Gradient Descent Optimizer

Cost=Loss=Error

parameter θ

이어서, 16 x 16 크기의 손글씨로 쓴 숫자 이미지의 픽셀 정보가 input으로 주어졌다고 할 때 10×1 길이의 행렬

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix}$$

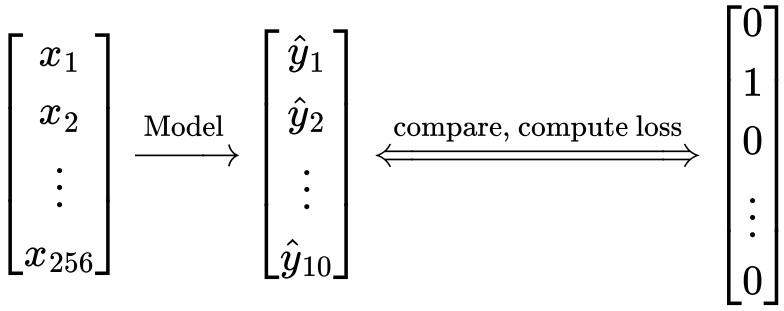
형태의 ouput 을 도출하는 모델을 생각하자. 만약 입력된 손글씨가 "2"라면 one-hot encoding을 이용하여

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

와 같은 형태의 target을 갖는다.

주어진 network parameter들을 모은 $\theta = \left\{W^1, b^1, \ldots, W^L, b^L\right\} \in \mathbb{R}^{2L}$은 각각의 sample data (이 경우 이미지와 레이블의 쌍) 에 대해서 **cost value** $L(\theta)$를 가진다. 이는 output과 target의 MSE 혹은 cross entropy를 의미한다. 이를

도식화하여 나타내면,



Total Cost

traning 에 쓰인 모든 sample data (즉 training data) 에 대해서, 그 개수가 N 개라 한다면 각각의 cost value $L^1(\theta), L^2(\theta), \cdots, L^N(\theta)$ 들을 총합한 값을 Total cost라 한다.

$$C(\theta) = \sum_{n=1}^{N} L^n(\theta)$$

즉 total cost $C(\theta)$ 는 model 의 현재 파라미터들에 의존하며, 클 수록 현재 네트워크의 성능이 안 좋다고 평가할 수 있는 척도로서 작용한다. 따라서, 인공지능의 total cost를 최소화하는 것을 목적으로 잡는 것이 바람직하다.

Gradient Descent

인공지능의 total cost를 최소화되는 지점은 무슨 지점일까? 우리는 적당히 응용수학적으로 생각했을 때, 최소의 total cost를 갖는 지점은 곧 θ의 각 원소들에 대해 계산한 gradient $\nabla_\theta C\left(\theta\right) = 0$ 이 되는 지점임을 알 수 있다. 이러한 1차 derivative/gradient 을 필요로 하는 최적화 알고리즘을 **first-order iterative optimization algorithm** 이라 한다.

심층학습의 경우 그 total cost의 그래프가 적당히 예쁘다고(!!) 가정할 수 있어, 다음과 같은 이산적인 과정을 통해 그 local minimum을 구할 수 있다.

(1) 초기 시작점 θ^0 을 정한다.
(2) 각 time step θ^t 에 대해, $-\nabla_\theta C\left(\theta^t\right)$ 를 계산한다.
(3) 해당 그래디언트의 방향에 특정 계수 η를 곱해 피드백해준다.

$$\nabla = \frac{\partial}{\partial x}\hat{i} + \frac{\partial}{\partial y}\hat{j} + \frac{\partial}{\partial k}\hat{z}$$
$$\nabla f = \frac{\partial f}{\partial x}\hat{i} + \frac{\partial f}{\partial y}\hat{j} + \frac{\partial f}{\partial k}\hat{z}$$

$$\boxed{\nabla_\theta C\left(\theta^t\right) = \begin{bmatrix} \frac{\partial C\left(\theta^t\right)}{\partial w_1} \\ \frac{\partial C\left(\theta^t\right)}{\partial w_2} \end{bmatrix}}$$

Batch Gradient Descent

매 학습(epoch)마다 전체 데이터셋에 대해 그래디언트를 계산
$\theta^{t+1} = \theta^t - \eta\nabla_\theta C\left(\theta^t\right)$의 update를 매 time step마다 실행.
$\Leftrightarrow w_i(t+1) = w_i(t) - \eta\frac{\partial C}{\partial w_i}$ for $\forall w_i$.

