##0308

2021년 3월 1일 월요일 오후 4:52

■ 6장. 학습 관련 기술들

□ 6.3 배치 정규화

● 6.3.1 매치 정규화 알고리즘

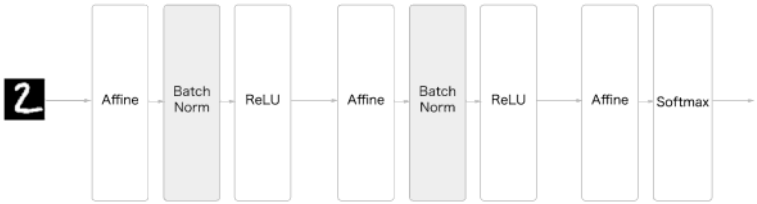
배치 정규화의 장점

- 학습속도 개선
- 초기값 설정에 크게 의존하지 않음
- 오버피팅 억제(드롭아웃 필요성 감소)

배치 정규화의 목적

각 층의 활성화값들이 강제로 펴지면서 학습을 원할하게 하기 위함

배치 정규화를 사용한 신경망의 예시



활성함수(ReLU)에 결과값 적용 전에 Batch_Norm 으로 필터링 : range[0,1]

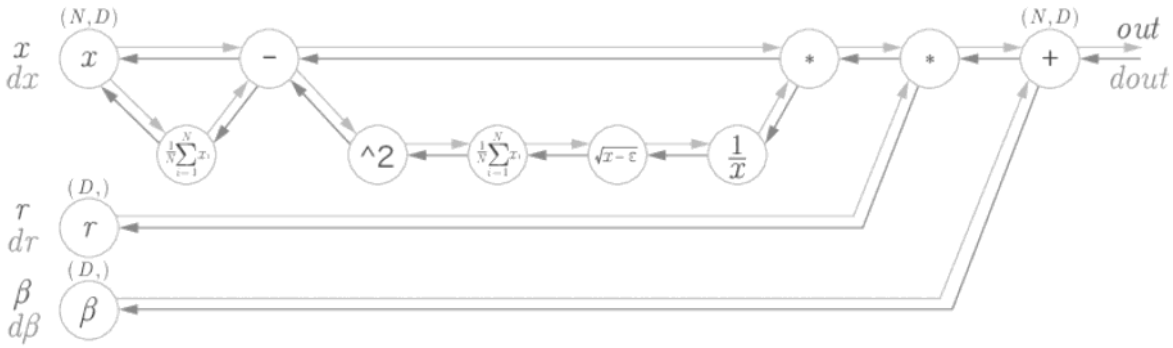
단위는 미니배치(mini Batch) 이다

책 p.211 하단부분에 의하면 Batch_Norm 은 활성함수의 앞 뒤 모두 사용될 수 있는 것으로 추측

[식6-7]

$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$	Mean
$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i-\mu_B)^2$	Variance
$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2+\varepsilon}}$	N ~ X_hat = ((X-M)/std)

[그림 6-17] #배치 정규화의 계산 그래프



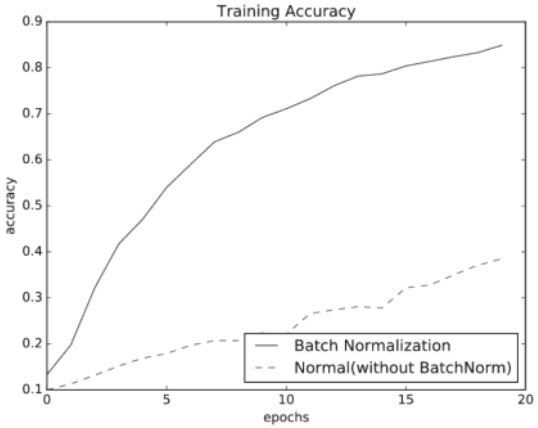
r(gamma), B(beta) 가 각각 확대(*), 이동(+)을 담당한다

