2018.10.11

NEURAL NETWORK 2

E조 : 유문상 신승진 김현수 박보정 서그림 서윤지

신경망 학습 관련 기술들

목 차

Optimization

●What is Optimization?

Minimizing Loss by the network`s training process.

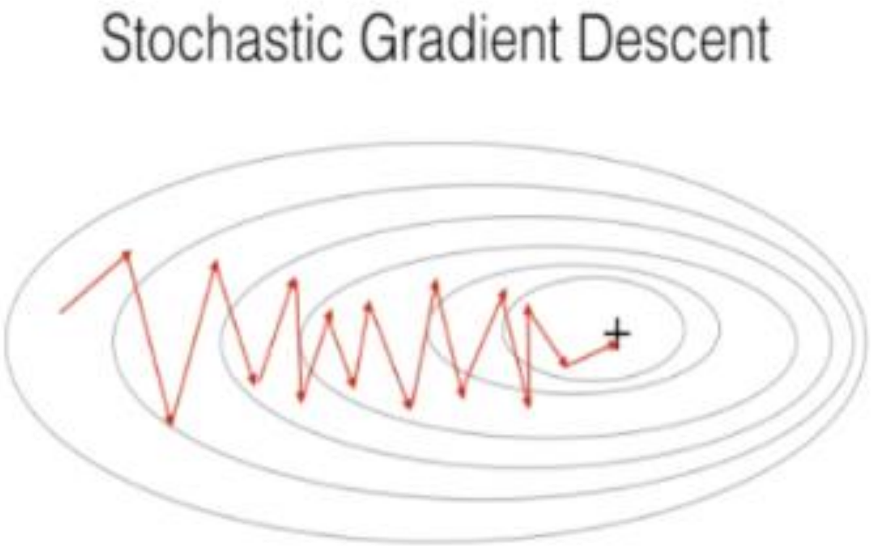
Loss/Cost function = dependent on the Model`s **internal learnable parameters (W, bias)** which are used in computing the target values(Y) from the set of predictors(X).

-> 파라미터 업데이트를 잘하는 알고리즘으로 학습을 잘 시키자!

Optimization

Stochastic Gradient Descent Algorithm

-> 배치 사이즈가 1인 경사하강법 알고리즘



$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

$\mathbf{W}$: 갱신할 가중치 매개변수

$\frac{\partial L}{\partial \mathbf{W}}$: W에 대한 손실함수의 기울기

η - : 학습률(learning rate)

Optimization

Stochastic Gradient Descent Algorithm

단점

1. 적절한 학습률 찾기가 힘들다 -> overshooting!
2. Non-Convex -> local minima에 빠지기 쉽다.

Optimization

Momentum

공이 곡면(기울기)를 따
라 구르듯 움직임

:추진력, 관성
p=m*v

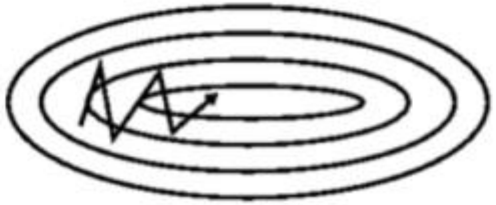
$$V(t)=\gamma V(t-1)+\eta \triangledown J(\theta).$$

and finally we update parameters by $\theta=\theta-V(t)$.

