}$, $\mathbb{Y}$, 학습률 ρ, 모멘텀 계수 α_1, 가중 이동 평균 계수 α_2
출력: 최적의 매개변수 $\widehat{\mathbf{\Theta}}$

난수를 생성하여 초기해 $\mathbf{\Theta}$를 설정한다.
$\mathbf{v} = \mathbf{0}, \mathbf{r} = \mathbf{0}$
$t=1$
repeat
  그레이디언트 $\mathbf{g} = \frac{\partial J}{\partial \mathbf{\Theta}}\Big|_{\mathbf{\Theta}}$ 를 구한다.
  $\mathbf{v} = \alpha_1 \mathbf{v} - (1-\alpha_1)\mathbf{g}$ // 속도 벡터
  $\mathbf{v} = \frac{1}{1-(\alpha_1)^t}\mathbf{v}$
  $\mathbf{r} = \alpha_2 \mathbf{r} + (1-\alpha_2)\mathbf{g} \odot \mathbf{g}$ // 그레이디언트 누적 벡터
  $\mathbf{r} = \frac{1}{1-(\alpha_2)^t}\mathbf{r}$
  $\Delta\mathbf{\Theta} = -\frac{\rho}{\epsilon+\sqrt{\mathbf{r}}}\mathbf{v}$
  $\mathbf{\Theta} = \mathbf{\Theta} + \Delta\mathbf{\Theta}$
  t++
until (멈춤 조건)
$\widehat{\mathbf{\Theta}} = \mathbf{\Theta}$

