Regularization

overfitting = train data의 specific pattern 학습 (≒ Noise)

이는 model의 capacity에 매우 밀접한 관계(영향)을 갖게 된다.

"언제까지 general trend를 학습하는지 알수없다") model capacity control

이때 Regularization을 추가해준다. (성능개선↓이면 w를 줄여라)

min $(Y-X\theta)^T(Y-X\theta)+\lambda\|\theta\|_2^2$ 으로 penalty term을 추가해준다.

L2 ⇒ w가 0이 되진 않는다.

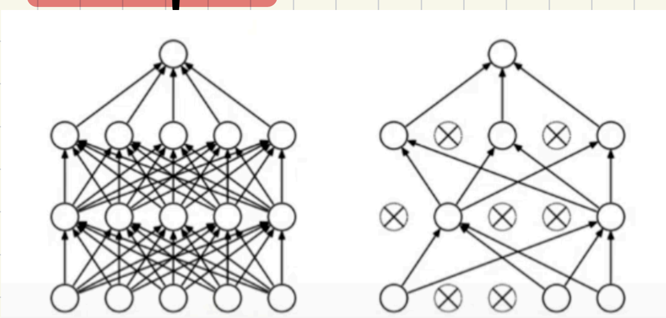
λ가 ↑할수록 w의 값들이 더욱 작은 원의 접선으로 mapping된다.

deep learning은 overfitting에 매우 취약하다.

(1) weight decay

$\Omega(w)=\sum\sum w_{ij}^2$ or $\Omega(w)=\sum\sum|w_{ij}|$ 을 더해준다.

(2) Drop out



(a) Standard Neural Net (b) After applying dropout.

모든 node 사용 X

특정 node들을 확률적으로 사용한다.

⇒ 이미지에 임의의 masking 처리한다. ⇒ 일반화 성능↑

Because 다양한 특징을 골고루 학습하도록 유도.

node1 n2 ... n 각 node에 맞는 리스트

0.4 0.2 ... 0.8 설정해서

가중치가 0.5 이상 node는 1 else 0으로 만들어준다.

그래서 gradient가 1/2 계산 사용했기에 scale이 1/2정도 낮게 학습한다.

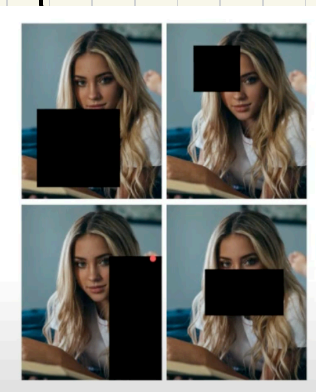
그래서 test 시 가중치 scale들은 2배 증가시켜준다.

(test시에 알 경우 p를 무조건 알아야함)

test code 유지 ⇒ 처음부터 train 시에 p로 나눠서 gradient를 계산해준다.

(3) Cutout

Input data 자체에서 Masking 처리를 진행한다.



⇒ Input시 적용가능해서 train 시간↑의 단점.

(4) Early stopping

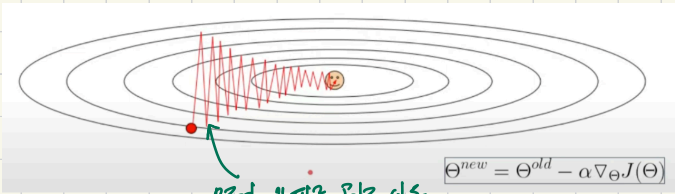
optimization

SGD 문제점 : (1) 느리다.

(2) saddle point 처리에 취약.

(3) mini-batch가 Big data set에 안좋다.

↳ Batch가↑ 일수록 무조건 성능이 올라간다.



무조건 가파른 곳으로 이동.

(2) Momentum (관성)

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

Momentum : $V_{t+1} = \rho V_t - \alpha\nabla f(x)$

이전에 사용된 gradient.

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

ρ는 이전 관성을 얼마나 더 사용할지.

(3) Adagrad.

⇒ 이전까지 움직인 방향의 gradient의 sum을 나눠준다.

⇒ 이미 빠르게 내려온 방향에 대해 브레이크를 잡아준다.

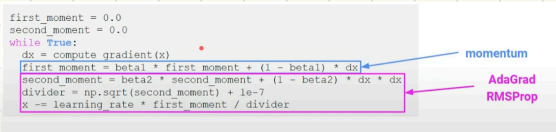
↳ 결국 다 0이되서 이동 거의 X

개선

(4) RMSprops

⇒ 이전 모든 gradient가 아니라 window 처럼 특정 구간에 대해서만 gradient를 sum.

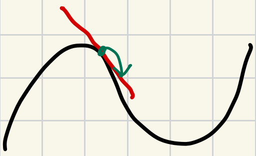
(5) Adam



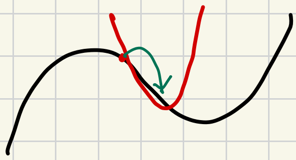
```
first_moment = 0.0
second_moment = 0.0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  divider = np.sqrt(second_moment) + 1e-7
  x -= learning_rate * first_moment / divider
```


momentum + RMSprops이다.

(1) first order optimization



(2) Second order optimization



⇒ 기울기를 구하는 횟수↓ But 1개의 Hessian gradient를 구하는 시간이 매우 오래걸린다.

Batch Normalization

W 초기화 ⇒ zero mean unit variance 설정!

Layer가 ↑ 일수록 적정 초기화 setting이 불리된다.

↳ 이를 해결 "Batch Norm"

mini-batch 마다 $\hat{x}^{(k)} = \dfrac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{k}]}}$ 강제로 가중치를 변형.

↳ 그냥 강제 처리해도 되나? NO.

가중치 처리 후 다시 복구시켜준다. ⇒ 복구시키는 learnable parameter를 data로부터 학습하도록 하는 Idea

$$W \Rightarrow y \quad \underbrace{\frac{y-\mu}{\sigma}}_{\text{Batch Norm}} \Rightarrow \hat{y} \quad \underbrace{\gamma\hat{y} + \beta}_{\text{복구}} \Rightarrow Loss$$

] activation에 들어가기 전에 넣게 된다.

But test 시에는 input이 1개이면 μ와 σ가 1개에 대해서만 정의 ∴ 0이된다.

그래서 train시에 사용한 μ와 σ들은 그대로 사용!

"zero mean unit variance를 모든 layer에서 유지 가능하도록 하는 장치"