FastCampus Pytorch

Ch5. Model Improving Methods

HARRY KIM

Lecture Content

1. Data Preprocessing
2. Other Optimizers
3. Weight Initialization
4. Regularization
5. Batch Normalization / Dropout

강의 자료

- Books
  - Pattern Classification Second Edition [Duda, 2001]
  - Pattern Recognition And Machine Learning [Bishop, 2006]
  - 밑바닥부터 시작하는 딥러닝 [사이토 고키, 2017]
  - 핸즈온 머신러닝 [오렐리앙 제롱, 2018]
  - 머신러닝, 딥러닝 실전개발 입문 [쿠지라 히코우즈쿠에, 2017]

- Online
  - UVA DEEP LEARNING COURSE [University of Amsterdam, 2018]
  - CS231n [http://cs231n.stanford.edu/, 2018]
  - Machine Learning [https://ko.coursera.org/learn/machine-learning, 2018]

1. Data Preprocessing

Data Preprocessing

- **데이터 전처리(Data Preprocessing)**
  - 데이터 전처리는 모델 학습에 매우 큰 영향을 줌

카카오(g)	시럽(g)	우유(g)
300	30	10
1000	35	15
500	10	10
200	60	30

Data Preprocessing

- **데이터 전처리(Data Preprocessing)**
  - 데이터 전처리는 모델 학습에 매우 큰 영향을 줌
    - 첫 번째 초콜렛과 가장 비슷한 초콜렛은?

카카오(g)	시럽(g)	우유(g)
300	**30**	**10**
1000	35	15
500	10	10
200	60	30

Data Preprocessing

▪ 데이터 전처리(Data Preprocessing)

- 데이터 전처리는 모델 학습에 매우 큰 영향을 줌
  - 첫 번째 초콜렛과 가장 비슷한 초콜렛은?
  - 두 번째 : $(300-1000)^2+(30-35)^2+(10-15)^2 = 490000+25+25 = 490050$
  - 세 번째 : $(300-500)^2+(30-10)^2+(10-10)^2 = 40000+400+0 = 40400$
  - 네 번째 : $(300-200)^2+(30-60)^2+(10-30)^2 = 10000+900+400 = 11300$

카카오(g)	시럽(g)	우유(g)
300	30	10
1000	35	15
500	10	10
200	60	30

Data Preprocessing

- **데이터 전처리(Data Preprocessing)**
  - 데이터 전처리는 모델 학습에 매우 큰 영향을 줌
    - 첫 번째 초콜렛과 가장 비슷한 초콜렛은?
    - 두 번째 : $(300-1000)^2+(30-35)^2+(10-15)^2 = \mathbf{490000} + 25 + 25 = \mathbf{490050}$
    - 세 번째 : $(300-500)^2+(30-10)^2+(10-10)^2 = \mathbf{40000} + 400 + 0 = \mathbf{40400}$
    - **네 번째** : $(300-200)^2+(30-60)^2+(10-30)^2 = \mathbf{10000} + 900 + 400 = \mathbf{11300}$

카카오(g)	시럽(g)	우유(g)
300	30	10
1000	35	15
500	10	10
200	60	30

Data Preprocessing

- **데이터 전처리(Data Preprocessing)**
  - 데이터 전처리는 모델 학습에 매우 큰 영향을 줌
    - 평균 : 전체의 합을 개수로 나눈 값
    - 분산 : 변수의 흩어진 정도를 나타내는 값

카카오(g)	시럽(g)	우유(g)
300	30	10
1000	35	15
500	10	10
200	60	30

평균 : 500
분산 : 126667

평균 : 33.75
분산 : 422.92

평균 : 16.25
분산 : 89.58

Data Preprocessing

▪ 데이터 전처리(Data Preprocessing)

- **Standard Scaler** : 데이터를 평균이 0이고 분산이 1인 데이터로 변환

$$X = \frac{X - \mu}{\sigma}$$

카카오(g)	시럽(g)	우유(g)
-0.56195	-0.18235	-0.66034
1.404879	0.060783	-0.13207
0	-1.15488	-0.66034
-0.84293	1.276444	1.452744

평균 : 0
분산 : 1

평균 : 0
분산 : 1

평균 : 0
분산 : 1

Data Preprocessing

- **데이터 전처리(Data Preprocessing)**
  - 데이터 전처리는 모델 학습에 매우 큰 영향을 줌
    - 첫 번째 초콜렛과 가장 비슷한 초콜렛은?
    - 두 번째 : 1.0725
    - **세 번째 : 0.2416**
    - 네 번째 : 1.3804

카카오(g)	시럽(g)	우유(g)
-0.56195	-0.18235	-0.66034
1.404879	0.060783	-0.13207
0	-1.15488	-0.66034
-0.84293	1.276444	1.452744

Data Preprocessing

▪ 데이터 전처리(Data Preprocessing)

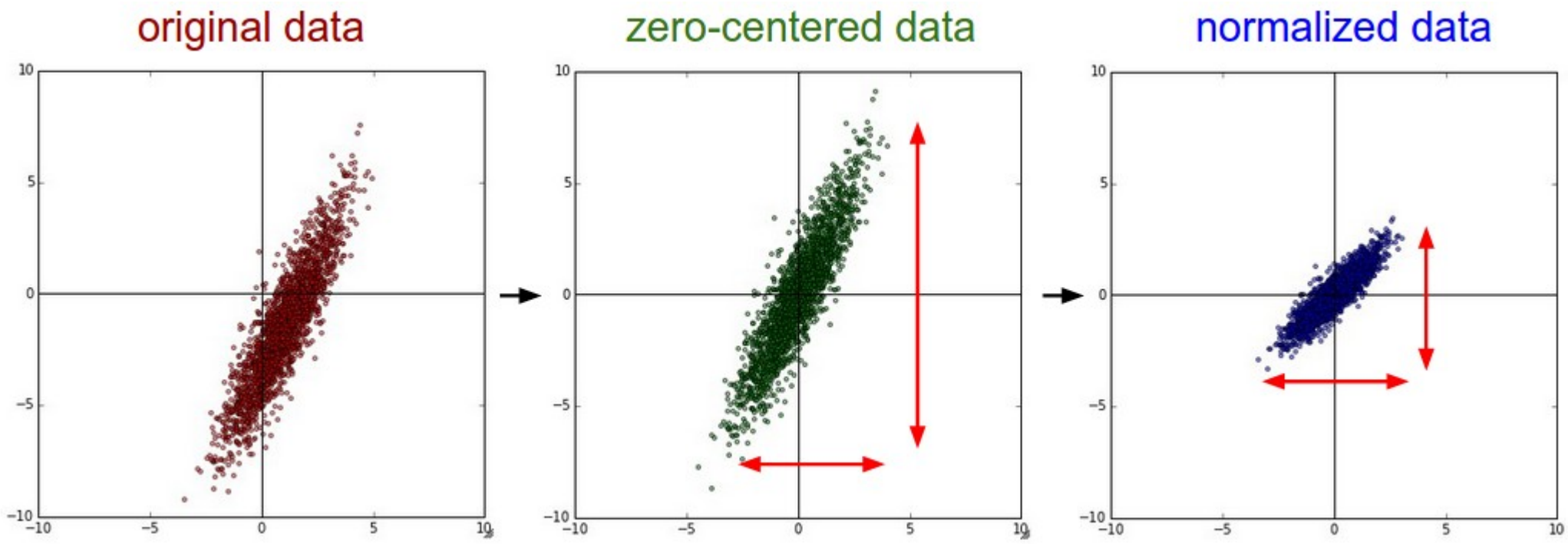
- **Minmax Scaler** : 데이터를 [0, 1] 사이의 데이터로 변환