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Momentum

AdaGrad / RMSProp

RMSProp + momentum 같은 방식

Q: 첫 스텝에서 문제점? 엄청 커짐

처음시도로 유용한 값들

beta1 = 0.9,

beta2 = 0.999

learning_rate = 1e-3 or 5e-4

```
first_moment = 0
second_moment = 0
for t in range(num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

5.2.4 적응적 학습률

- 동작 예시　　실시간 애니메이션 http://cs231n.github.io/neural-networks-3
  - [그림 5-13(a)]는 중앙으로 급강하하는 절벽 지형
  - [그림 5-13(b)]는 중앙 부근에 안장점이saddle point 있는 지형

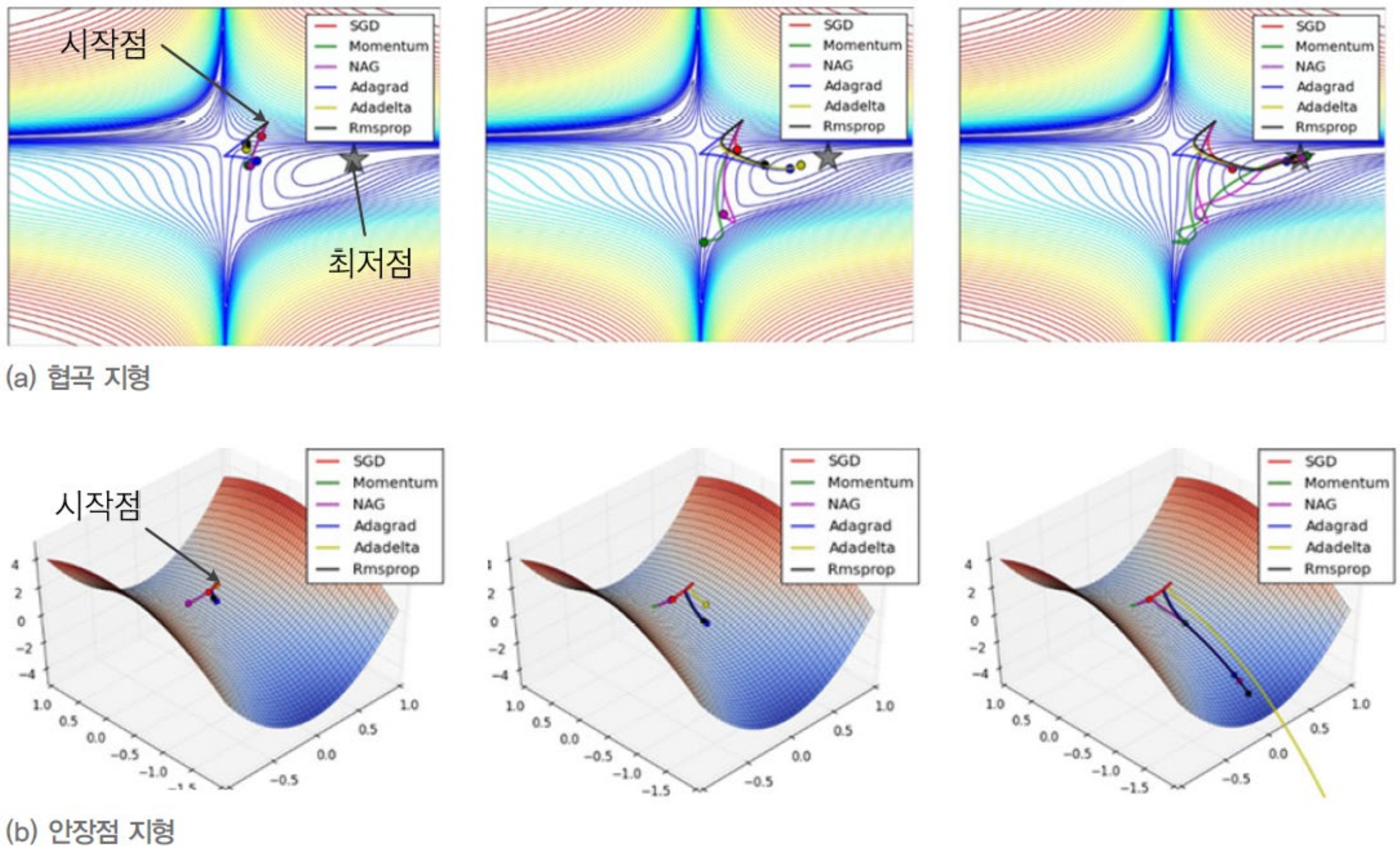
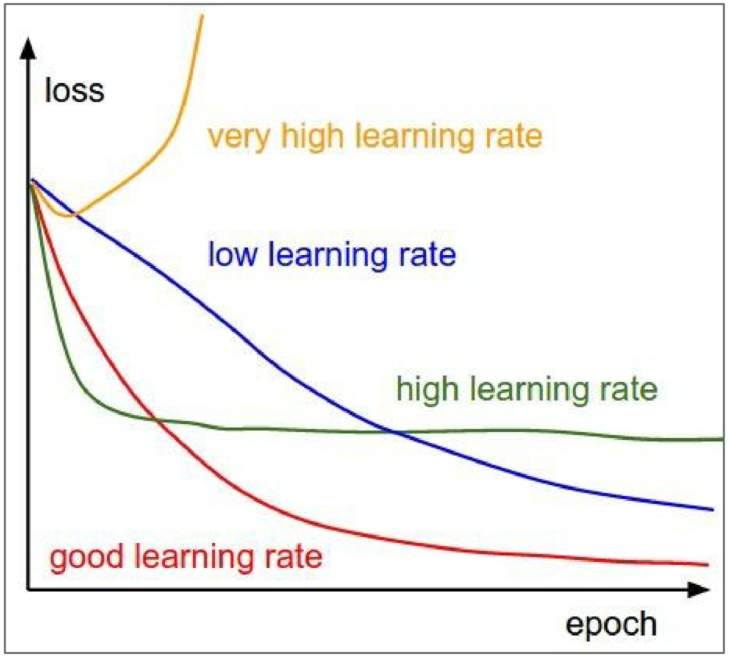


그림 5-13 모멘텀과 적응적 학습률 기법의 수렴 특성을 보여 주는 예제[4]

5.2.4 적응적 학습률



=> Learning rate decay over time!

step decay:
e.g. decay learning rate by half every few epochs.

exponential decay:

$$\alpha = \alpha_0 e^{-kt}$$

1/t decay:

$$\alpha = \alpha_0 / (1 + kt)$$