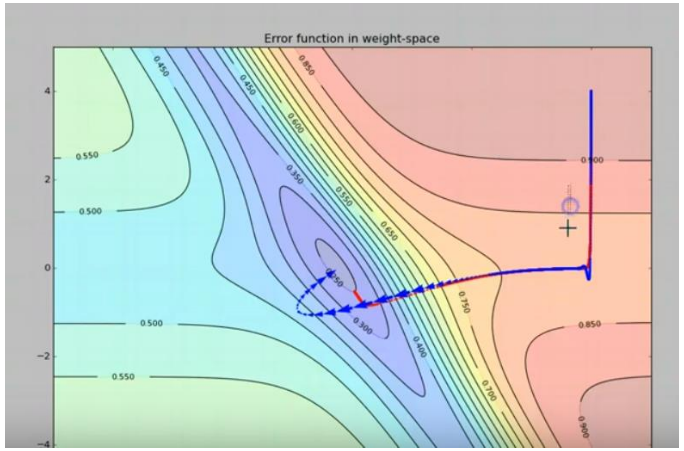
Optimization

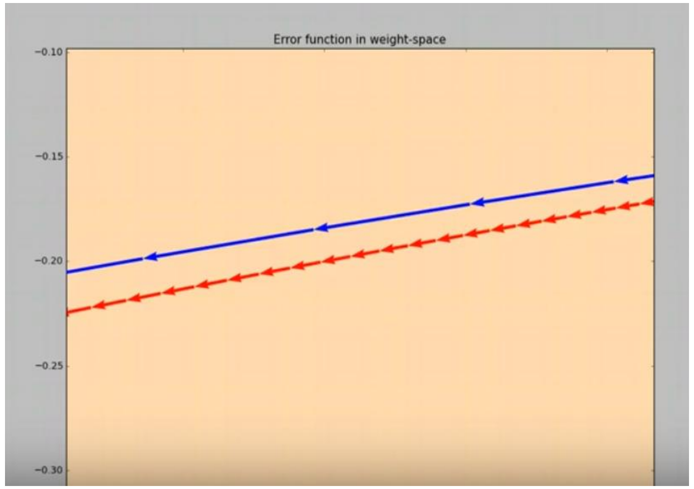
Momentum

Without momentum



With momentum





장점
1. 빠르고 안정적으로 global optima에 수렴한다.
2. 노이즈를 줄여준다.

Optimization

Adagrad

- 과거 기울기를 제곱해서 계속 더함

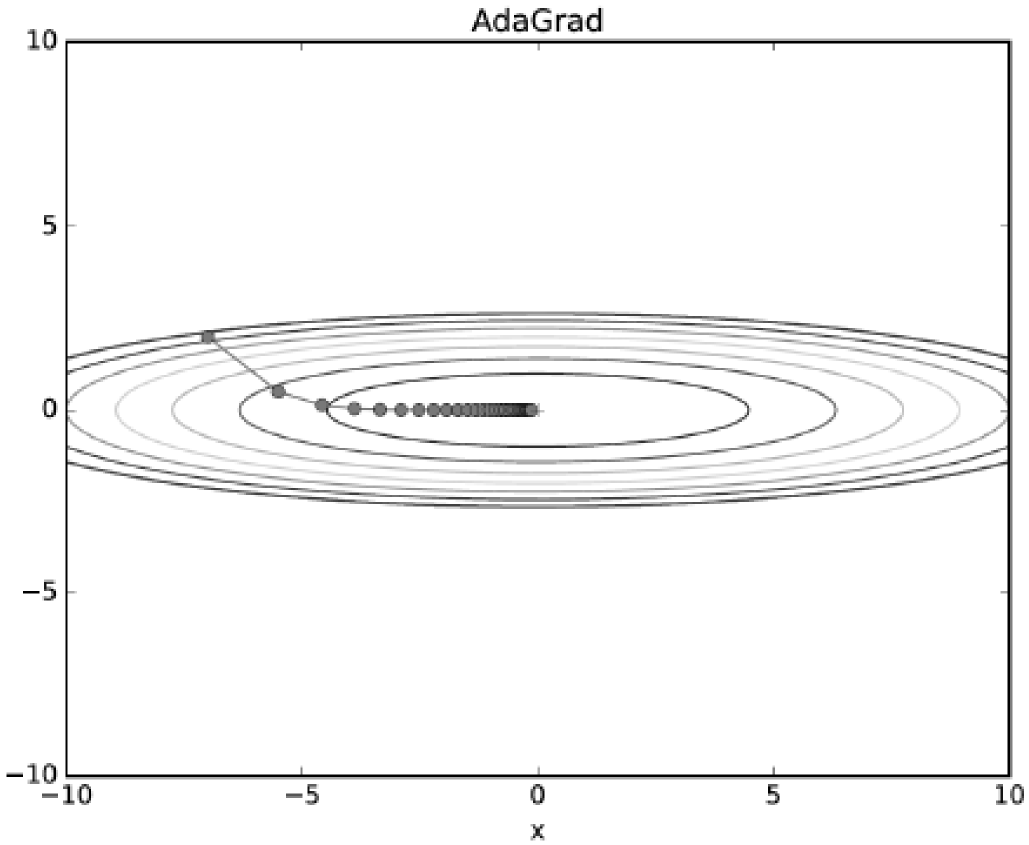
$$\mathbf{h} \leftarrow \mathbf{h} + \frac{\partial L}{\partial \mathbf{W}} \odot \frac{\partial L}{\partial \mathbf{W}}$$