$$X = \frac{X - X_{min}}{X_{max} - X_{min}}$$

카카오(g)	시럽(g)	우유(g)
0.125	0.4	0
1	0.5	0.25
0.375	0	0
0	1	1

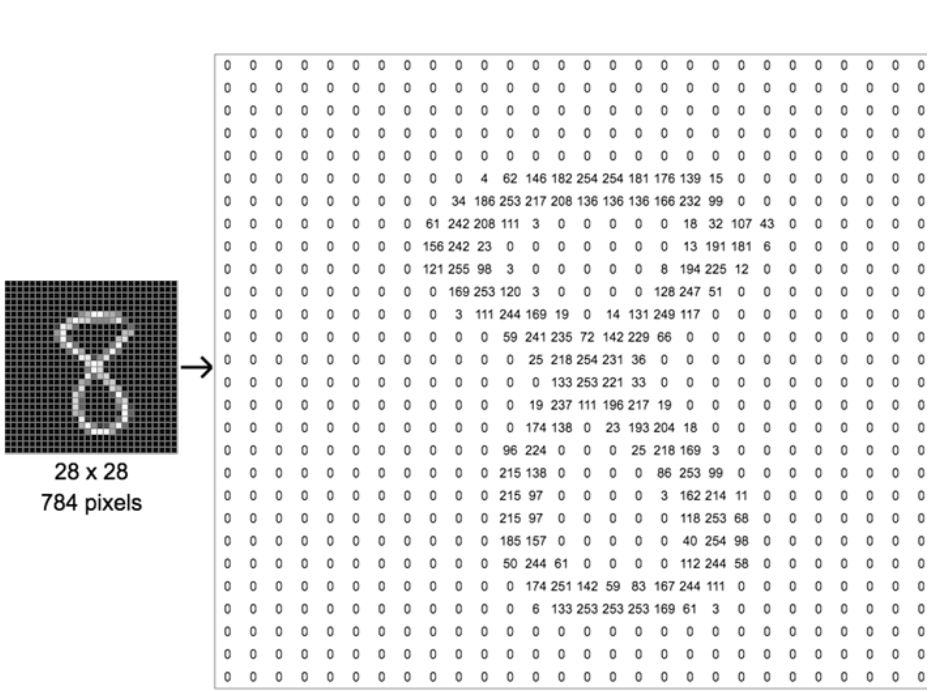
Data Preprocessing

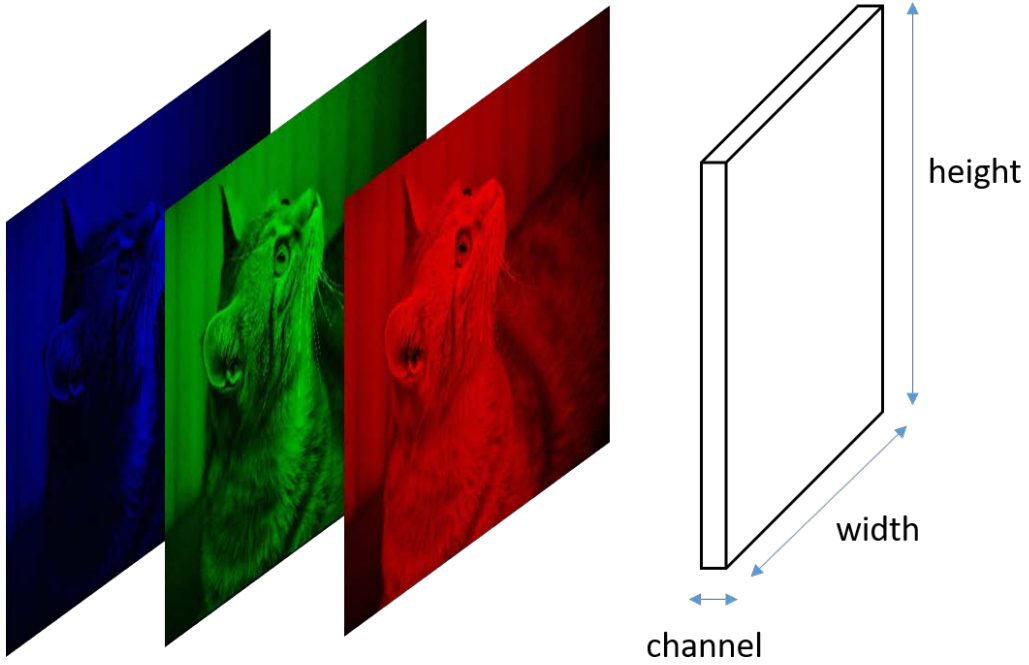
- **표준화/정규화(Normalization)의 이점**
  - 각각의 크기(Scale)를 맞춰주어, 어느 한 특성으로 인해 결과가 크게 변동하지 않게끔 함
  - = 특성끼리의 비중을 동일시



Data Preprocessing

- 데이터 전처리(Data Preprocessing)
  - 필터마다 혹은 이미지 전체에 앞과 같은 정규화 과정 사용 가능함





http://corochann.com/understanding-convolutional-layer-1227.html

Data Preprocessing

- **데이터 증폭(Data Augmentation)**
  - 정규화 이외에도 데이터 증폭과 같은 것도 데이터 전처리라고 할 수 있음
  - 사진은 앞과 같은 정규화 이 외에도 데이터 증폭 가능

원본

자르기(Crop)

회전(Rotation)

뒤집기(Flip)

2. Other Optimizers

Other Optimizers

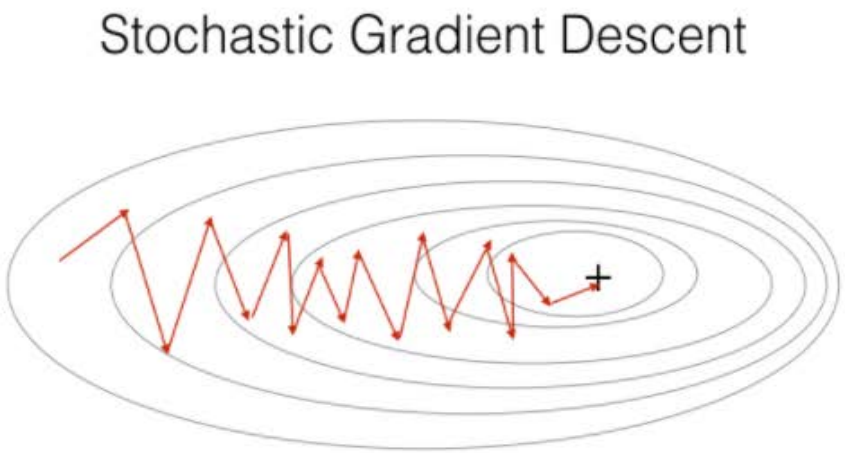
- **최적화 알고리즘(Optimizers)**
  - 여태까지는 SGD(Stochastic Gradient Descent) 방법만 활용
  - 허나 실제로는 다양한 최적화 알고리즘 존재
    - GD(Gradient Descent)
    - SGD(Stochastic Gradient Descent)
    - Momentum
    - AdaGrad
    - Adam

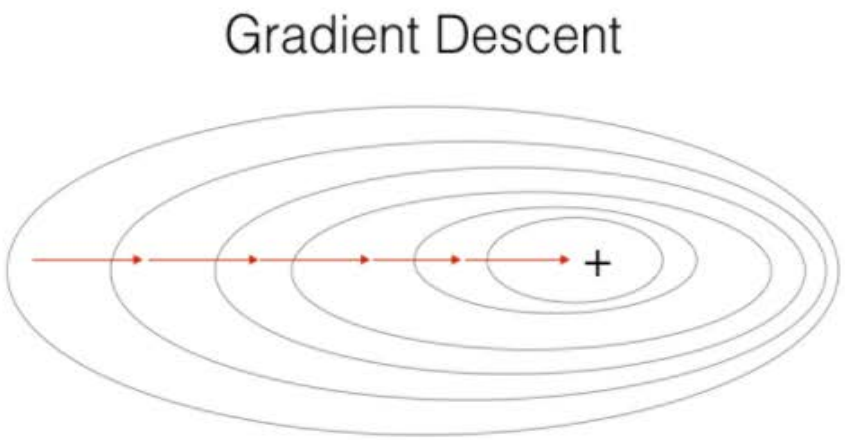
Other Optimizers

- **GD(Gradient Descent) v.s. SGD(Stochastic Gradient Descent)**
  - GD(Gradient Descent)
    - $w' := w - \eta * grad$
    - 위 알고리즘을 모든 Training Data에 대해 (Full Batch) 반복 실행
    - 주어진 데이터로 최적의 경로를 탐색
    - 하지만 계산량이 크기 때문에 메모리 과부하, 속도 저하 등의 문제 존재
  - SGD(Stochastic Gradient Descent)
    - $w' := w - \eta * grad$
    - 위 알고리즘을 각 샘플(혹은 Batch)에 대해 반복 실행
    - 최적의 경로가 아닐 수 있음
    - 하지만 계산량이 작기 때문에 빠른 학습이 가능

Other Optimizers

- GD(Gradient Descent) v.s. SGD(Stochastic Gradient Descent)