최초 r=1, B=0 으로 시작하며 학습하면서 적절한 값으로 조정된다

gamma = scale(확대) : 기울기 개념

batta = shift(이동) : 절편 개념

[그림 6-18] #배치 정규화 효과

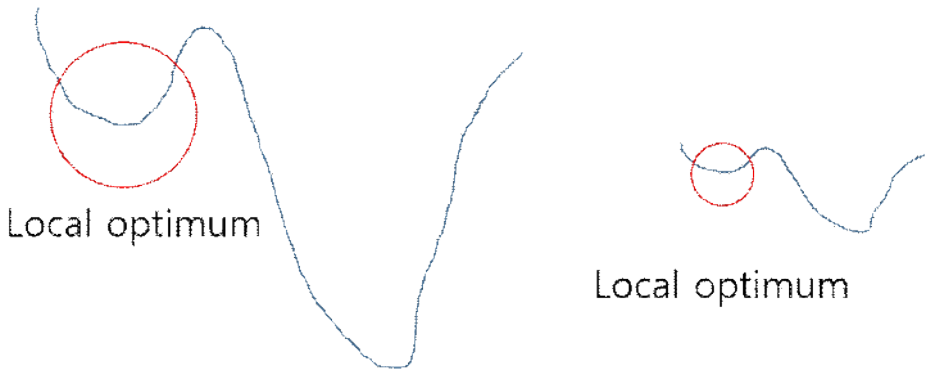


에폭당 정확도 상승폭이 Batch Normalization을 사용했을 때 훨씬 크다

※ 추가학습 : https://eehoeskrap.tistory.com/430

배치정규화

학습 자체를 전체적으로 안정화하며 ML 진행시 발생하는 Gradient Vanishing / Gradient Exploding 문제를 해결하는 것



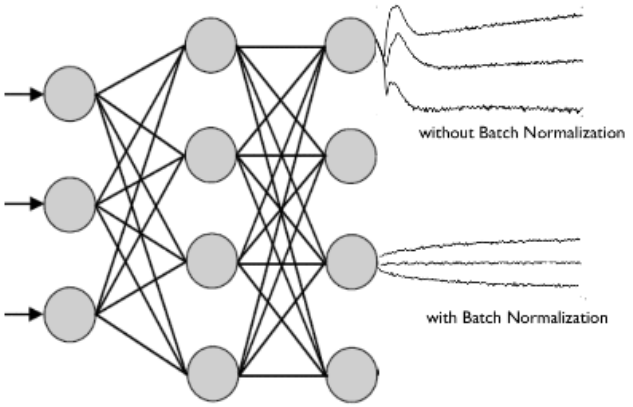
#(좌) Normalization 적용 전 / (우) Normalization 적용 후

Internal Covariance Shift (내부 공변량 이동)

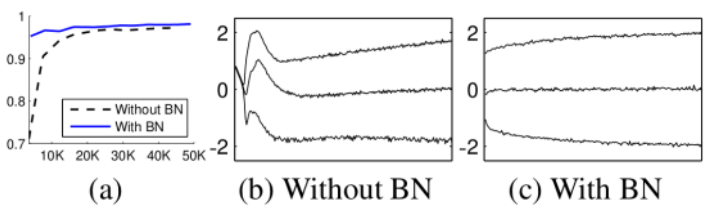
배치 정규화 논문에서는 학습에서 불안정화가 일어나는 이유를 'Internal Covariance Shift' 라고 주장하고 있는데,
이는 네트워크의 각 레이어나 Activation 마다 입력값의 분산이 달라지는 현상을 뜻한다.

Covariate Shift : 이전 레이어의 파라미터 변화로 인하여 현재 레이어의 입력의 분포가 바뀌는 현상

Internal Covariate Shift : 레이어를 통과할 때 마다 Covariate Shift 가 일어나면서 입력의 분포가 약간씩 변하는 현상



예시



BN : Batch Normalization

MNIST 에서의 배치 정규화 실험

 # 수식 #책과 동일한 수식

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

#배치 정규화의 입력 및 출력 값

#배치 정규화는 미니배치의 평균과 분산을 이용해서 정규화 한 뒤에, scale 및 shift 를 감마(γ) 값, 베타(β) 값을 통해 실행한다.

#이 때 감마와 베타 값은 학습 가능한 변수이다. 즉, Backpropagation(역전파)을 통해서 학습이 된다.

