Optimizer에 대해서

작성일시	@2022년 4월 17일 오전 9:48
유형	개념💡
상태	
에픽	
스프린트	
우선순위	
작업	
시작일자	@2022년 4월 17일
제품 관리자	
엔지니어	

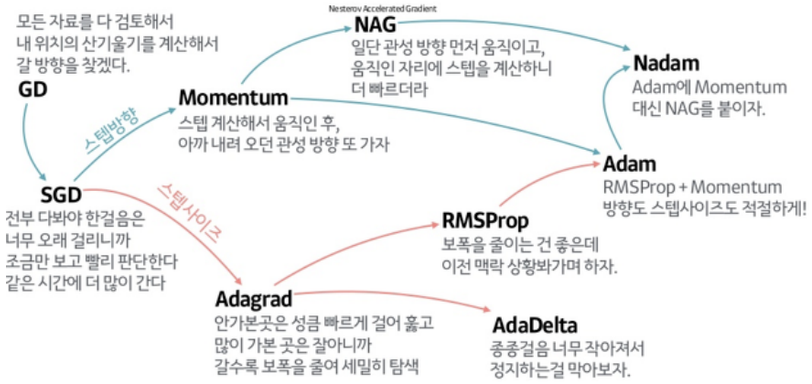
Optimizer의 역할

- Optimizer : 최적화
- 궁극적인 목적은
- 궁극적인 목적은 DB에서 최소의 코스트로 결과를 찾는다.
- 즉, 신경망의 모델이 실제 결과와 예측 결과의 차이를 최소화 시키는 것이다.
- epoch, neuron 수, Dropout 등등 모델이 생성할 때 수많은 parameter를 조정하게 되는데 그 중 가장 드라마틱하고 쉽게 바꿔주는 것이 optimizer이다.

> 💡 ***네비로 Optimizer 이해하기***
> 우리가 처음 가는 길을 운전한다고 가정하자. 이때 네비가 있어서 길을 알려주면 최소한의 실패는 없을 것이다. 하지만 네비도 한 가지의 길만 알려주지는 않는다. '큰 길' 위주로 알려주는 방법도 있고, `최단 거리`로 알려주거나 `고속도로 기반의 빠른 길`, 그리고 `무료 길 위주`로 알려주는 등등 네비 자체적으로 **다양한 알고리즘**이 있어서 사용자는 해당 알고리즘을 선택하고 그 다음 네비가 알려주는 방식대로 운전을 한다. 즉, **네비는 optimizer**와 역할이 동일하며 **네비 안에 다양한 알고리즘은 옵티마이저의 종류**와 대응할 수 있다.

출처 : [머신러닝] 옵티마이저(Optimizer)란?

Optimizer의 종류



출처 : https://www.slideshare.net/yongho/ss-79607172

Gradient Descent

- 경사를 따라 내려가면서 W를 업데이트한다.

$$W := W - \alpha \frac{\partial}{\partial W} J(W)$$

> $J(\theta)$: θ 를 미지수로 갖는 목적함수
> α : learing rate

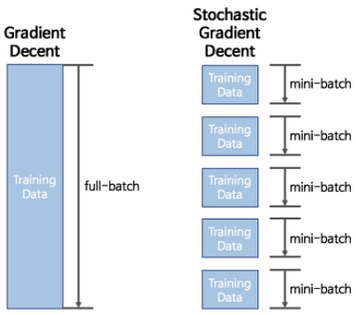
- 극소값을 바로 찾지 않고 step별로 update하는 이유는?

- 대부분의 non-linear regression은 closed form solution이 존재하지 않는다. 즉, 문제에 대한 해답을 식으로 명확하게 제시할 수 없다.
- closed form solution이 존재해도 수많은 parameter가 있을 때에는 GD로 해결하는 것이 계산적으로 더 효율적이다.