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{1}{\sqrt{\mathbf{h}}} \frac{\partial L}{\partial \mathbf{W}}$$

Adagrad modifies the general learning rate η at each time step t for every parameter θ(i) based on the past gradients that have been computed for θ(i).

Optimization

Adagrad



Optimization

Adagrad

단점:
그레디언트 제곱의 합이 항상 양의 값이기 때문에
트레이닝을 계속 할 수록 합이 점점 커진다.
그게 분모로 들어가면 step size가 점점 작아진다.
결과적으로 모델이 학습을 멈추게 된다.
-> 수렴하기 어렵고 러닝 스피드도 줄어든다.

Optimization

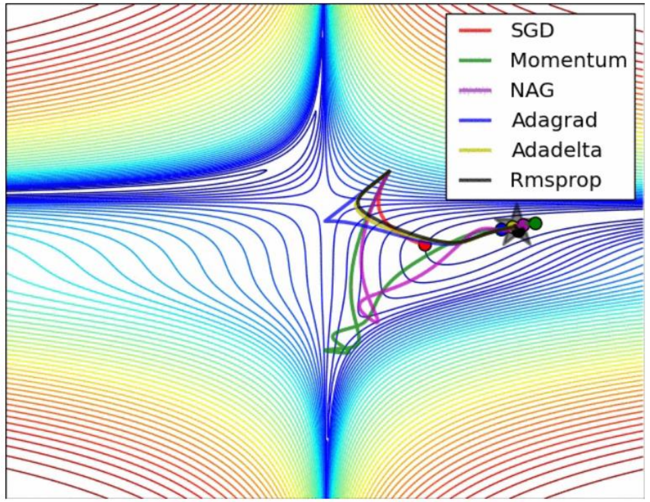
Adam

RMSprop + Momentum
-> 실무에서 작동 good

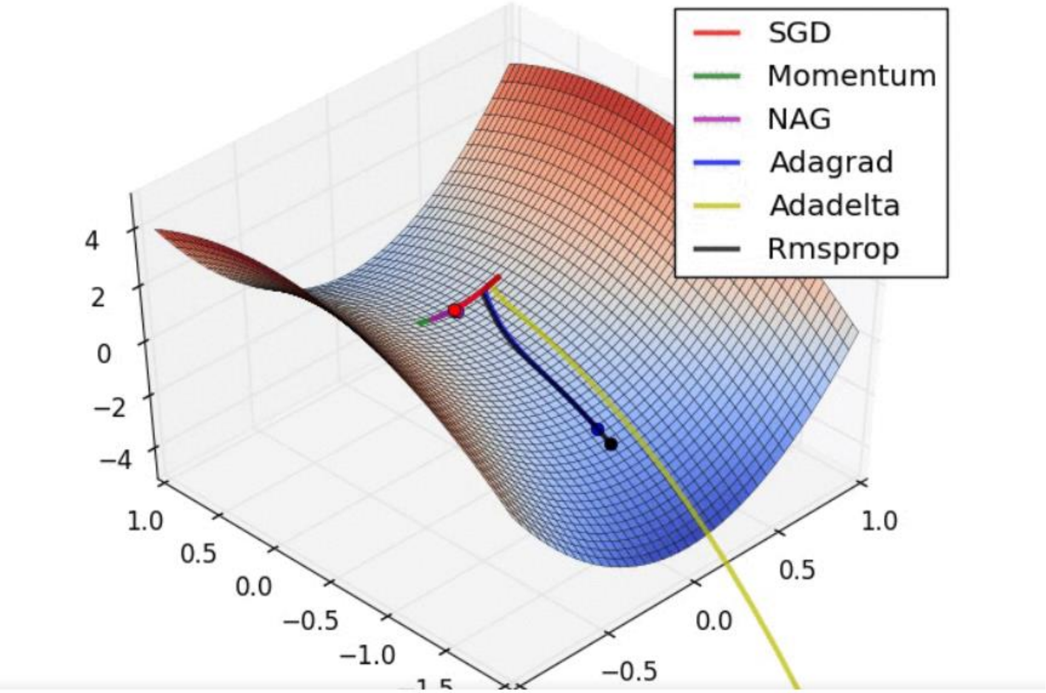
장점:
1. vanishing learning rate 해소
2. slow convergence & high variance in a 세타 update