배치정규화의 장점

- 배치 정규화는 단순하게 평균과 분산을 구하는 것이 아니라 **감마(Scale), 베타(Shift) 를 통한 변환을 통해 비선형 성질을 유지 하면서 학습 될 수 있게 해줌**
- 배치 정규화가 신경망 레이어의 중간 중간에 위치하게 되어 **학습을 통해 감마, 베타를 구할 수 있음**
- **Internal Covariate Shift 문제**로 인해 신경망이 깊어질 경우 학습이 어려웠던 문제점을 **해결**
- gradient 의 스케일이나 초기 값에 대한 dependency 가 줄어들어 Large Learning Rate 를 설정할 수 있기 때문에 결과적으로 **빠른 학습 가능함**, 즉, **기존 방법에서 learning rate 를 높게 잡을 경우 gradient 가 vanish/explode 하거나 local minima 에 빠지는 경향이 있었는데 이는 scale 때문이었으며, 배치 정규화를 사용할 경우 propagation 시 파라미터의 scale 에 영향을 받지 않게 되기 때문에 learning rate 를 높게 설정할 수 있는 것**임
- regularization 효과가 있기 때문에 **dropout 등의 기법을 사용하지 않아도 됨** (효과가 같기 때문)
- 학습 시 Deterministic 하지 않은 결과 생성
- Learning Rate Decay 를 더 느리게 설정 가능
- 입력의 범위가 고정되기 때문에 saturating 한 함수를 활성화 함수로 써도 saturation 문제가 일어나지 않음, 여기서 saturation 문제란 가중치의 업데이트가 없어지는 현상임

□ 6.4 바른 학습을 위해

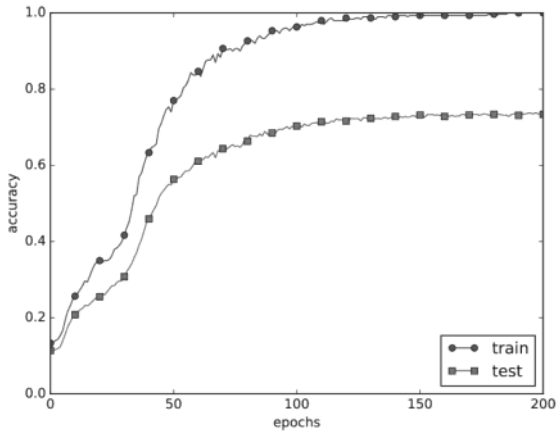
●6.4.1 오버피팅

오버피팅을 유발하는 상황

- 매개변수가 많고 복잡한 모델
- 훈련 데이터가 적을 때

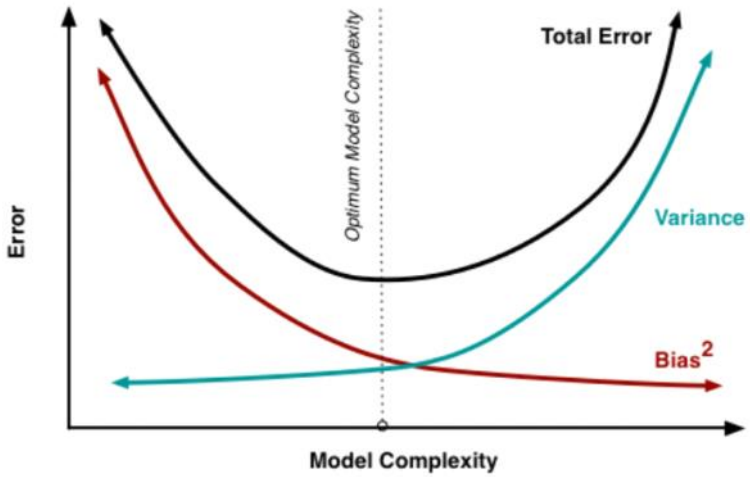
책에서는 6만개의 훈련 데이터 중 300개만 사용하고 hiden_layer 를 6개를 생성하여 7층 신경망을 구성하였음

[그림 6-20]



trainset 과 testset 간의 variance 가 epochs 가 증가할때마다 커지는 것을 관찰할 수 있다. (overfitting)
epochs 가 적은 부분인 50 epochs 미만에서는 variance 가 작지만 accuracy(정확도)가 낮은 underfitting이 관측된다.

