PyTorch로 딥러닝 제대로 배우기

- 기초편 -

Part8. Loss and Optimizer

강사: 김 동 희

목차

I. Losses

1. MSE(Mean Squared Error)

❑ L1 Loss

- $l_n = |x_n - y_n|$
- 가장 natural way of measuring the distance
- 벡터의 모든 구성 요소는 동등하게 가중 (weighted equally)
- ex) |3-1| = 2, |11-1| = 10

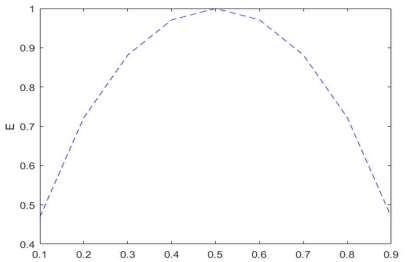
❑ MSE; L2 Loss

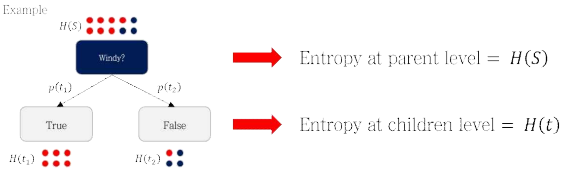
- $l_n = (x_n - y_n)^2$
- 벡터의 각 구성 요소를 제곱함으로써 error값에 따라 가중치가 다름
- 즉, error값이 클 수록 더 강하게 반응
- ex) $(3-1)^2 = 4$, $(11-1)^2 = 100$

2. CrossEntropyLoss

❑ Entropy

- 엔트로피는 정보의 불확실성을 의미
- 사건이 발생할 확률이 같은 경우(P(0.5)) 불확실성이 가장 높음 = entropy가 가장 높음
- 동전을 던졌을 때, 앞면이 나올지 뒷면이 나올지 예측하기 어려움
- 6면 주사위에서 5면이 1이고 1면이 6인 경우라면, 주사위를 한번 던졌을 때 1이 나올 확률이 높음(P(5/6))
  → 예상 가능함





출처: https://tyami.github.io/machine%20learning/decision-tree-2-ID3/

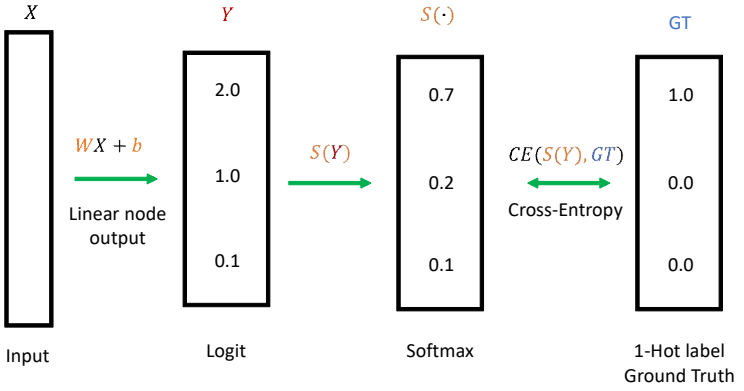
2. CrossEntropyLoss

❑ Cross Entropy

- $H(p,q) = -\sum_{c=1}^{C} \mathrm{p}(y_c)\log(q(y_c))$
- 예측 모델(q)은 실제 분포 p를 모르는 상태에서 구분 분포를 따라가고자 함
- 즉, entropy를 줄이는 쪽으로 minimize 함 (불확실성을 줄이겠다)
- Loss 함수에서는 **분포의 차이를 줄이는 것이 목표**
- PyTorch에서 CrossEntropyLoss는 Softmax가 포함되어 구현
- $l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \cdot 1\{y_n \neq ignore_{index}\}$

2. CrossEntropyLoss

❑ **Cross Entropy**



3. Kullback-Leibler divergence loss

❑ KLDivLoss

- 서로 다른 두 분포의 차이(dissimilarity)를 측정하는데 쓰이는 measure
- $D_{KL}(P||Q) = \sum_{x\in\chi} P(x) log\left(\frac{P(x)}{Q(x)}\right) = -\sum_{x\in\chi} P(x)[\log(Q(x) - log\big(P(x)\big)] = H(P,Q) - H(P)$
- Cross-entropy는 entropy 보다 항상 0보다 크거나 같은 값을 같기 때문에 P의 entropy(정보량)과 P,Q의 cross-entropy가 같을 때 최소 값을 가짐
- P를 실제분포라하고 Q를 예측 분포라고 한다면, H(P)는 항상 상수 값을 갖기 때문에 Cross-entropy를 최소화하는 방법으로 loss를 줄임
- 단, $D_{KL}(P||Q) \neq D_{KL}(Q||P)$