Optimization

Conclusion



SGD optimization on loss surface contours



Goal: loss최소화를 빠르고 올바른 방향으로 하는 parameter를 찾아주는 최적의 oprimizer 찾기 !

Adaptive algorithms (Adadelta, adagrad) -> 빠른 수렴과 올바른 방향으로 parameter를 업데이트한다.
SGD, Momentum -> 느리고 올바른 수렴 방향 찾지 못함

Optimization

참조

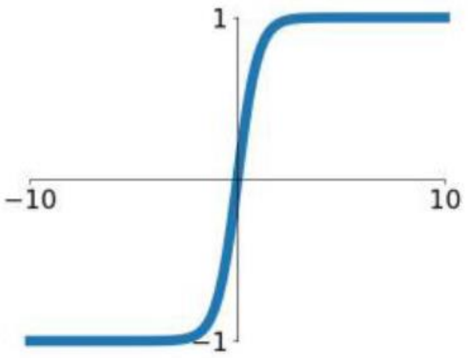
https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f

Weight Initialization

1. initialize with small random numbers
$\mu = 0$, $\sigma = 0.01$

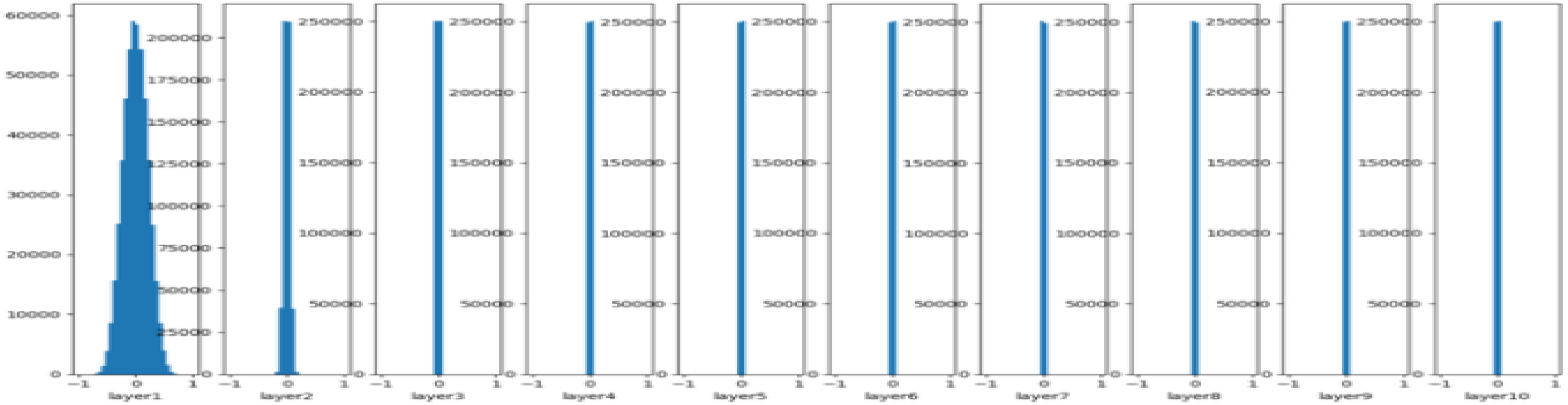
10-layer net
500 neurons on each layer
activation function: tanh non-linearities

```
# assume some unit gaussian 10-D input data
D = np.random.randn(1000, 500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)

act = {'relu':lambda x:np.maximum(0,x), 'tanh':lambda x:np.tanh(x)}
Hs = {}
for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1] # input at this layer
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01 # layer initialization

    H = np.dot(X, W) # matrix multiply
    H = act[nonlinearities[i]](H) # nonlinearity
    Hs[i] = H # cache result on this layer

# look at distributions at each layer
print 'input layer had mean %f and std %f' % (np.mean(D), np.std(D))
layer_means = [np.mean(H) for i,H in Hs.iteritems()]
layer_stds = [np.std(H) for i,H in Hs.iteritems()]
for i,H in Hs.iteritems():
    print 'hidden layer %d had mean %f and std %f' % (i+1, layer_means[i], layer_stds[i])

# plot the means and standard deviations
plt.figure()
plt.subplot(121)
plt.plot(Hs.keys(), layer_means, 'ob-')
plt.title('layer mean')
plt.subplot(122)
plt.plot(Hs.keys(), layer_stds, 'or-')
plt.title('layer std')

# plot the raw distributions
plt.figure()
for i,H in Hs.iteritems():
    plt.subplot(1,len(Hs),i+1)
    plt.hist(H.ravel(), 30, range=(-1,1))
```