1) 모델 복잡도에 따른 오버피팅여부확인

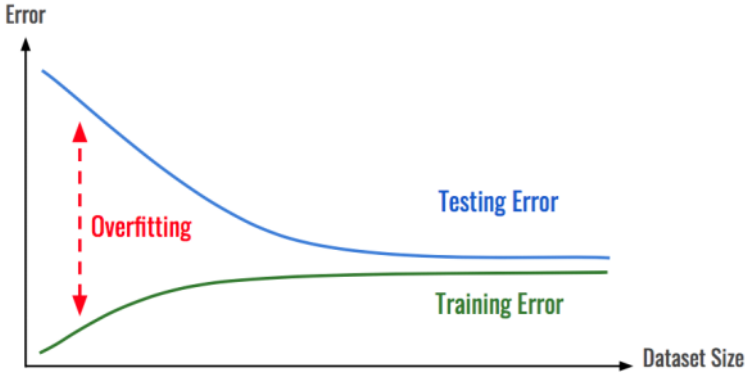


그래프의 좌측이 underfitting, 우측이 overfitting
실선 부분이 최적화 지점(total error 의 최소값)
경사하강법으로 total error 을 찾는 이유중에 하나

아래 그래프들의 출처 및 details url :
https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42

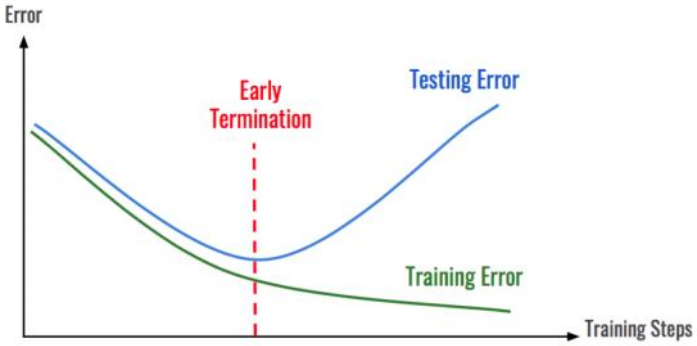
1-2) 모델복잡도에 따른 overfitting

2) Dataset Size - 훈련데이터가 적을 때



Dataset Size 가 클수록 overfitting 으로부터 멀어지며 일반화가 더 수월하다
Dataset 이 작으면 training 에 과적합될 우려가 크다
ex) 20~40대 초의 평균희망 직종을 조사하는데 trainset 으로 우리반 학생들만을 적용하고 testset 으로 서울시민 무작위 15명인 경우

3) Early stopping - Regularization 방법 중 하나



GridCV 옵션을 과하게 주면 빨간 점선을 넘어갈 확률이 높아진다.
어떤 모델들은 Early stopping 을 지정할 수 있다.

●6.4.2 가중치 감소

"오버피팅 억제를 위해 전통적으로 사용되는 방법 중 하나로 가중치 감소(weight decay)가 존재한다."
- 가중치 매개변수의 값이 큰 경우 오버피팅이 발생하는 경우가 많음
- 큰 가중치에 대해서 큰 페널티를 부과

L1 Regularization & L2 Regularization

https://light-tree.tistory.com/125

[Index]

1. Norm	2. L1 Norm	3. L2 Norm	4. L1 Norm 과 L2 Norm 의 차이
5. L1 Loss	6. L2 Loss	7. L1 Loss, L2 Loss 의 차이	8. Regularization
9. L1 Regularization	10. L2 Regularization	11. L1 Regularization, L2 Regularization 의 차이와 선택 기준	

위 목차 중 1~4 만 간단하게 정리

1. Norm

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}.$$

p는 Norm 의 차수

p	Norm 종류
1	L1 Norm
2	L2 Norm

2. L1 Norm

$$d_1(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_1 = \sum_{i=1}^{n} |p_i - q_i|, \text{ where } (\mathbf{p}, \mathbf{q}) \text{ are vectors } \mathbf{p} = (p_1, p_2, \ldots, p_n) \text{ and } \mathbf{q} = (q_1, q_2, \ldots, q_n)$$

