

답러닝 2주차

* Overfitting을 막는 regularization

① Regularization? → Main purpose is avoid 'Overfitting'

② 그렇다면 Overfitting이란? ⇒ 학습data를 너무 맹신하는 것.

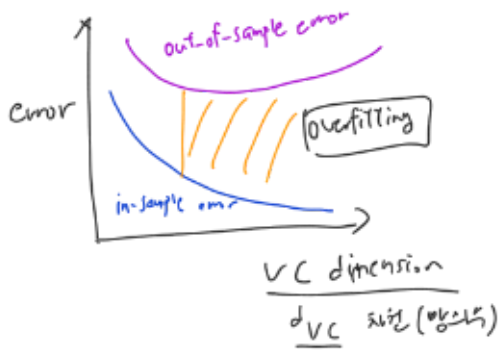
↳ 너무 train data만 잘 맞추는 나머지 test data에서는 제대로 맞추지 못하는 현상...?

⊗ underline (아래선) functions를 실제 우리는 알지 못하기 때문이

프로시 → train set error와 test set error의 차이를 통해 해결해나가야 한다.

대부분의 overfitting은 data noise로 인해 발생.

↳ random measurement error



⊗ Prevent Overfitting

① 더 많은 data 모으기

② 적절한 능력을 갖는 model을 사용

③ 앙상블 (Ensemble) → 다양한 모델들의 결과를 평균을 낸다

ex) data 1000개이면

↓

이제 overfitting과는
안 관련

5개 모델 생성하고

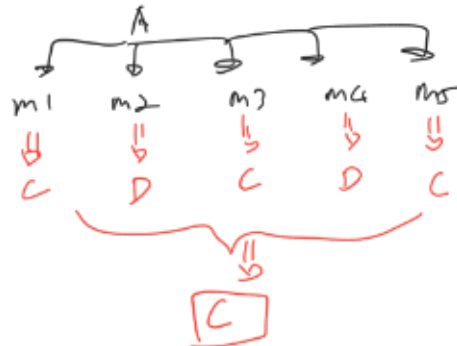
m_1 m_2 m_3 m_4 m_5

↓

data 8000개 각각 학습

test data A를
5개 모델에
input

output이
가장 많이 나오는 것
선택



④ Dropout, Drop Connect, or Batch Norm

* Early Stopping → validation error가 감소하다가 다시 증가하면 학습 Stop

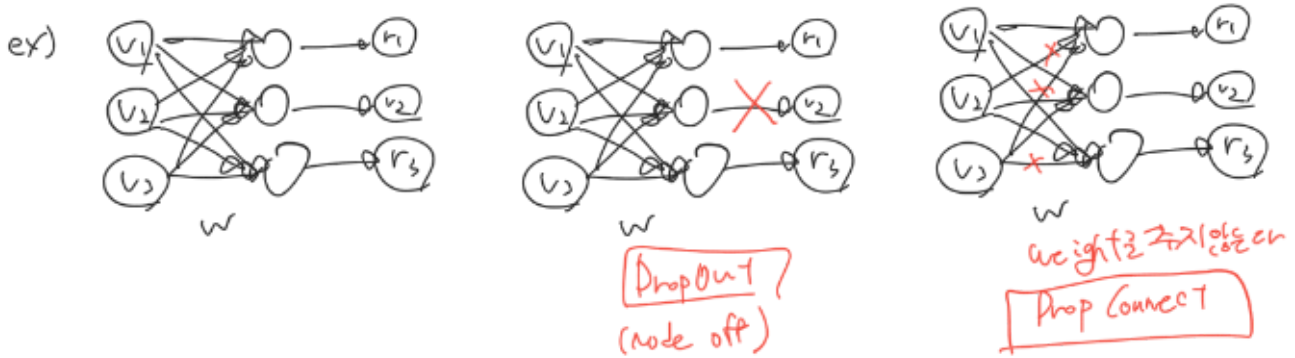
← ... 이걸 limit

* Weight-decay \rightarrow parameter \rightarrow $E(w) = E_0(w) + \frac{1}{2} \lambda \sum_i w_i^2$

