09 & 10. 경사하강법
(Gradient Descent)

- Momentum
- Netsterov Momentum

학습률 고정

- Adagrad

경사도 다른 변수의
학습률 다르게!

- RMSProp / Adadelta / Adam

Momentum
+ Adagrad

경사 하강법 (Gradient Descent)

– Momentum

• 1st : 하강 방향 ⊖. α는 학습률 (매번 계산되는 단계값을 학습률로 고정! → 이동 방향 제한)

$v^{(1)} = -\alpha g^{(1)}$, 속도 (velocity)

↳gradient

$x^{(1)} = x^{(0)} + v^{(1)} = x^{(0)} - \alpha g^{(0)}$, 경사하강(Gradient Descent)

→ 속도 : 최초의 계획점이 주어지면 거기서의 경사도 계산 후, 주어진 학습률에 의해 하강 방향으로 이동하는 값의 크기

• 2nd :

이전 속도 현재 계획 점에서의 velocity (속도)

$v^{(2)} = \beta v^{(1)} - \alpha g^{(1)} = \beta(\alpha - g^{(0)}) - \alpha g^{(1)}$, 가속도 (Momentum)

Momentum 파라미터

$x^{(2)} = x^{(1)} + v^{(2)}$

• kth :

$v^{(k+1)} = \beta v^{(k)} - \alpha g^{(k)} = \beta(-\alpha g^{(k-1)}) - \alpha g^{(k)}$

$x^{(k+1)} = x^{(k)} + v^{(k+1)}$ → Momentum update formula

→ 최소값이 있는 방향으로 계획점 꾸준히 update

But, 빨리, 큰 step으로 이동 & 최소값 안정적 계산!

⇒ Momentum 방법은 이전 반복에서 얻어진 경사 하강 방향으로의 진행 크기 (Velocity)를 누적!

최초에는 경사 하강법 (Gradient Descent)와 똑같이 계획점 이동

2nd는 이전에 얻어진 velocity를 새로 얻어진 velocity에 누적

완만한 목적함수 경우

⇒ 좀 더 빨리 가고 싶은데 이전에 진행했던 방향 크기 누적해 진행!

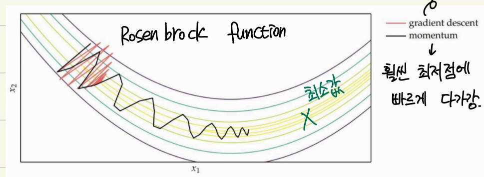
⇒ 경험적으로, β는 1일 때 가속도 커져 1보다 작게! 0.9 쯤 이용

따라서, 실제 hyper-parameter은 학습률 (α)

• $v^{(k+1)} = \beta v^{(k)} - \alpha g^{(k)}$, β=0.9

$x^{(k+1)} = x^{(k)} + v^{(k+1)}$

Gradient Descent는 사이클링 발생.



→ long deep valley 있으면 Momentum 방법 좋음

다른 함수에서는 Gradient Descent와 Momentum 비슷

– Nesterov Momentum

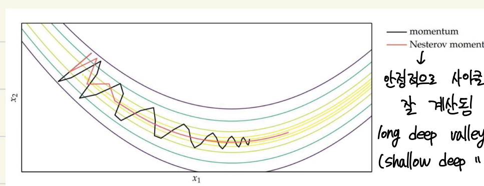
원래 Momentum

: $v^{(k+1)} = \beta v^{(k)} - \alpha g^{(k)} = \beta v^{(k)} - \alpha \nabla f(x^{(k)})$

→ $v^{(k+1)} = \beta v^{(k)} - \alpha \cdot \nabla f(x^{(k)} - \beta v^{(k)})$

$x^{(k+1)} = x^{(k)} + v^{(k+1)}$

Nesterov Momentum이 완만한 경사 방향으로 큰 step size 보이게 됨



long deep valley일 때! (shallow deep 〃)

⇒ Gradient Descent, Conjugate gradient Descent, Momentum, Nesterov Momentum 모두 다변량 변수에 각각 동일한 학습률 부여

분산이 커지며 변화가 큰 변수의 경사도 방향으로 끌려다님

→ 변수 별로 다른 가중치 주어 계산량 줄어들어야 함.