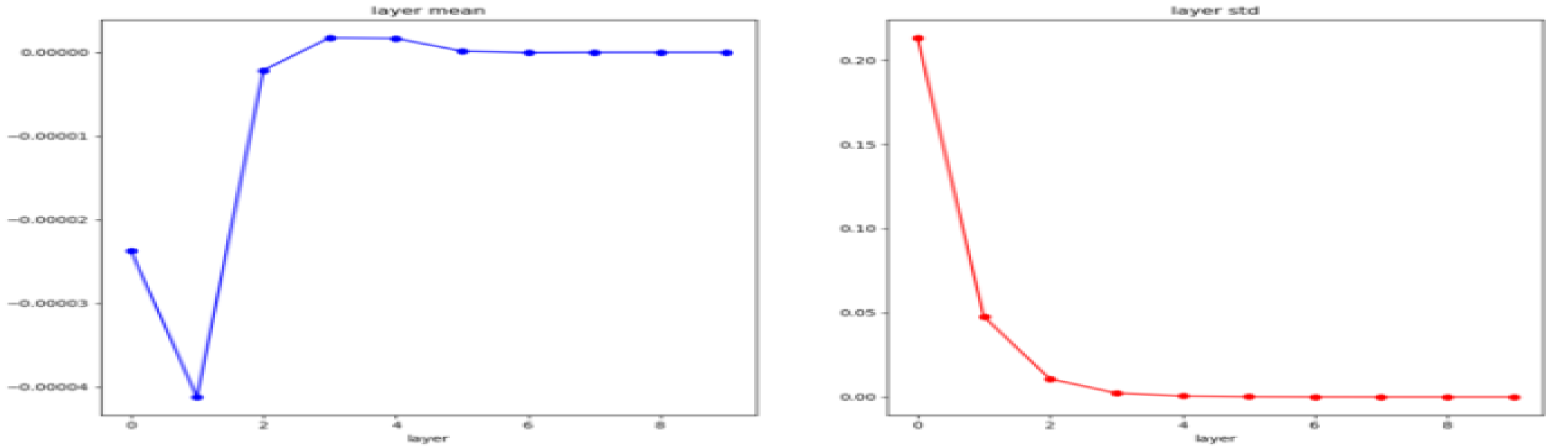


tanh(x)