* Drop out

\rightarrow 각 layer (층)에서 몇몇 Node를 off 시킴 (확률 시에만)
random

* Drop Connect \rightarrow Node를 off 하는 것이 아니라 Weight를 Unconnect



Batch Normalization

\rightarrow learning rate를 늘려줌.

$$\mu_B = \frac{1}{m} \sum_{i=1}^m x_i \quad (\text{mini batch mean}) \text{ 평균}$$

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2 \quad // \text{ mini-batch variance}$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i = \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale \& shift}$$

* Batch Normalization을 통해 Overfitting 막기 쉽다.
Regularization

* 조금씩 조금씩 작게 \rightarrow 크게 늘려가자.

$$\Rightarrow \boxed{128} \rightarrow \boxed{\begin{matrix} 128 \\ 128 \end{matrix}} \rightarrow \boxed{\begin{matrix} 256 \\ 256 \end{matrix}}$$

* 기계학습의 generalization의 최선은 'more data'를 갖는 것이다.

+ take data를 추가해 줘도 됨

* 중간중간 noise를 집어넣음.

* Semi-Supervised Learning \rightarrow Unsuper + Super

Unsupervised = input만 갖고 있음
Supervised = input, output 둘다 갖고 있음.

* Multi-task Learning

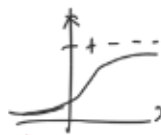


* parameter sharing : ex) CNN

* Sparse representation \rightarrow
 Sparse weight \rightarrow weight면은 0이 많음.
 Sparse activation \rightarrow 결과면 0이 많음

* ReLU (Rectified Linear Unit)

\rightarrow sigmoid 함수가



0과 1 사이에서 변화폭이 작아 gradient Descent를 이용해서

계속 Backpropagation을 layer를 지나면서 하게되면

가속도가 사라지는
현상

Gradient Vanishing 문제

$\Rightarrow 0.8 \times 0.8 \times 0.8 \Rightarrow 0.8$ 보다

점점 작아져서 0이 수렴하게 됨.

* ReLU는 input이 0보다 작으면 0.

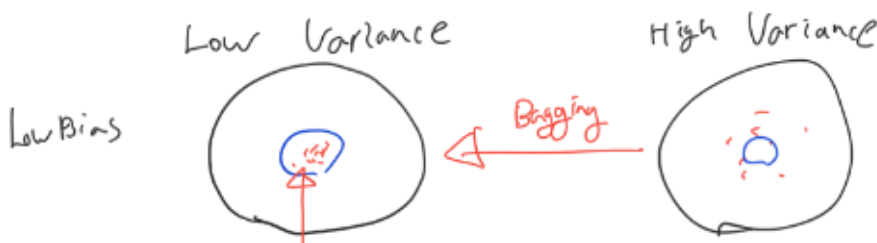
0보다 크면 그대로

$$f = \begin{cases} 0 & (x < 0) \\ x & (x \geq 0) \end{cases}$$

Gradient vanishing을 막기 위해 같은 다층 layer로 대체로 넘긴다.

* ReLU의 이점 \Rightarrow sparse activation (출력이 0이어서 부분 활성화 가능)

Variance \Rightarrow 분산



High Bias



* Bagging = High 편향을 Low 편향으로 만든다. (dataset을 multiple set으로 나눔)

* Boosting \Rightarrow 약한 모델을 계속 붙여서 upgrade 시킴.