Stochastic Gradient Descent

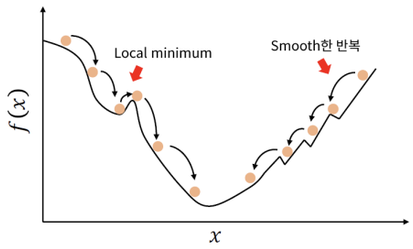
- full-batch가 아닌 **mini-batch로 학습을 진행**하는 것
  - *batch로 학습하는 이유는?*

    *full-batch로 epoch마다 weight를 수정하지 않고 **빠르게** mini-batch로 weight를 수정하면서 학습하기 위해서*
- GD vs. SGD



출처 : https://seamless.tistory.com/38

Momentum



- SGD의 **높은 편차를 줄이고 수렴을 부드럽게 하기 위해** 고안
- SGD + **관성**
- 현재의 batch뿐만 아니라 **이전의 batch 학습 결과도 반영**한다.
- γ는 현재 기울기뿐만 아니라 현재 속도를 함께 고려해 이동 속도를 나타내는 `momentum term`이다.

  $v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$

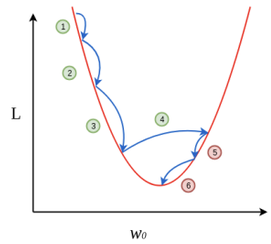
  > γ : Momentum 계수
  > η : learning rate
  > v_t : t번째 step에서의 x의 이동벡터

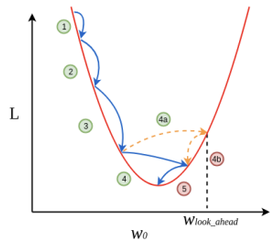
- 이전 gradient들의 영향력을 매 update마다 γ배씩 감소

Nesterov Accelerated Gradient

- momentum이 너무 높으면 알고리즘이 minima를 놓치고 건너뛰어버릴 우려가 있다.
- Momentum vs. NAG
  - 모멘텀에서 이동벡터를 계산할 때는, 현재 위치에서 기울기와 momentum step을 독립적으로 계산하고 합치지만, NAG는 momentum step을 고려해서 현재의 기울기에 gradient step을 이동한다.

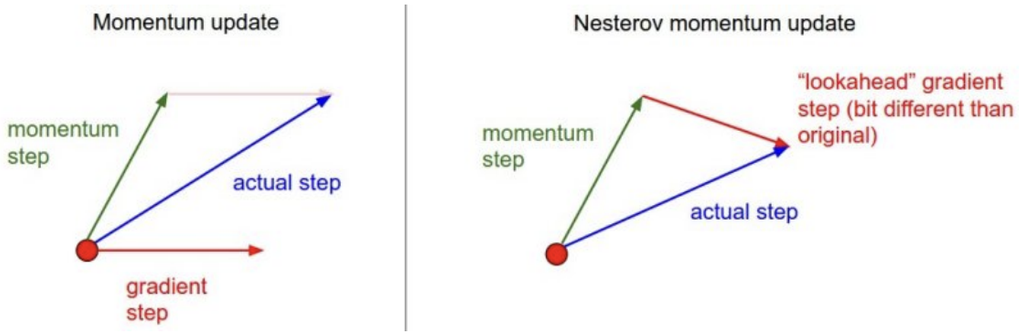
    $v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta_t - \gamma v_{t-1})$





(a) Momentum-Based Gradient Descent (b) Nesterov Accelerated Gradient Descent

$\bigcirc \Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Positive(+)}$ $\bigcirc \Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Negative(-)}$



Adagrad : Adaptive Gradient

- 지금까지의 optimzier의 단점은 learning rate가 모든 parameter와 각 cycle에 대해 일정하다.
- Adagrad는 각 parameter와 각 step마다 학습률 η를 변경할 수 있다.
- second-order 최적화 알고리즘의 유형으로, 손실함수의 도함수에 대해 계산된다.

> 지금까지 많이 변화하지 않은 변수들은 step size를 크게 하고, 지금까지 많이 변화했던 변수들은 step size를 작게 하자.
>
> 자주 등장하거나 많이 변화한 변수들은 optimum 값에 가까이 있을 확률이 높기 때문이다.