Weight Initialization



Small random numbers : 히스토그램

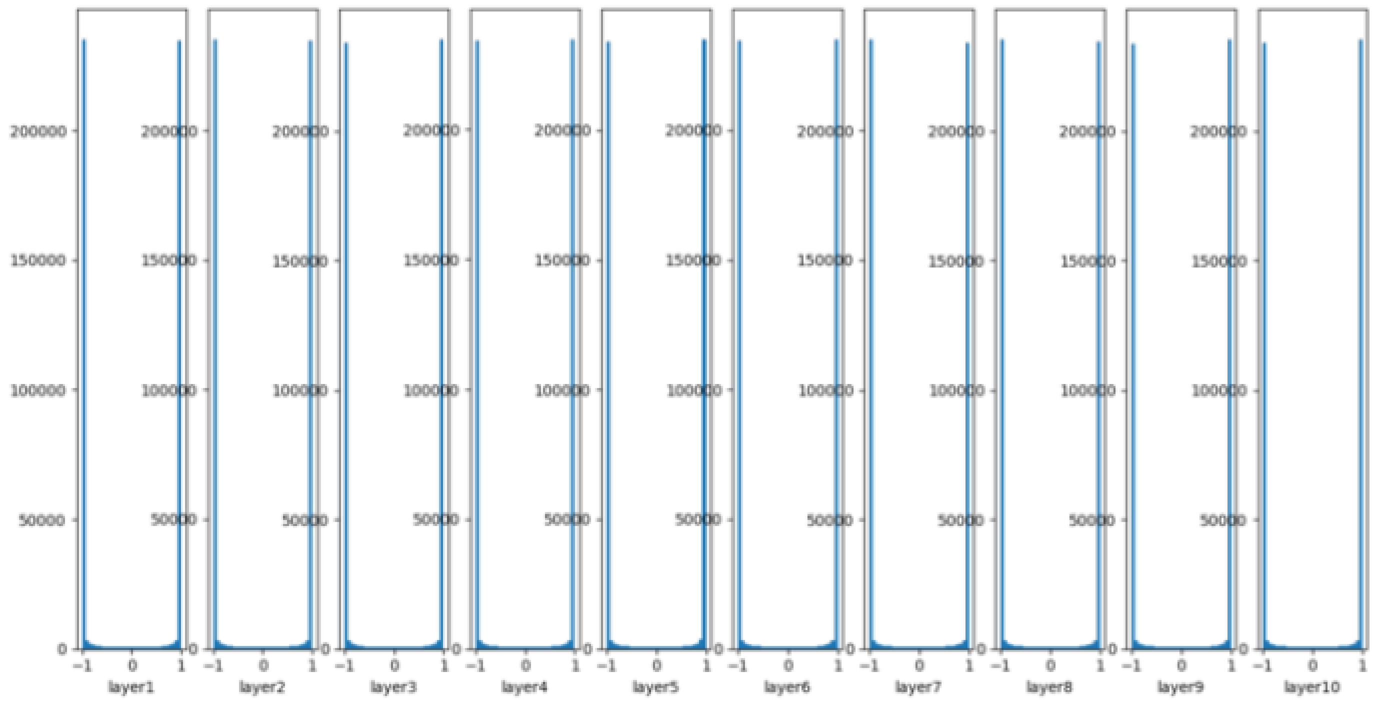


Small random numbers : 평균과 표준편차

Weight Initialization

2 initialize with standard normal distribution

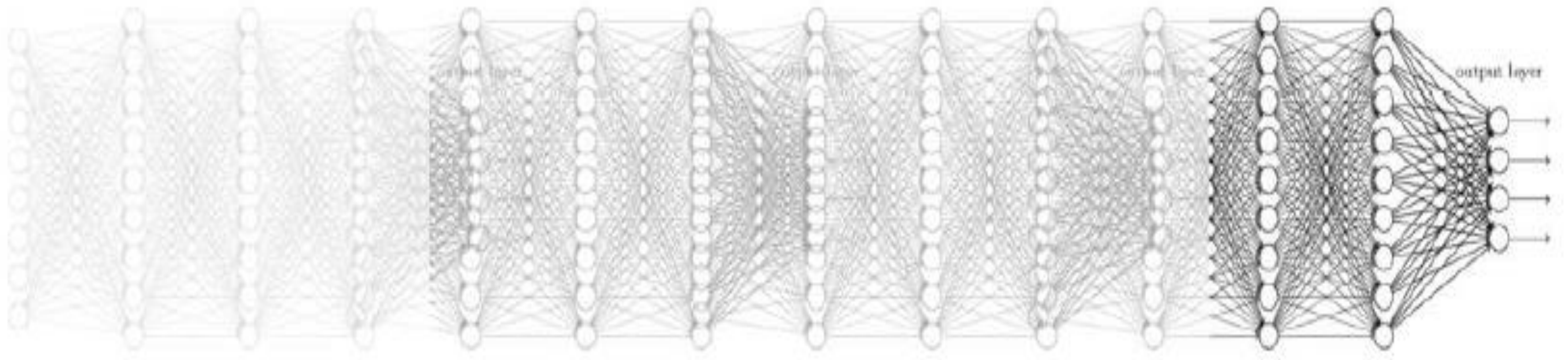
$\mu = 0, \;\; \sigma=1$



가우시안 표준 정규 분포 : 히스토그램

Weight Initialization

Gradient Vanishing Problem

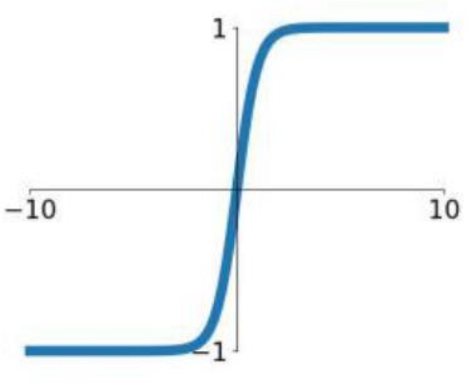


Deep Neural Network
Problem: Gradient goes to '0' -> Optimization is HARD!