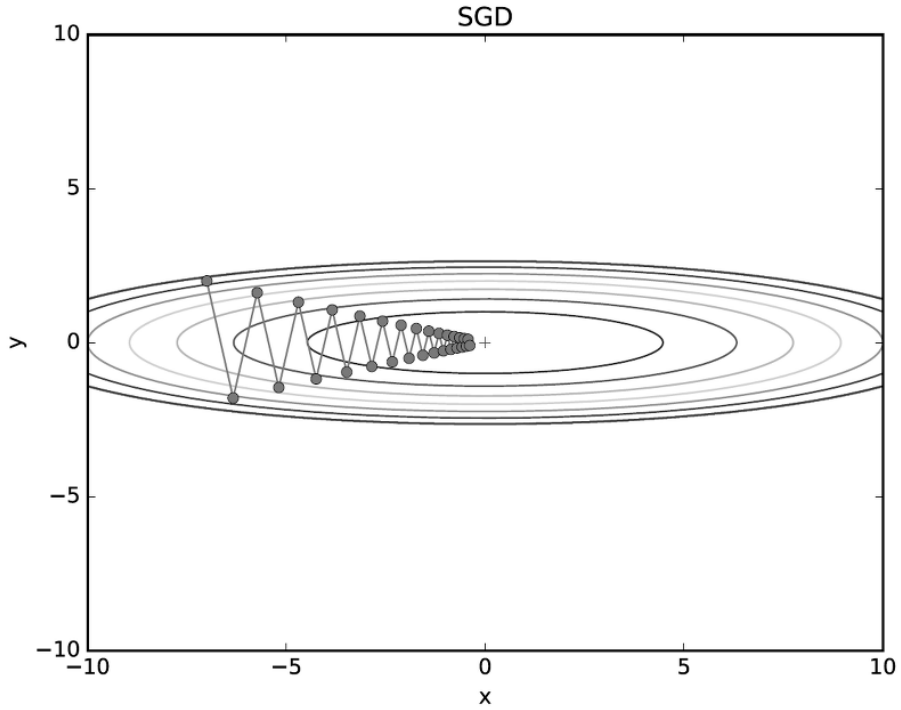




https://engmrk.com/mini-batch-gd/

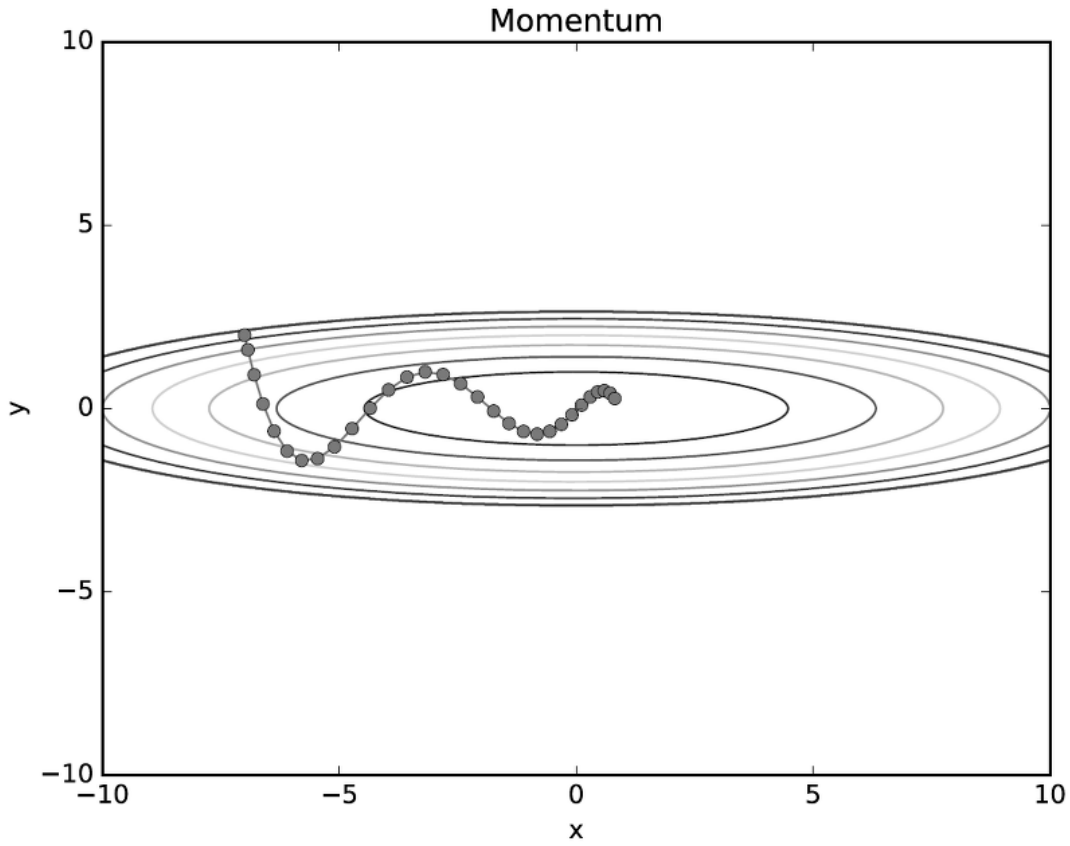
Other Optimizers

- **SGD의 단점**
  - 비등방성 함수(isotropic function, 방향에 따라 기울기가 달라지는 함수)에서 비효율적
    - 원, 구 : 등방성 함수 (방향에 따라 기울기 상관 없음)
    - 타원, 쌍곡선 : 비등방성 함수 (방향에 따라 기울기 바뀜)
  - 기울기가 최소점을 가르키지 않는다면 지그재그로 움직임
  - $w' := w - \eta * grad$



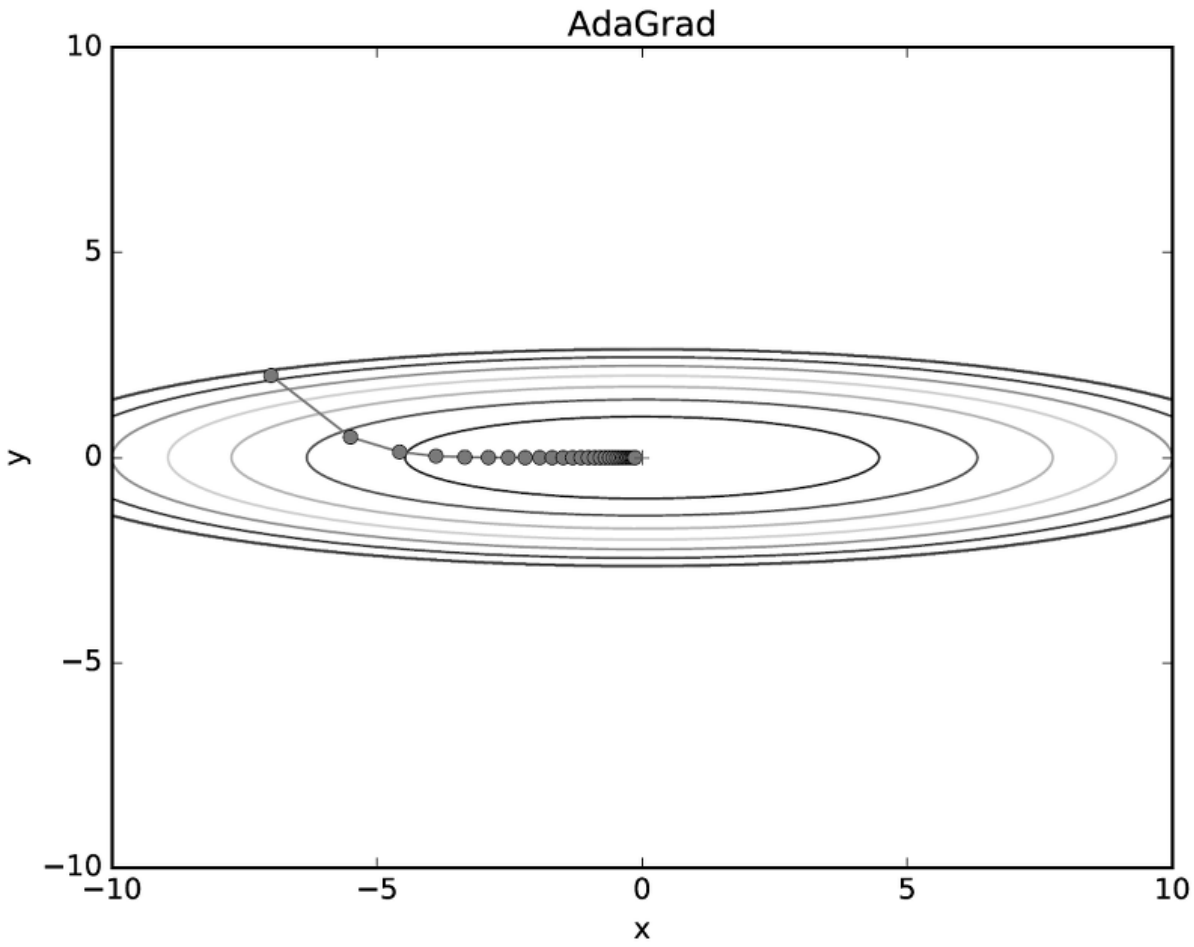
Other Optimizers

- **Momentum**
  - 이전 변화량(속도)을 가중치 업데이트에 사용
  - **관성**을 가지는 모델
  - $v' = \alpha * v - \eta * grad$
  - $w' := w + v'$
  - $\alpha = 0.9$ 등으로 설정 ($\alpha = 0$이면 SGD)



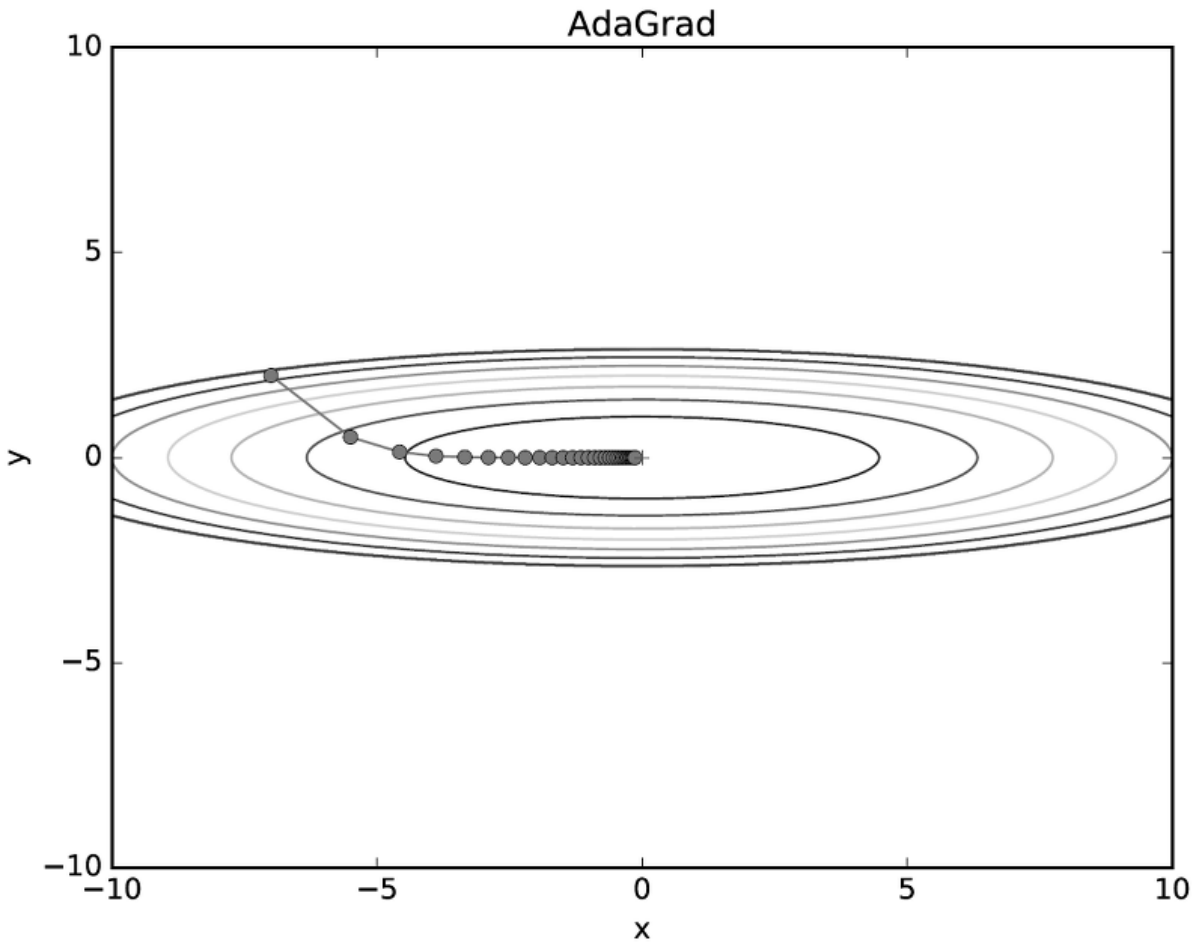
Other Optimizers

- **AdaGrad**
  - 학습률을 감소시키면서 학습 진행
  - $h' = h + grad \odot grad$
  - $w' := w - \eta \frac{1}{\sqrt{h'+\varepsilon}} grad$
  - 크게 움직일 수록 학습률 감소