장점

- 아래로 볼록한 cost surface에 대해 전역적 최소값으로의 수렴이 보장됨.
- 노이즈에 대해서 강함

단점

- 큰 메모리 용량 필요
- 느림
- 실시간으로 변화하는 데이터에 대해 적응하지 못함

Stochastic Gradient Descent

매 단일 training data 쌍에 대해서 그래디언트를 계산
마찬가지로 $\theta^{t+1} = \theta^t - \eta \nabla_\theta L^n \left(\theta^t\right)$ 의 업데이트를 실행

장점

- 빠르기 때문에 online learning이 가능

단점

- 노이즈에 민감함
- SGD fluctuation

Minibatch Gradient Descent

전체 training data를 특정한 단위의 minibatch들로 분리.
(하나의 batch는 m 개의 임의의 샘플 데이터)

분리된 minibatch들에 대해서 순차적으로 그래디언트를 계산
마찬가지로 $\theta^{t+1} = \theta^t - \eta \nabla_\theta L^n \left(\theta^t\right)$ 의 업데이트를 실행

장점

- 충분히 빨라 online learning 가능
- 노이즈의 영향이 충분히 작음

단점

- minibatch의 크기가 정해줘야 하는 변수.

Gradient Descent Optimizer

$\theta^{t+1} = \theta^t - \eta \nabla_\theta C \left(\theta^t\right)$ 에서의 Learning rate η 의 값을 조절.
learning rate: 가는 걸음의 속도 (θ는 weight들)

너무 느리면 신중하지만 시간이 오래 걸리고
너무 빠르면 헛걸음 + 돌아갈 수 없는 강을 건넘

Learning rate decay: 초반에는 빨리빨리 움직이다가 loss 최소에 가까워질수록 느리고 신중하게 움직이기 -> 시간에 대해서만 조절하지 않고 주변 상황에 따라 유동적으로 조절하면?

Momentum

$v^{t+1} = \gamma v^t - \eta \nabla_\theta C\left(\theta^t\right)$
$\theta^{t+1} = \theta^t + v^{t+1}$
관성 반영. 이전의 운동을 같이 반영한다.
SGD는 local minima에서 진동가능 -> 관성으로 넘어감
SGD는 골짜기를 지그재그로 이동가능 -> 잠시 후면 관성이 아래쪽을 향함

Adagrad (Adaptive gradient)

$\theta^{t+1} = \theta^t - \frac{\eta}{\sqrt{G_\theta^t + \epsilon}} \nabla_\theta C\left(\theta^t\right)$ where $G_\theta^t = \sum_{i=0}^t \left(\nabla_\theta C\left(\theta^i\right)\right)^2$
learining rate를 weight마다 다르게 업데이트하자.
node는 각각의 feature를 나타냄. 따라서 더 직접적인 feature일수록 거길 지나가는 weight가 클 것임. 그 값들이 코스트에 영향을 많이 미쳤을 것이다. -> 이미 optimum값에 가까이 갔을 확률 높다. 즉 G의 값이 클 것이다. 따라서 learning rate는 감소. 근데 time step 커질 때마다 G 커진다. learning rate 너무 작아짐

RMSProp (Root Mean Square Propagation)

$\theta^{t+1} = \theta^t - \frac{\eta}{\sqrt{G_\theta^t + \epsilon}} \nabla_\theta C\left(\theta^t\right)$ where $G_\theta^t = \gamma G_\theta^{t-1} + (1-\gamma)\left(\nabla_\theta C\left(\theta^t\right)\right)^2$
최근에 가까울수록 중요 앞의것 감소+ 예전거일수록 r^n(1-r) 에서 n증가. 문제 해결 + 최근반영

Adam (Adaptive moment estimation)

$\hat{m}_\theta = \frac{m_\theta^{t+1}}{1-(\beta_1)^{t+1}}$ where $m_\theta^{t+1} = \beta_1 m_\theta^t + (1-\beta_1)\nabla_\theta C\left(\theta^t\right)$

$\hat{G}_\theta = \frac{G_\theta^{t+1}}{1-(\beta_2)^{t+1}}$ where $G_\theta^{t+1} = \beta_2 G_\theta^t + (1-\beta_2)\left(\nabla_\theta C\left(\theta^t\right)\right)^2$

$\theta^{t+1} = \theta^t - \eta \frac{\hat{m}_\theta}{\sqrt{\hat{G}_\theta + \epsilon}}$

=Momentum+RMSProp.

갈수록 분모 $1-\beta^{t+1}$ 증가 -> Learning rate decay

RMSProp 식에서 사용한 테크닉 반영

- 엡실론 넣는건 0이 되지 않기 위함임