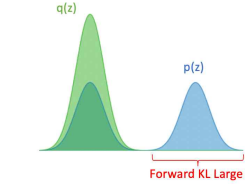
https://3months.tistory.com/436

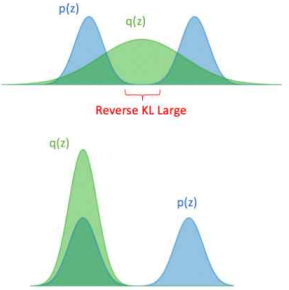
https://towardsdatascience.com/why-is-cross-entropy-equal-to-kl-divergence-d4d2ec413864

3. Kullback-Leibler divergence loss

❑ KLDivLoss

- $D_{KL}(P||Q) = H(P,Q) - H(P) = Forward$
- $D_{KL}(Q||P) = H(Q,P) - H(Q) = Reverse$



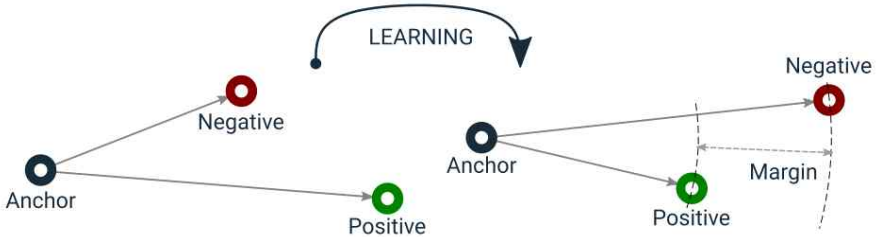


http://adamlineberry.ai/vae-series/kl-divergence

4. TripletMarginLoss

❑ **Triplet Margin Loss**

- Anchor를 기준으로 Positive 또는 Negative하게 학습을 유도
- Anchor에 positive한 샘플들은 가까이에, Anchor와 negative 한 샘플은 멀리 embedding 되도록 유도

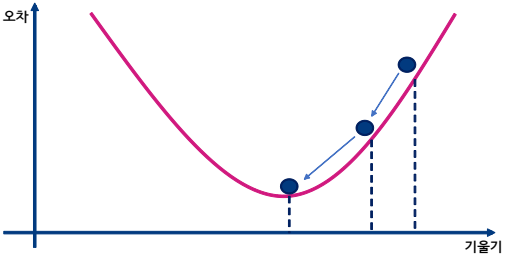


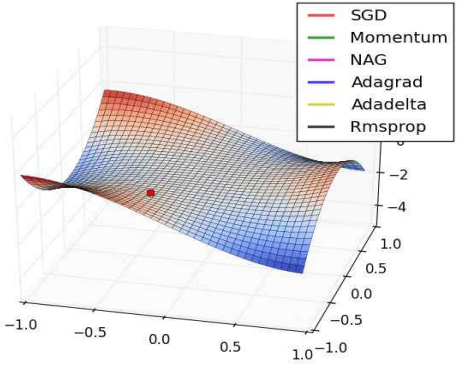
II. Optimization Functions

1. Optimization

❑ Overview

"오차가 가장 작은 점을 찾자"



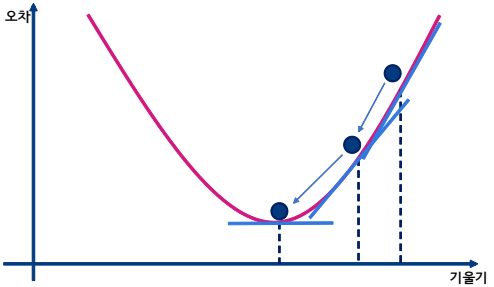


But, 현재 point가 minima인 것을 어떻게 알지?

1. Optimization

❑ 미분의 개념

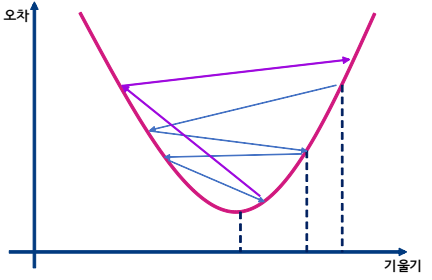
- 미분은 순간 변화율
- $\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$
- x의 작은 변화가 함수 f(x)를 얼마나 변화시키느냐?
- 즉, 기울기를 의미함
- 기울기가 0에 가까울 수록, 최저점을 의미함