경사 하강법 (Gradient Descent)

- Adagrad : Adaptive subgradient (2011)

각 변수별 다른 gradient

- x_i의 크기가 다르므로 x_i에 서로 다른 α
- $x_i^{(k+1)} = x_i^{(k)} - \left(\frac{\alpha}{\sqrt{s_i^{(k)}} + \varepsilon}\right) g_i^{(k)}$ $s_i^{(k)}$에 따라 변수별 학습률이 다르게 부여

  → $s_i^{(k)} = \sum_{j=1}^{k} (g_i^{(j)})^2$, i번째 변수의 k번째 까지의 제곱합 → 각 변수별 누적

  → $\varepsilon = 1 \times 10^{-8} = 1E-8$

  ↳ 분모가 0이 되는 것 방지

  → 변수들이 gradient 양이 크면 $s_i^{(k)}$ 커지니까 학습률 작아짐
- k가 증가할수록 $\frac{\alpha}{\sqrt{s_i^{(k)}}}$가 지속적으로 작아짐 (decaying)

  ↳ 누적이니까 $\sqrt{s_i^{(k)}}$ 커짐.

  → 업데이트 안되는 문제! (vanishing)

- RMS Prop : Geoffrey Hinton

- 경사도 제곱 크기 유지
- $\hat{s}^{(k+1)} = \gamma \hat{s}^{(k)} + (1-\gamma)(g^{(k)} \odot g^{(k)})$, $\gamma = 0.9$

  gradient 제곱되는 dot product

  ↳ γ와 $1-\gamma$로 해서 합 = 1 $\hat{s}^{(k+1)}$ 커지는 것 막음

  ↓

  gradient의 가중치 줄임 (가중평균)

  β, α → $\gamma, (1-\gamma)$

$$x_i^{(k+1)} = x_i^{(k)} - \frac{\alpha}{\sqrt{s_i^{(k)}} + \varepsilon} \cdot g_i^{(k)}$$
$$= x_i^{(k)} - \frac{\alpha}{RMS(g_i) + \varepsilon} \cdot g_i^{(k)}$$

→ $RMS(g_i)$ 커지면 $\frac{\alpha}{RMS(g_i)+\varepsilon}$ 작아질 위험 有

- Adadelta : Adaptive Learning Rate Methods

- 학습률 : RMS 증가 비율
- $\hat{s}^{(k+1)} = \gamma \hat{s}^{(k)} + (1-\gamma)(g^{(k)} \odot g^{(k)})$, $\gamma = 0.9$

  $x_i^{(k+1)} = x_i^{(k)} - \frac{RMS(\Delta x_i)}{RMS(g_i)+\varepsilon} g_i^{(k)}$

  → $RMS(\Delta x_i)$는 $\hat{s}^{(k)}$와 $\hat{s}^{(k+1)}$의 차이 제곱

  $\frac{\alpha}{\sqrt{s_i^{(k)}}}$가 작아지는 문제 완화.

  변수마다 동일 α 적용되는 문제 완화

- Adam : Adaptive momentum estimation method

- momentum : $v^{(k+1)} = \gamma_v v^{(k)} - (1-\gamma_v) g^{(k)}$
- Squared gradient : $s^{(k+1)} = \gamma_s s^{(k)} + (1-\gamma_s)(g^{(k)} \odot g^{(k)})$

  → $v^{(k+1)}$, $s^{(k+1)}$ 추정식 사용하자!
- momentum correction : $\hat{v}^{(k+1)} = \frac{v^{(k+1)}}{(1-\gamma_v^k)}$ like 평균

  ↳ momentum 이용X.

  k증가 → 분모는 1과 가까운 값
- Squared gradient correction : $\hat{s}^{(k+1)} = \frac{s^{(k+1)}}{(1-\gamma_s^k)}$

  ↳ 값이 커질 위험이 있으니 전체 반복에서 동일한 평균값 추정 like 분산
- Update : $x^{(k+1)} = x^{(k)} - \frac{\alpha \hat{v}^{(k+1)}}{\sqrt{\hat{s}^{(k+1)}} + \varepsilon}$

  ↓

  계산량 多. 안정적으로 빠르게 최소값에 접근해 Computational Cost 줄어들 수 있음