A에서 이정도면 B를 붙여서 다시 학습, $\dots \rightarrow A+B \dots$
(A+B)

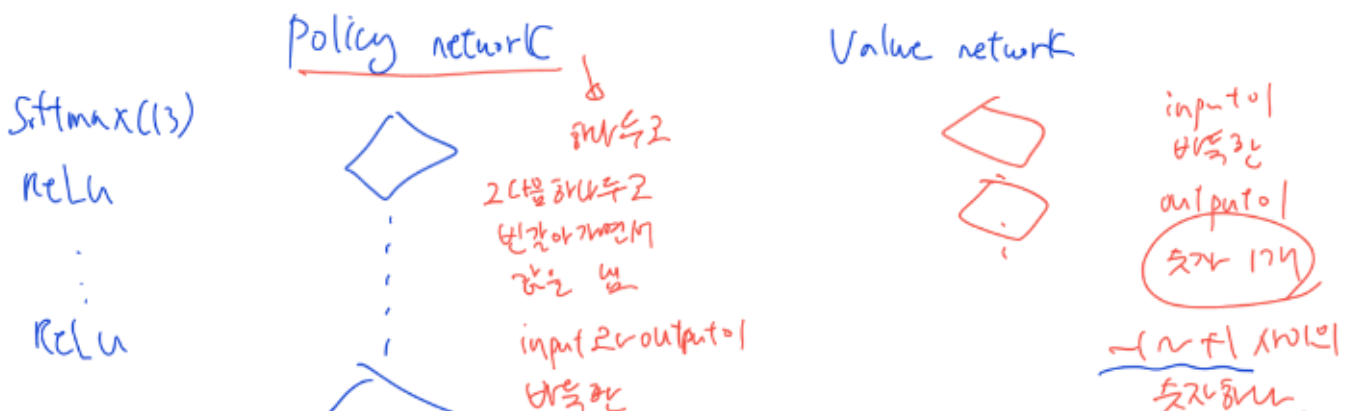
* Adversarial Training \Rightarrow 인강이 웃으면 작은 noise를 주어서
output이 완전 달라짐
개리가 매우 가파르다.

* 3강 Nature 논문으로 살펴보는 ALPHAGO 알고리즘.

M C T S


* Monte - Carlo Tree Search : 나와 상대방이 번갈아가면서 Game
= Selection \rightarrow Expansion \rightarrow Simulation \rightarrow Backpropagation

Value network = 0 RL policy를 통해서만 얻어진다.
알파고 가 생각하는 판의 가치





마지막 output = 그때 self play 결과를 사용



* RL policy network \Rightarrow 알파고끼리 대결시켜서

이거면 이긴판에 대해서는 + (positive) 값으로 저장
 \hookrightarrow 1로 두고 현재판 action 을 MAX화시킨다.
 진다면 진판의 기록은 - (negative)로 값으로 저장
 \hookrightarrow -1로 두고 현재판 action 을 min화한다.

(-값이므로 최소화해야 이길 수 있음)

* Value network \Rightarrow $\begin{cases} \text{win} : +1 \\ \text{lose} : -1 \end{cases}$

\hookrightarrow RL policy + self play result = value net
 (알파고끼리 대결)

문제 카드로 search tree

MCTS loop

player 2명이 한판씩 번갈아서 치는



다음과, 2리봉두에 대한

이길지, 점수의 경우의 수를 계산

① Selection



② Expansion



③ Evaluation



④ Backpropagation

* MCTS 조건 1) Min, Max가 존재

2) 게임 규칙 존재

3) 게임 길이 제한

알고리즘 = ① Selection \rightarrow 어떤 자식 노드로 결정을 선택 (내가 이길 수 있는)

② Expansion \rightarrow 어떤 노드를 선택한 후 그 노드가 leaf이면 자식 노드 생성
 = 그 다음 경우의 수를 고려

③ Simulation

• expansion 과정으로 받은 노드로부터 game 실행.

(게임이 끝날 때까지 진행) *backpropagation* 과정을 통해 움직인다.

④ *backpropagation* : 트리의 root로 다시 올라가서 이길 확률을 증가시킨다.

MCTS를 반복해서 승리 경로의 확률은 ↑

마지막 수정: 2020년 1월 14일