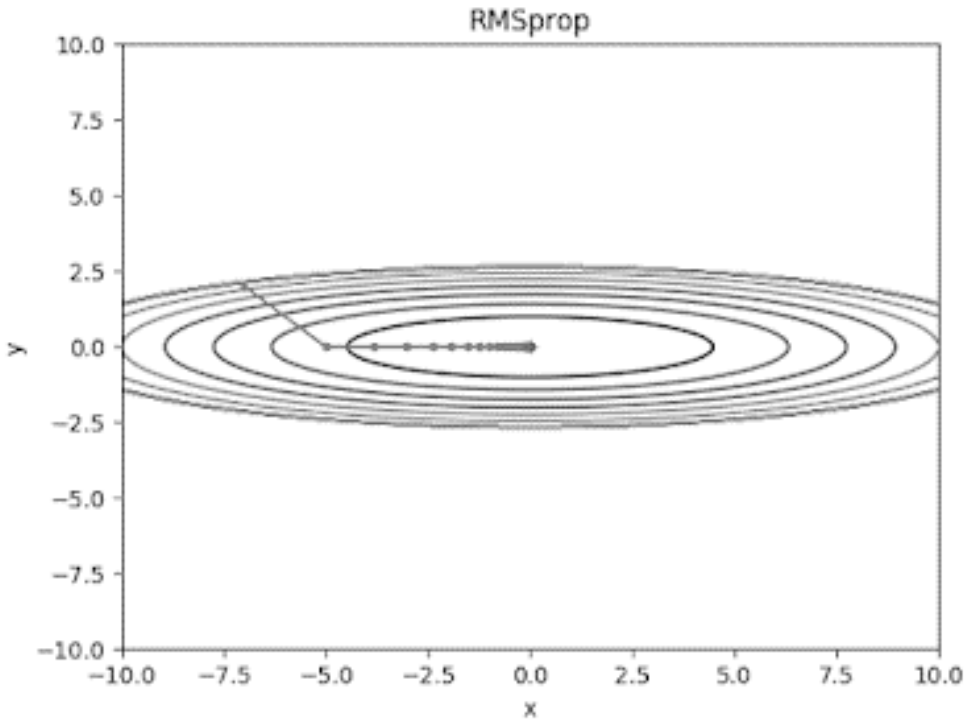
Other Optimizers

- **AdaGrad의 단점**
  - 학습률을 감소시키면서 학습 진행
  - 어느 순간이 되면 학습률=0 가능
  - 즉, 학습이 의미 없어짐



Other Optimizers

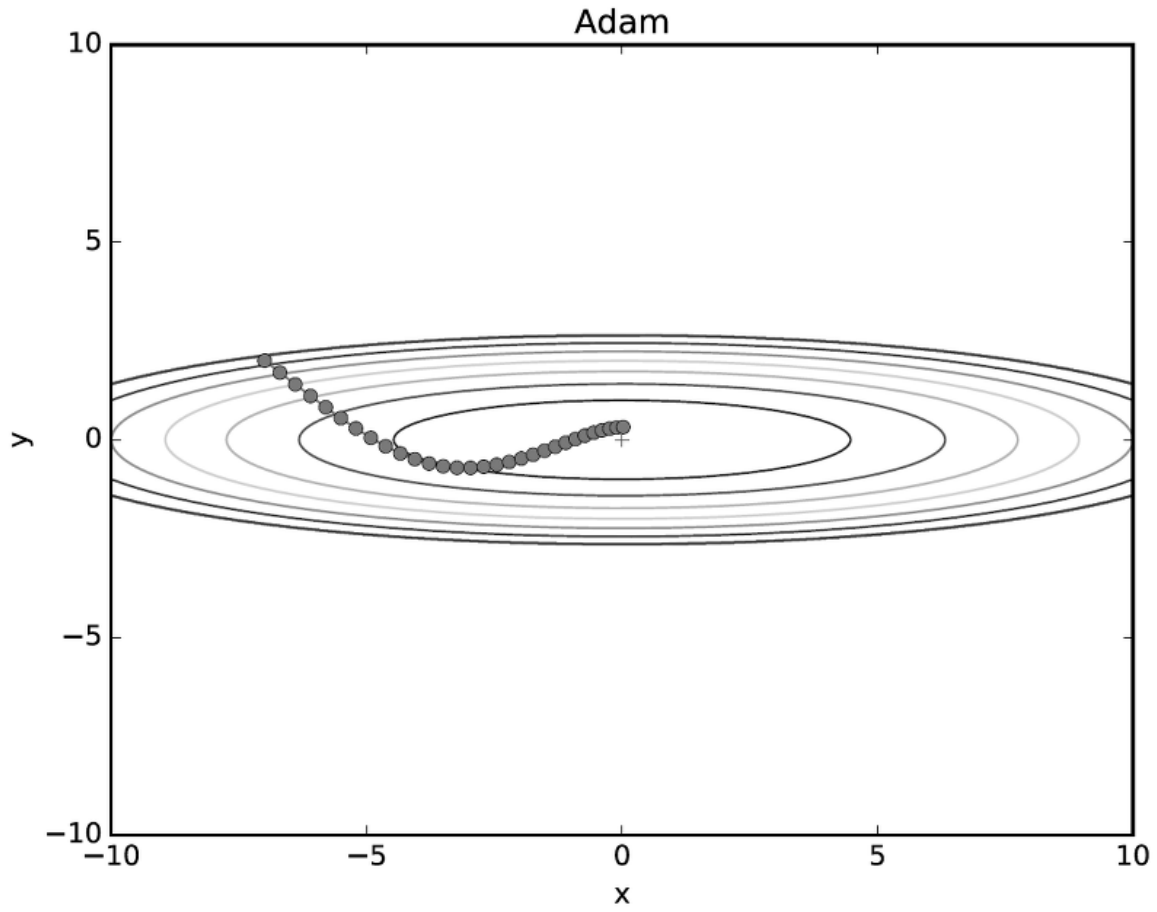
- **RMSProp**
  - 학습률을 감소시키면서 학습 진행
  - 하지만 가장 가까운 기울기에 가중하며 진행
  - 과거 기울기의 반영 규모를 기하급수적으로 감소
  - = 지수이동평균(Exponential Moving Average)
  - $h' = \gamma h + (1-\gamma) * grad \odot grad$
  - $w' := w - \eta \frac{1}{\sqrt{h\prime+\varepsilon}} grad$
  - $\gamma = 0.9$ 정도로 설정



http://bakbang.blogspot.com/2017/08/rmsprop.html

Other Optimizers

- **Adam**
  - **Ada**grad + **M**omentum
  - 두 가지 방법의 조합
  - 따라서 매개변수도 2개 지정
  - $h' = \beta_2 h + (1-\beta_2) * grad \odot grad$
  - $v' = \beta_1 v + (1-\beta_1) * grad$
  - $\hat{v}' = \frac{v'}{1-\beta_1^t}, \hat{h}' = \frac{h'}{1-\beta_2^t}$
  - $w' := w - \eta \frac{1}{\sqrt{\hat{h}' + \varepsilon}} \hat{v}'$
  - $\beta_1 = 0.9,\ \beta_2 = 0.999$

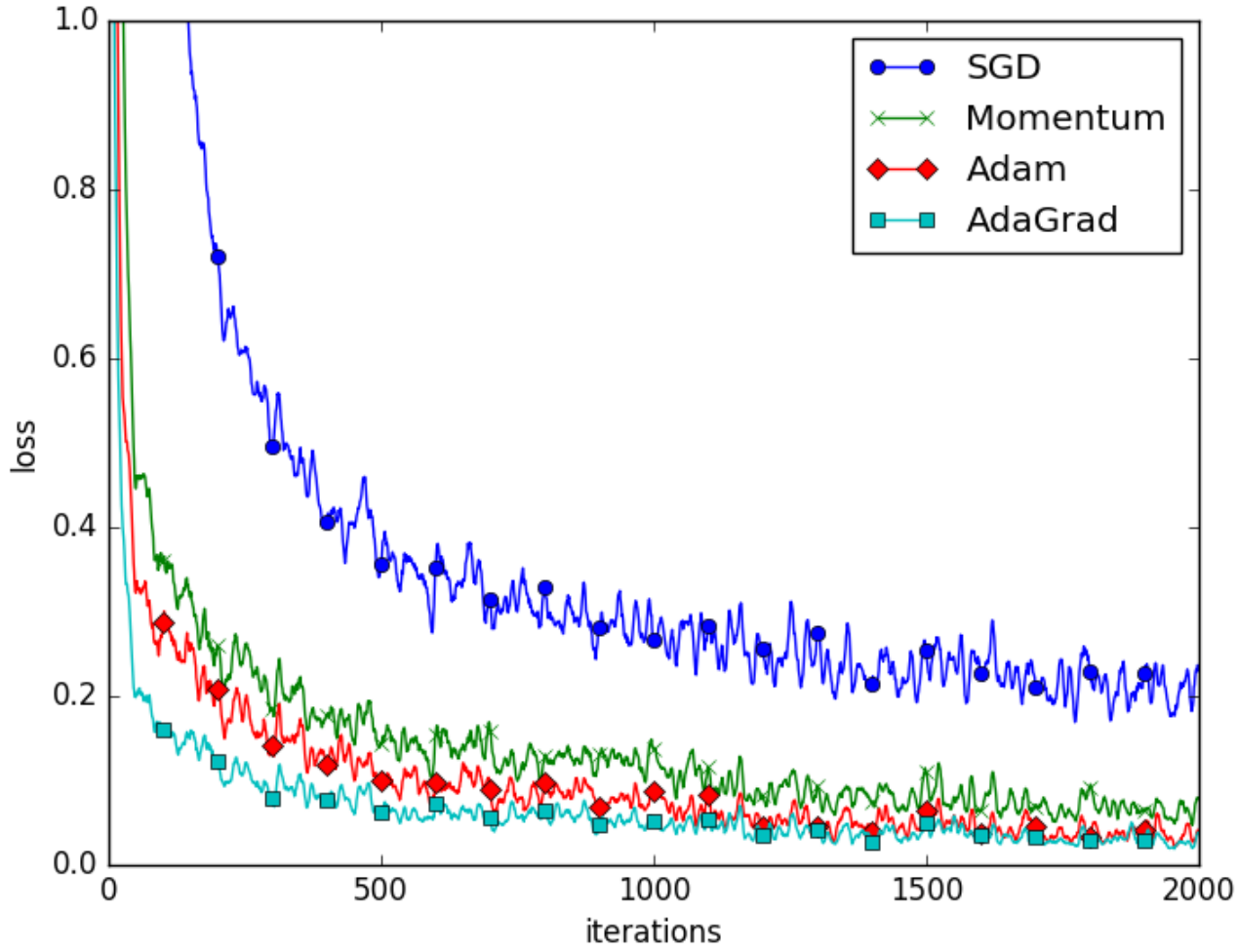


Other Optimizers

- 이 외에도
  - **NAG**
    - Momentum과 달리 SGD와 같이 업데이트 후, 그 자리에서 관성 방향으로 움직임
  - **AdaDelta**
    - AdaGrad의 학습률이 저하되는 것을 방지하기 위해 이전 몇 개의 $grad \odot grad$만 저장