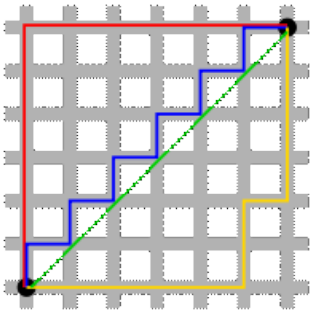
간단하게 벡터 p, q 의 각 원소들의 차 의 절대값의 합

3. L2 Norm

$$\|\boldsymbol{x}\|_2 := \sqrt{x_1^2 + \cdots + x_n^2}.$$

sqrt(([p-q]) 의 제곱의 합)
피타고라스의 정리와 유사

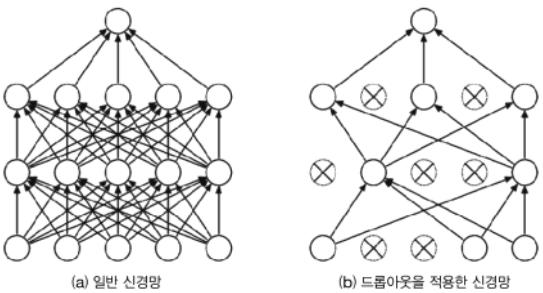
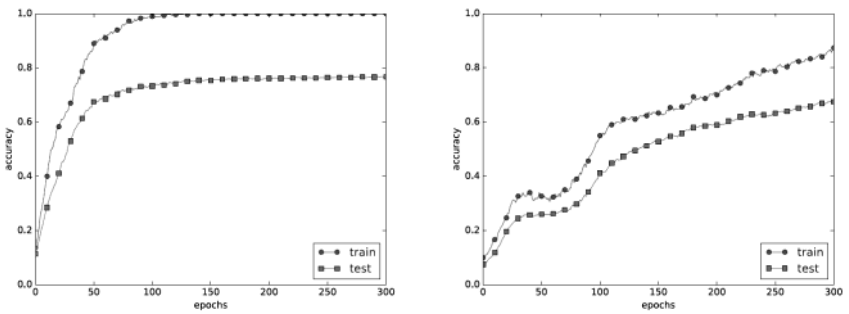
4. 둘의 차이



- 녹색 선이 L2 Norm
- L1 Norm 은 모든 선으로 표현 가능
- L2 Norm 은 가장 빠른 하나의 값만 가짐
  (두 점 사이의 거리가 최소인 값만을 가짐)

그 외 L1 Regularization, L2 Regularization 은 XGBoost 및 앙상블 모형들의 파라미터이므로 알아두면 좋습니다.

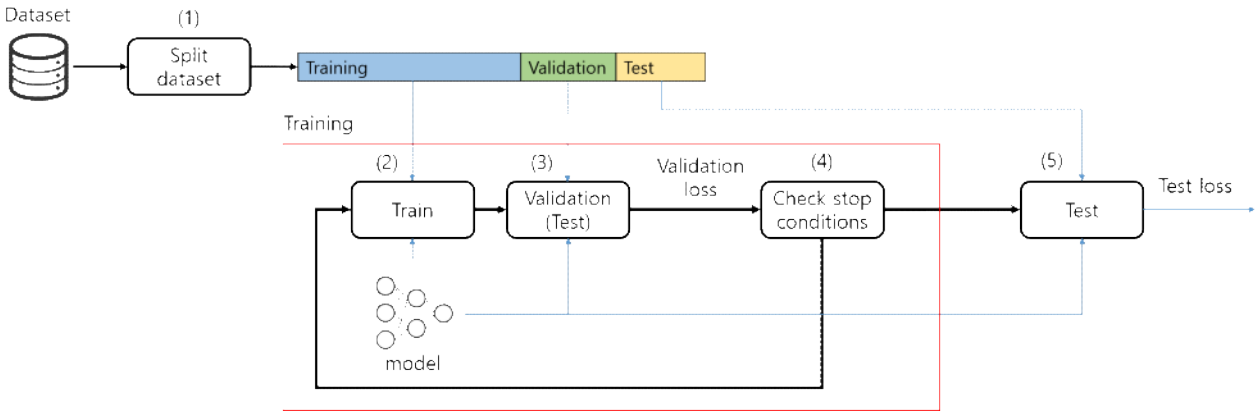
●6.4.3 드롭아웃

[그림 6-22] #원리	[그림 6-23] #결과
 (a) 일반 신경망 (b) 드롭아웃을 적용한 신경망	
	오버피팅(variance)이 감소한 것을 볼 수 있다. 좌(드롭아웃X), 우(드롭아웃 적용)