$G_t = G_{t-1} + (\nabla_\theta J(\theta_t))^2$
$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} \cdot \nabla_\theta J(\theta_t)$

- 단점
  - 이계도함수를 계산해야 하기 때문에 계산 비용이 많이 든다.
  - 학습을 진행하면 진행할 수록 학습률이 줄어든다.

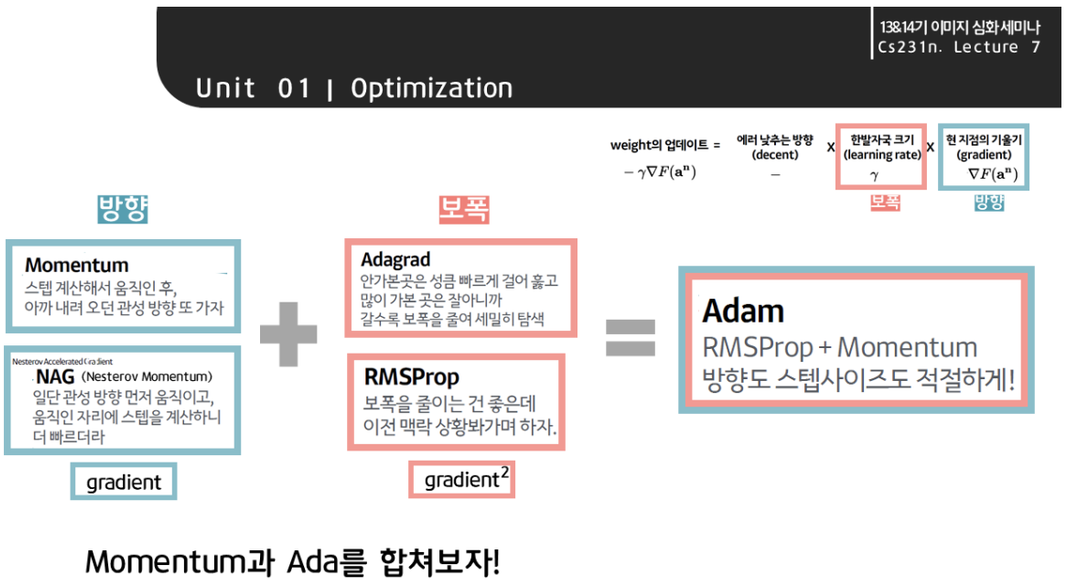
RMSProp

- Root Mean Square Propatation
- Adagrad는 간단한 convex function에서는 잘 동작하지만, 복잡한 다차원 곡면 함수를 상대적으로 잘 탐색하지 못한다. 또한 기울기의 단순 누적만으로는 충분하지 않다.
- RMSProp는 기울기를 단순 누적이 아닌, 지수 가중 평균 `Exponentially weighted moving average`를 사용하여 최신 기울기를 더 크게 반영되도록 하였다.

$g_{ij}^{(t)} = \beta g_{ij}^{(t-1)} + (1-\beta)(\frac{\partial L}{\partial w_{ij}^{(t)}})^2$
$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \frac{\eta}{\sqrt{g_{ij}^{(t)}}} \frac{\partial L}{\partial w_{ij}^{(t)}}$

Adam : Adaptive Moment Estimation

- Adam = RMSProp + Momentum



- Adam은 local minima를 뛰어넘을 수 있다는 이유만으로 빨리 굴러가는 것이 아닌, minima의 탐색을 위해 조심스럽게 속도를 줄이고자 하는 것이다.
  - RMSProp와 같이 root mean squared gradients를 저장할 뿐만 아니라, 과거 gradient의 decaying avrage도 저장한다.
- 장점
  - loss의 최솟값으로의 빠른 수렴
  - vanishing learning rate 문제 해결
  - high variance 문제 해결
- 단점
  - 계산 비용이 많이 든다.

출처 : [ML] 신경망에서의 Optimizer - 역할과 종류

출처 : 딥러닝 Optimization 함수 정리

출처 : 옵티마이저 Optimizer 정복기 (부제: CS231n Lecture7 Review)