Other Optimizers

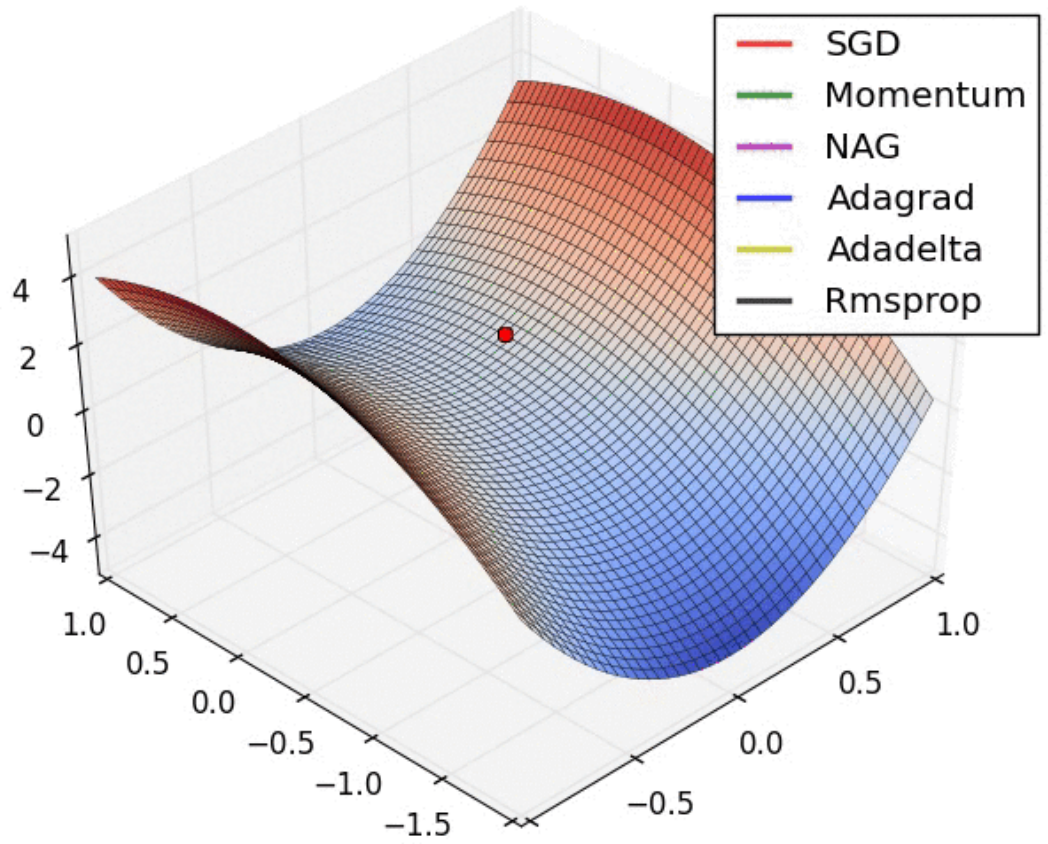
- SGD, Momentum, AdaGrad, Adam

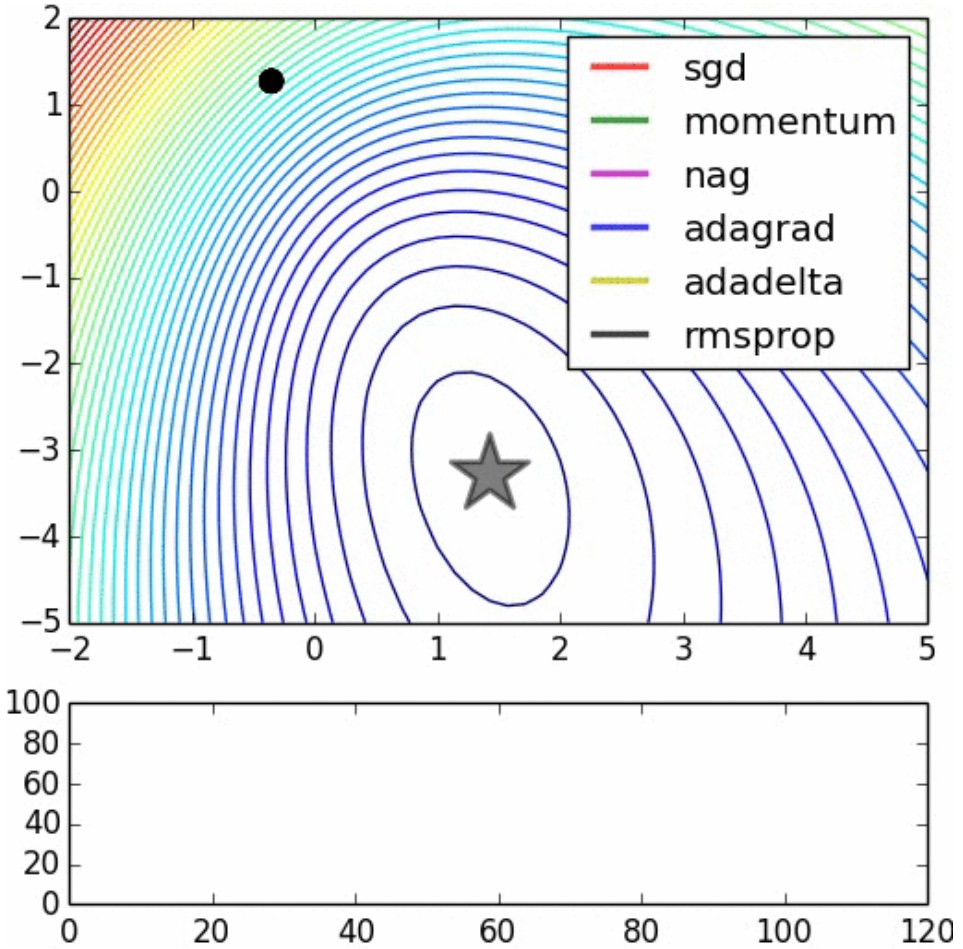


Fast campus

Other Optimizers

- SGD, Momentum, NAG, AdaGrad, Adadelta, RMSProp

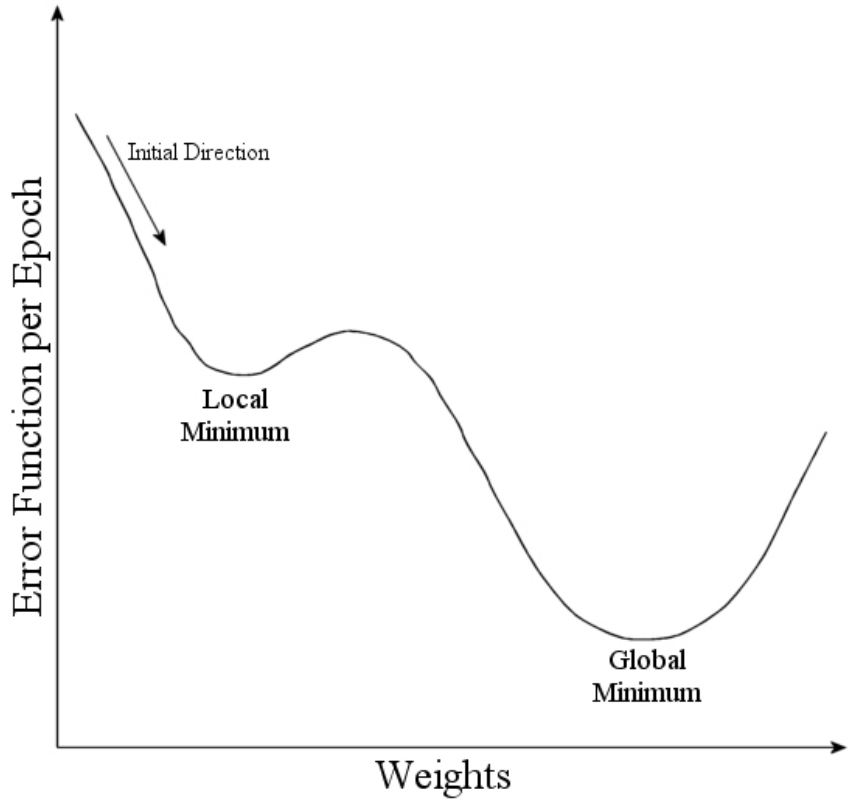




3. Weight Initialization

Weight Initialization

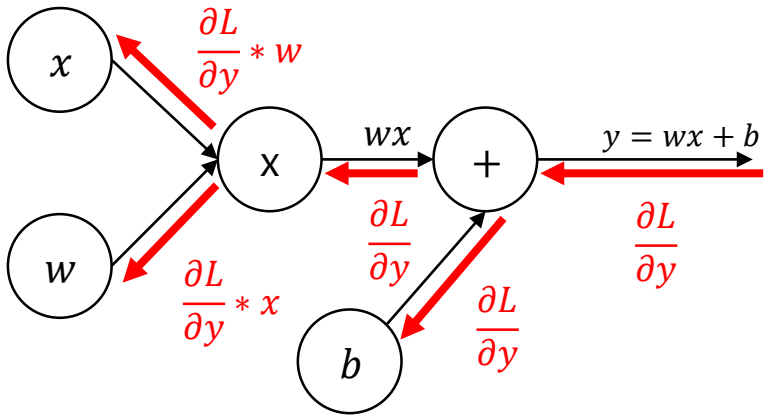
- **가중치 초기화(Weight Initialization)**
  - 가중치는 모델에서 중요한 역할 = 가중치에 따라 모델의 성능이 큰 차이
  - 따라서 초기 가중치에 따라 모델의 개선 방향이 결정됨



https://www.cse.unsw.edu.au/~cs9417ml/MLP2/BackPropagation.html

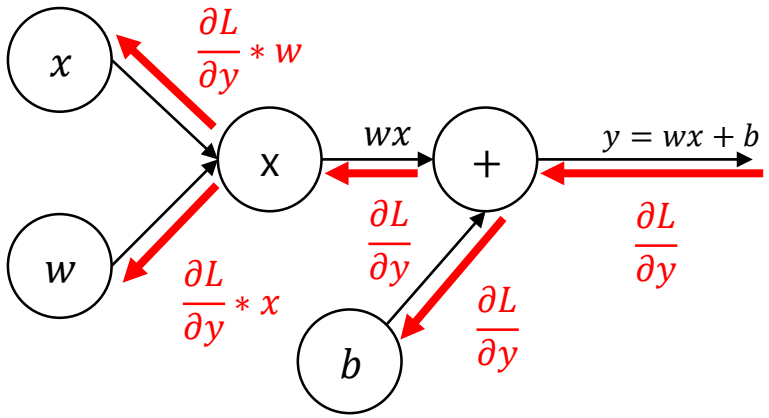
Weight Initialization

- 가중치 초기화(Weight Initialization)
  - 가장 단순한 방법 : 모든 값을 0으로 초기화
    - 가중치의 다양성 버려짐
    - 역전파로 전달 받은 값이 무의미해짐
    - 초기 가중치의 고정으로 인한 모델의 한계 존재