드롭아웃은 ML(기계학습)에서 앙상블 학습이 여러 모델의 평균을 내는 것과 같은 효과를 낸다
Regularization 방법 : L1 Regularization, L2 Regularization, **Dropout**, Early stopping
Early stopping 은 ML에서도 자주 사용하는 방식으로, 일정 range 를 지나도록 정해진 기준(Cost)이 개선되지 않으면 학습을 종료

□ 6.5 적절한 하이퍼파라미터 값 찾기

● 6.5.1 검증 데이터



Kaggle ---> (1)

Training+Validatoin ---> Train_data(in kaggle) ---> (2)+(3)

Test ---> Test_data(in kaggle) ---> (5)

책의 p.222 의 4~9 줄을 유심히 읽어보면 kaggle 에 도전할 때 유용한 팁을 얻을 수 있다

우리는 (5)에 해당하는 Test_data의 label 값을 알지 못하므로 (2), (3) 으로 데이터를 나누어서 모델성능평가를 해야 한다.

하지만 데이터가 너무 작고, kaggle 에서 주어진 Train_data 를 떼어내는 방식에 따라 다른 결과값이 나오는 경우가 종종 있다.

이 방법을 해결하기 위해 gridCV, k-fold, random sampling tuning 등이 있다.

Train_data 를 gridCV 에 통으로 넣고 돌릴 경우 책에서 지적한 시험 데이터에 overfitting 되는 결과를 보게 될 수 있다.

(제가 그 광경을 며칠째 보고있습니다 ㅠㅜ)

이 때, gridCV 자체적으로 n-fold 를 나누어 테스트하기 때문에 괜찮을 수 있다고 생각할 수 있지만 kaggle 에 제출할 수 있는 횟수가 정해져 있기 때문에 파생변수를 추가하거나, hyper parameter 등을 조절할 때의 testing 의 회수가 제한되는 문제점이 있다.

이를 극복하기 위해 Train_data 에서 일정 퍼센트(10~15%)를 떼어내서 semi-testset 으로 두고 나머지를 gridCV 튜닝하는 방법을 고려하고 있다. 그런데 일정 퍼센트를 random_state 에 상관없이 분리하는 방법을 찾지 못해 현재 찾는 중이다. (방법공유좀 ㅜ)

●6.5.2 하이퍼 파라미터 최적화

신경망 하이퍼파미터 최적화 시 grid search(격자탐색)과 같은 규칙적 탐색보다 무작위 샘플링 탐색이 더 좋다고 알려져 있다.

하이퍼 파라미터 조정 단계

0단계 : h.p.(하이퍼파라미터) 값의 범위 설정

1단계 : 설정된 범위에서 h.p. 의 값을 무작위로 추출

2단계 : 1단계에서 샘플링한 h.p. 의 값을 사용하여 학습 및 검증 데이터로 정확도 평가(에폭은 작게)

3단계 : 1~2단계를 특정 횟수(100회 등) 반복하여 정확도의 결과를 보고 h.p. 의 범위를 좁힌다

위 방법은 ML 적용하는 사람의 주관적 판단이 개입될 여지가 크다.

이를 배제하고 싶다면 베이즈 최적화(Bayesian Optimization)을 적용할 수 있다.

Url : http://sanghyukchun.github.io/99/

●6.5.3 하이퍼파라미터 최적화 구현하기

#np.random.uniform()

In [213]:
```
for i in range(10):
    x = np.random.uniform(0,1)
    print(x)
# 정해진 range 내의 값을 랜덤으로 출력
```

executed in 4ms, finished 03:09:47 2021-03-08

```
0.43812605043258124
0.15930724129166207
0.40621948716463496
0.6352539782107409
0.7258040085323368
0.14091123001359607
0.48274508308055697
0.20998062698362552
0.8165927639069839
0.5821709399866771
```

p.225 [code]

In [281]:

```
import numpy as np

weight_decay = 10 ** np.random.uniform(-8, -4)
lr = 10 ** np.random.uniform(-6,-2)

print(weight_decay)
print(lr)
```

executed in 4ms, finished 03:12:14 2021-03-08

```
1.0811056068320671e-08
8.319702340945825e-06
```

weight_decay(가중치 감소 계수), lr(학습률) 을 임의추출하는 코드
위와 같은 방법으로 hyperParameter 의 범위를 좁혀서 모델 학습
이러한 방법은 ML 에서도 널리 사용되고 있다