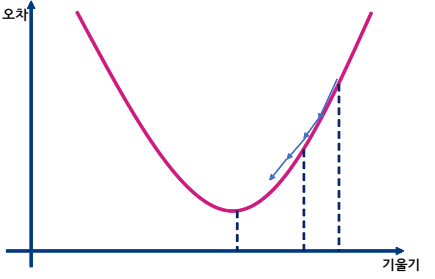


1. Optimization

❑ Learning rate

- 얼마만큼 움직일 것인지 결정하는 문제가 학습률(Learning rate)라 함
- 학습률이 너무 크면 최저점을 찾지 못하고 발산하고,
- 학습률이 너무 작으면 최저점을 찾는데 오래 걸림

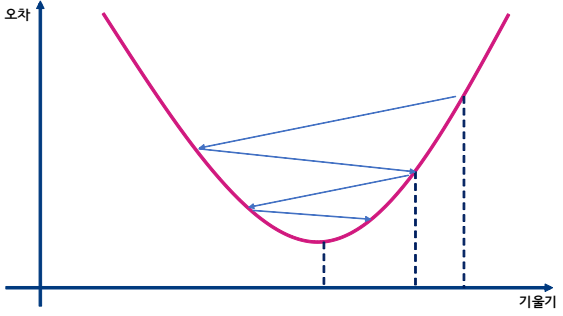




2. Stochastic Gradient Decent

❑ Gradient Decent

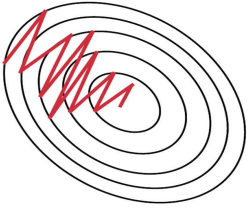
- $W \leftarrow W - \eta \frac{\partial L}{\partial W}$
- 일정한 거리를 움직이면서 오차 함수에서의 최저점을 찾는다.
- 경사를 내려가는 것과 같다고 하여, Gradient Decent라 부른다.



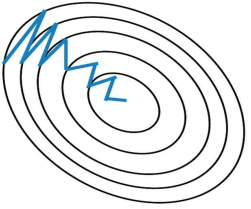
2. Stochastic Gradient Decent

❑ Momentum

- SGD는 상당히 비 효율적인 움직임을 보임
- $v \leftarrow \alpha v - \eta \frac{\partial L}{\partial W}$
- $W \leftarrow W + v$
- 기울기 방향으로 물체가 가속된다는 물리 법칙을 적용



Stochastic Gradient Descent **withhout** Momentum



Stochastic Gradient Descent **with** Momentum

3. AdaGrad

❑ AdaGrad

- 학습률을 효과적으로 변경해야하는 필요성이 대두됨
- $h \leftarrow h + \left(\frac{\partial L}{\partial W}\right)^2$
- $W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$
- 매개변수의 원소 중, 변화가 큰 원소는 학습률을 낮게 적용
- 즉, 가중치가 크게 변한 파라미터는 next step에서 적게 움직임

4. RMSProp

❑ RMSProp

- AdaGrad 알고리즘은 기울기를 계속 제곱하기 때문에 결국 나중에는 거의 변하지않음
- 이를 보완하기 위한 알고리즘
- 지수 이동평균을 사용하여 과거 기울기는 서서히 잊고, 최신 기울기 정보를 크게 반영
- $\boldsymbol{G} \leftarrow \gamma \boldsymbol{G} + (1-\gamma)\left(\frac{\partial L}{\partial \boldsymbol{W}}\right)^{\mathbf{2}}$
- $\boldsymbol{W} \leftarrow \boldsymbol{W} - \frac{\eta}{\sqrt{G+\epsilon}}\frac{\partial L}{\partial \boldsymbol{W}}$

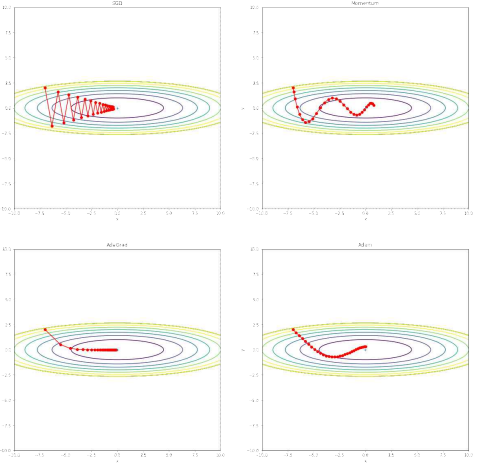
5. Adam (Adaptive Moment estimation)

❑ Adam

- Momentum + RMSProp Optimizer
- 기울기의 지수 평균과 제곱값의 지수 평균을 저장

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)



6. 정리하기

- **Optimization**은 오차가 가장 작은 점을 찾는 수학적 모델링
- 얼마만큼 이동 할 것인지, 어느 방향으로 이동할 것인지 결저하는 것이 주요 문제
- 상황에 따라 다양한 **Optimization** 알고리즘을 시도해 볼 가치는 있음
- 웬만한 상황이면 **Adam or RMSProp** 알고리즘이 좋다.

감사합니다.