Weight Initialization

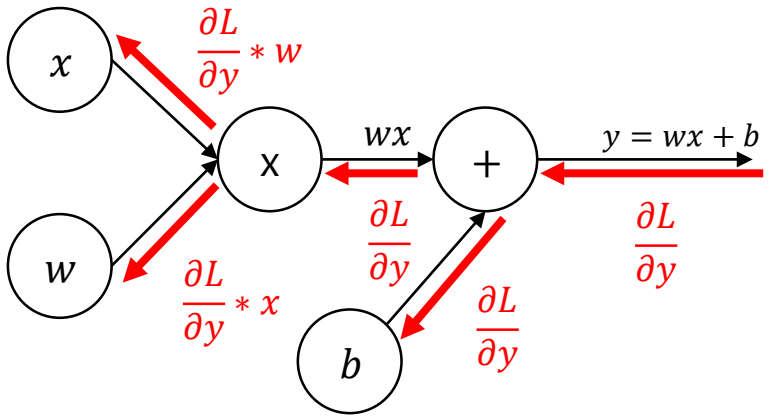
- **가중치 초기화(Weight Initialization)**
  - 가장 단순한 방법 : 모든 값을 0으로 초기화
    - 가중치의 다양성 버려짐
    - 역전파로 전달 받은 값이 무의미해짐
    - 초기 가중치의 고정으로 인한 모델의 한계 존재



  - 두 번째 단순한 방법 : 모든 값을 0-1 사이의 랜덤 값으로 초기화
    - 초기 음수 가중치의 무시

Weight Initialization

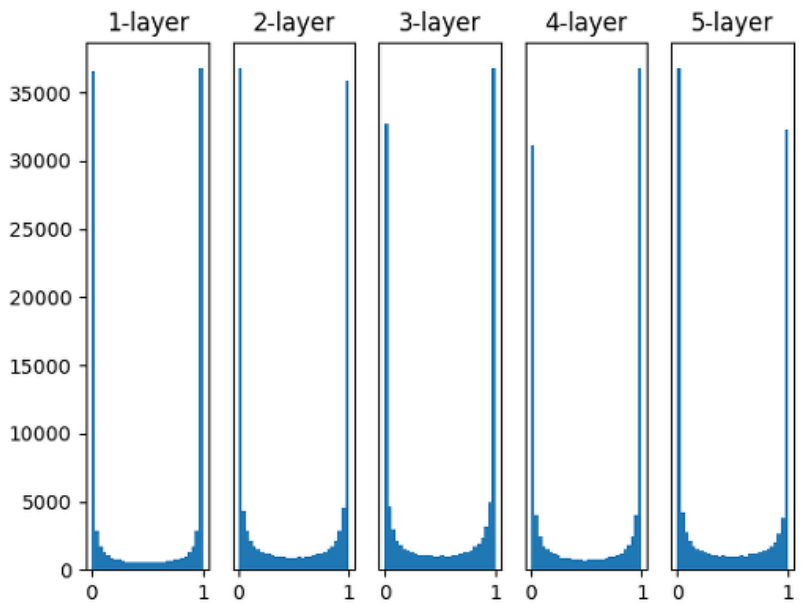
- **가중치 초기화(Weight Initialization)**
  - 가장 단순한 방법 : 모든 값을 0으로 초기화
    - 가중치의 다양성 버려짐
    - 역전파로 전달 받은 값이 무의미해짐
    - 초기 가중치의 고정으로 인한 모델의 한계 존재



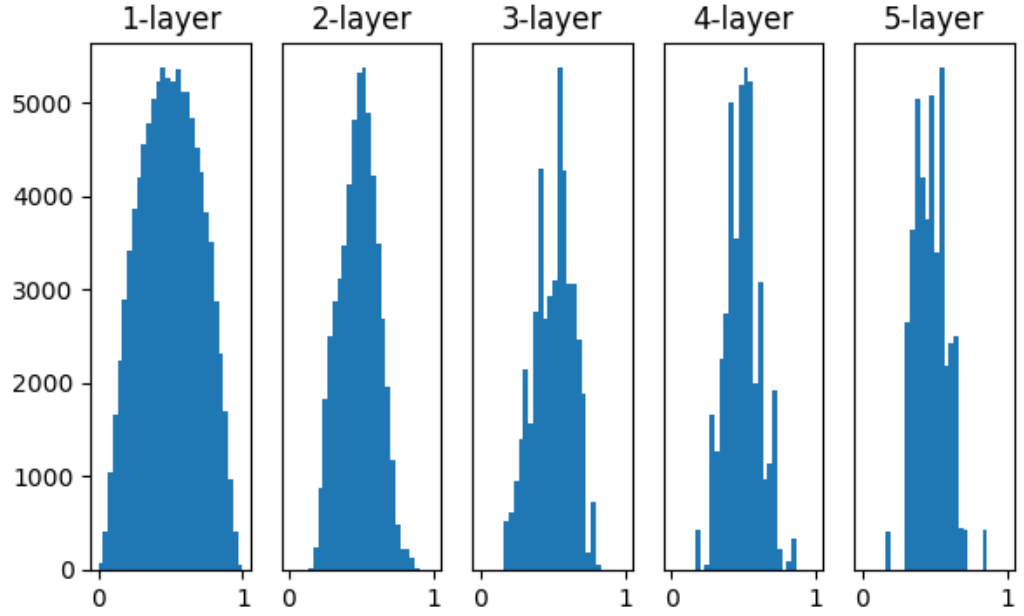
  - 두 번째 단순한 방법 : 모든 값을 0-1 사이의 랜덤 값으로 초기화
    - 초기 음수 가중치의 무시
  - 세 번째 단순한 방법 : 모든 값을 평균이 0이고, 분산이 1인 정규분포를 따르도록 추출
    - 가장 보편적인 방법
    - 하지만 항상 좋은 성능을 보장하지는 않음

Weight Initialization

- 가중치 초기화(Weight Initialization)
  - Xavier Initialization (For Sigmoid Function)
    - 입력층과 출력층의 개수를 바탕으로 초기화
    - np.random.randn(in_num, out_num) / sqrt(in_num)



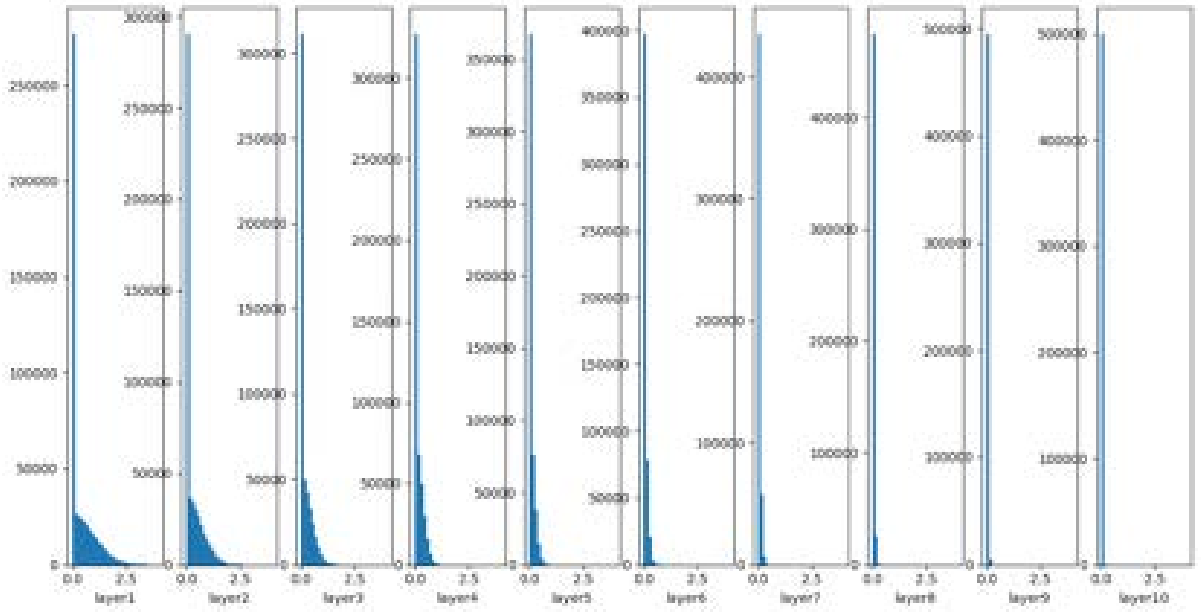
http://gomguard.tistory.com/184



http://gomguard.tistory.com/184

Weight Initialization

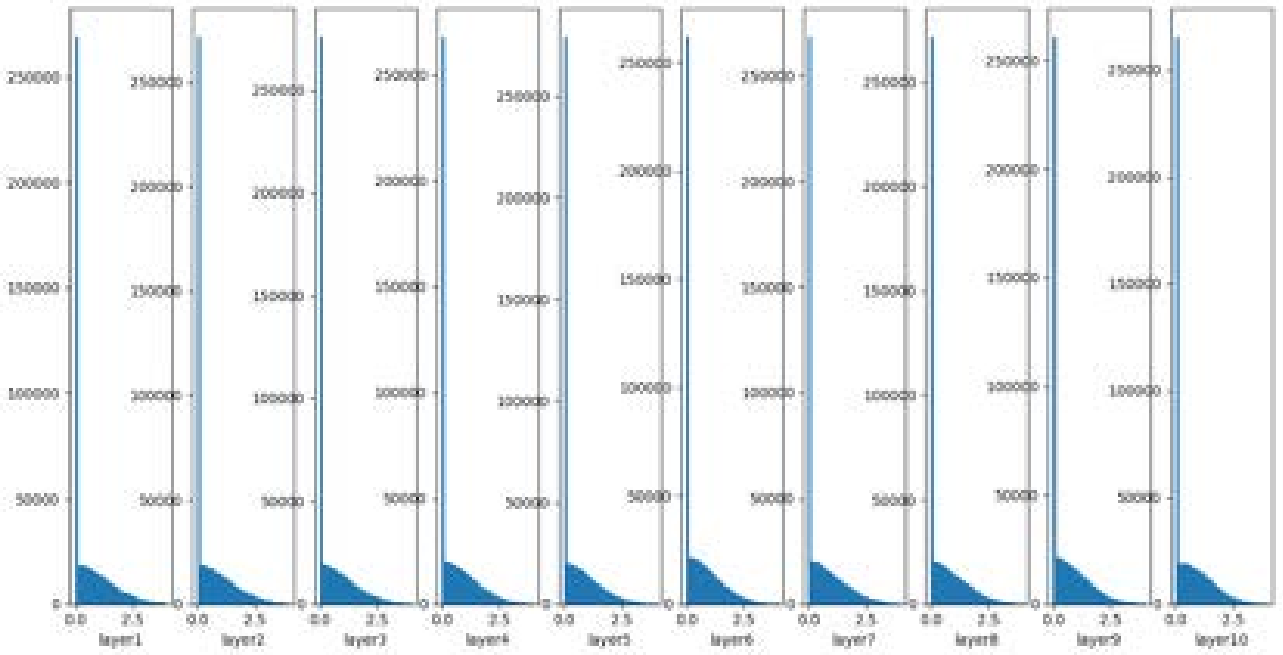
- 가중치 초기화(Weight Initialization)
  - Xavier Initialization (For Sigmoid Function)
    - 입력층과 출력층의 개수를 바탕으로 초기화
    - np.random.randn(in_num, out_num) / sqrt(in_num)
    - ReLu에 대해서는 좋지 않은 결과



http://gomguard.tistory.com/184

Weight Initialization

- 가중치 초기화(Weight Initialization)
  - Kaiming/He Initialization (For Relu Function)
    - 입력층과 출력층의 개수를 바탕으로 초기화
    - np.random.randn(in_num, out_num) / sqrt(in_num/2)



http://gomguard.tistory.com/184

Weight Initialization

- **편향 초기화(Bias Initialization)**
  - 편향(Bias)은 0으로 초기화하여도 상관 없음
    - 보통 가중치에 랜덤한 값을 부여하여 대칭성을 해결하기 때문
  - 또한 보편적으로 **0으로 초기화**

4. Regularization

Regularization

- **Regularization**
  - Parameter(Weight, Bias)에 제약을 거는 것
    - 모델을 과적합(Overfitting)되게 하지 않음
    - 보다 좋은 모델로 수렴하도록함
  - L1 regularization (Lasso)
    - 목적함수에 $\lambda|w|$를 더한다.
  - L2 regularization (Ridge)
    - 목적함수에 $\frac{1}{2}\lambda w^2$를 더한다.
  - L1 + L2 regularization (Elastic Net)
    - 목적함수에 $\lambda_1|w| + \frac{1}{2}\lambda_2 w^2$를 더한다.

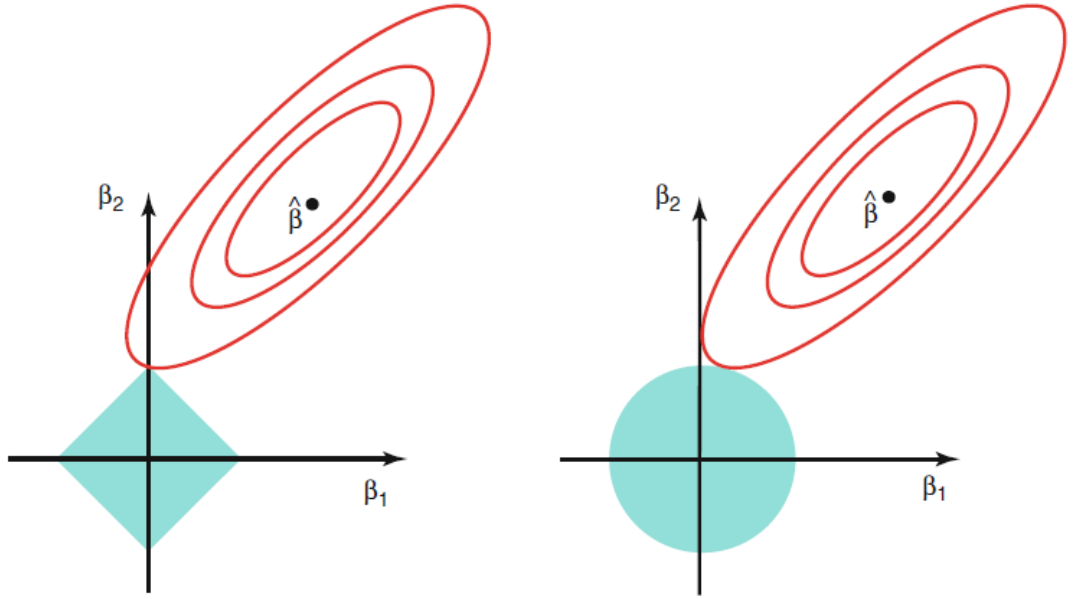


FIGURE 6.7. *Contours of the error and constraint functions for the lasso* (left) *and ridge regression* (right). *The solid blue areas are the constraint regions,* $|\beta_1| + |\beta_2| \leq s$ *and* $\beta_1^2 + \beta_2^2 \leq s$, *while the red ellipses are the contours of the RSS.*

Regularization

- **Regularization**
  - 선형회귀에서의 Regularization

$$f(x) = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n \ (x_0 = 1)$$

$$cost = MSE + Lasso = \frac{\sum_x (w^T x - y)^2}{n} + \lambda |w|$$

$$w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix} \quad x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

Regularization

- **Regularization**
  - 선형회귀에서의 Regularization
    - $|w|$를 최소화하기 위해, 각각의 가중치 w를 최소화시키게 됨
    - 즉 최대한 w를 0으로 만들게 됨 = 변수 선택

$$f(x) = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n \ (x_0 = 1)$$

$$cost = MSE + Lasso = \frac{\sum_x (w^T x - y)^2}{n} + \lambda|w|$$

Regularization

- **Regularization**
  - 선형회귀에서의 Regularization
    - $|w|$를 최소화하기 위해, 각각의 가중치 w를 최소화시키게 됨
    - 즉 최대한 w를 0으로 만들게 됨 = 변수 선택

$$f(x) = w_0x_0 + 0 * x_1 + 0 * x_2 + \cdots + w_nx_n \ (x_0 = 1)$$

$$= w_0x_0 + 0 + 0 + \cdots + w_mx_m + \cdots + w_nx_n \ (x_0 = 1)$$

5. Batch Normalization & Dropout

Batch Normalization & Dropout

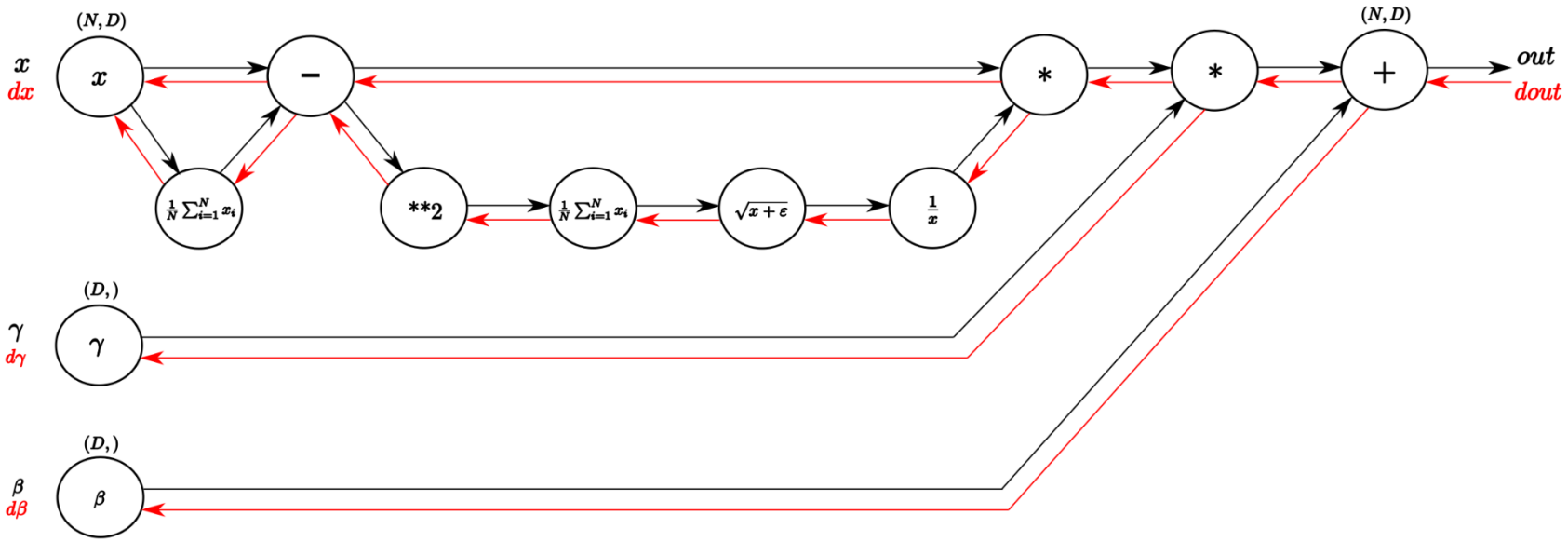
- **Batch Normalization & Dropout**
  - 앞서 소개한 방법들은 신경망이 아닌 머신러닝에서도 많이 쓰이는 방법들
  - 이제부터는 주로 신경망에서 쓰이는 기법에 해당

Batch Normalization & Dropout

Batch Normalization

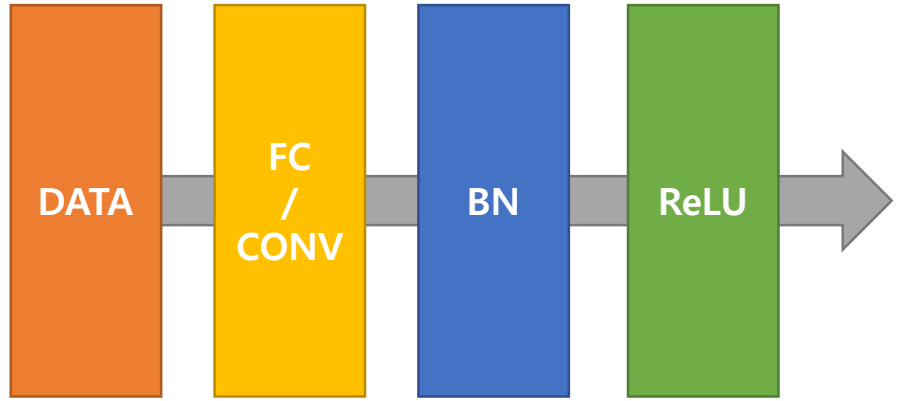
- Ioffe and Szegedy에 의해 2015년 제안된 방법, 하지만 뛰어난 성능 향상 제공
- Activation의 입력값이 표준 정규분포를 갖도록 강제

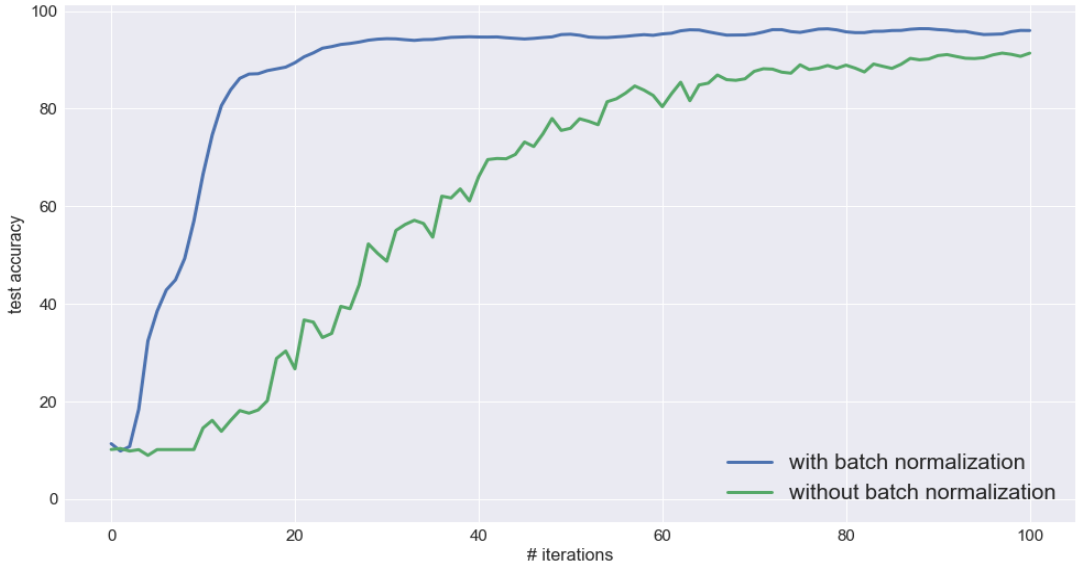
$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i \rightarrow \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2 \rightarrow x' = \frac{x-\mu}{\sqrt{\sigma^2+\varepsilon}} \rightarrow y = \gamma x' + \beta$$



Batch Normalization & Dropout

- **Batch Normalization**
  - 보통 Activation Function 전에 사용
  - NN 뿐만 아니라, CNN에서도 사용 가능
  - 사용 시 뛰어난 **성능 향상**
    - 빠른 학습의 진행
    - 초기값 변화에 강함
    - 과적합 방지

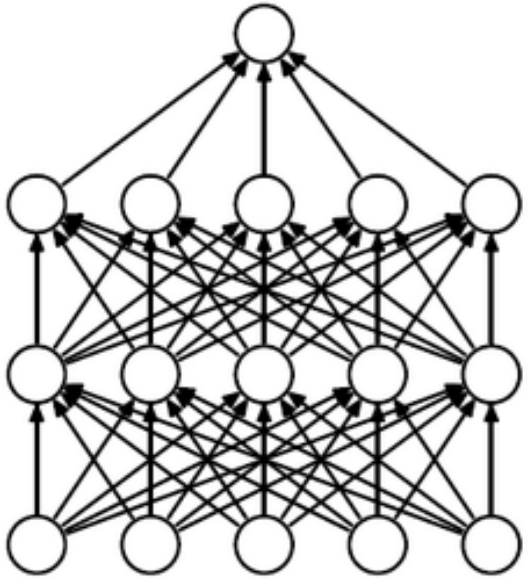




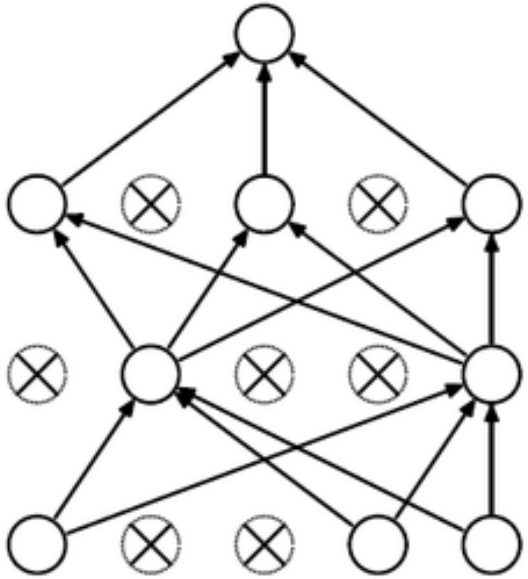
https://towardsdatascience.com/how-to-use-batch-normalization-with-tensorflow-and-tf-keras-to-train-deep-neural-networks-faster-60ba4d054b73

Batch Normalization & Dropout

- **Dropout**
  - 과적합 방지하기 위한 솔루션
  - 뉴런을 임의로 삭제하면서 학습하는 방법
  - 데이터를 학습할 때마다 무작위로 뉴런 삭제



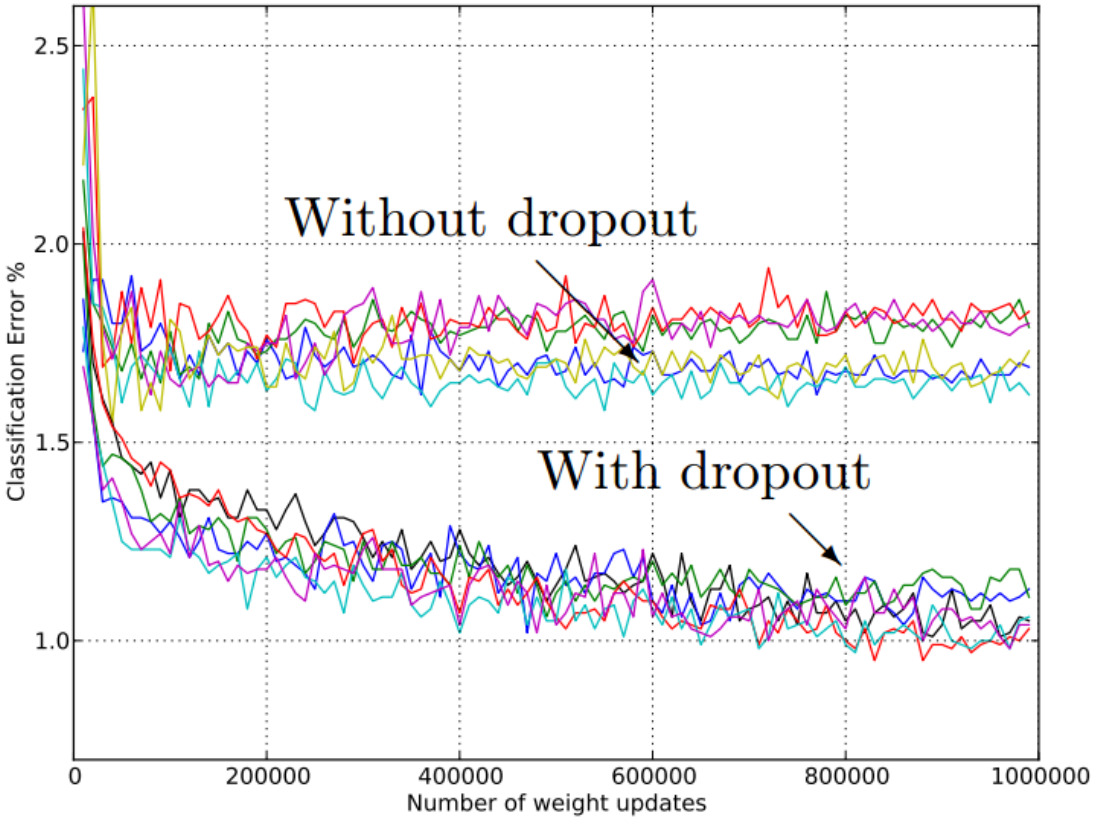
(a) Standard Neural Net



(b) After applying dropout.

Batch Normalization & Dropout

- **Dropout**
  - Test 때에는 모든 뉴런에 신호 전달
  - 단, 삭제한 비율을 곱하여 출력
    - 학습 시와 비슷한 출력를 위함
  - 따라서 과적합이 자동적으로 방지
    - 앙상블이라고 해석 가능



https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/dropout_layer.html

가장 중요한 것

Most Important Thing

- "언제나 좋은 건 없다"
  - 데이터 특징
    - 형태
    - 개수
  - 하이퍼파라미터
    - 학습률
    - 배치 수
    - 에폭
  - 랜덤 시드
  - …

Always try everything!!!

실 습