Solution:
1) Change Activation Function
2) Weight Initialization

Weight Initialization

Conclusion:
1. W를 표준 정규 분포로 초기화하면 안된다.
2. Gradient vanishing problem해결 위해 결과값이 정규분포 모양을 가져야 한다.



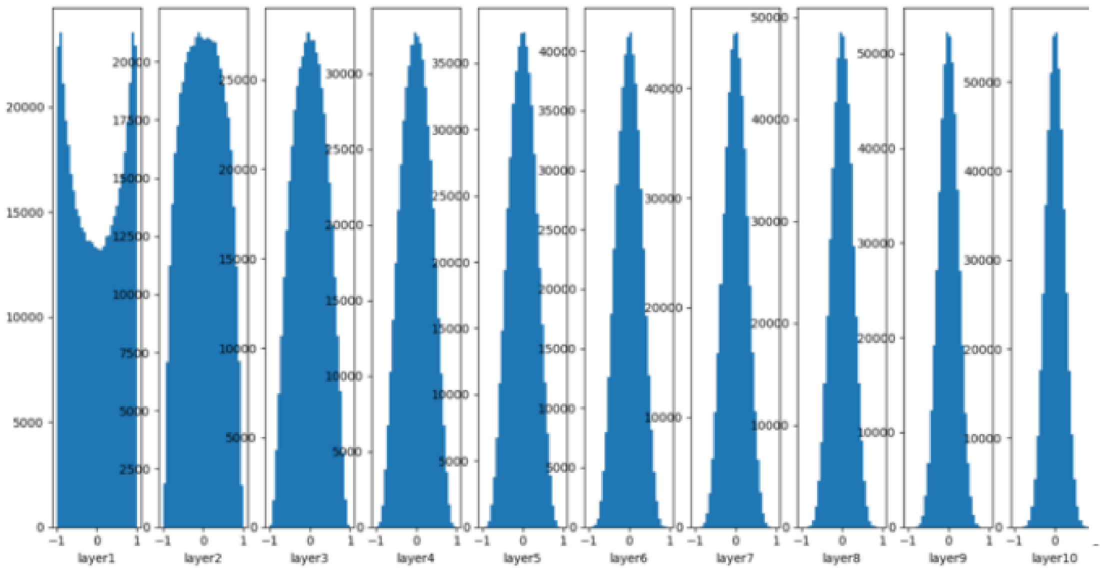
tanh(x)

Weight Initialization

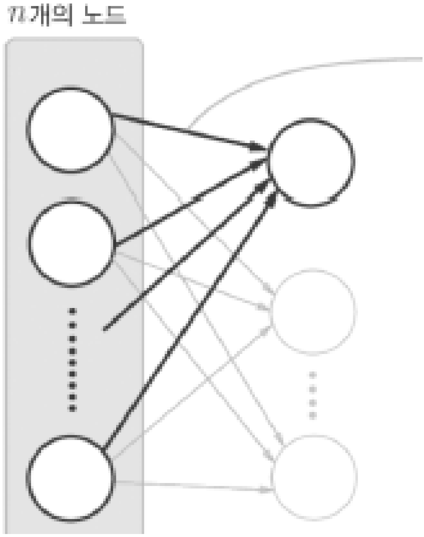
Xavier Initialization
[Glorot et al.. 2010]

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in)
```

-> 표준편차가 1/√n 인 정규분포로 초기화

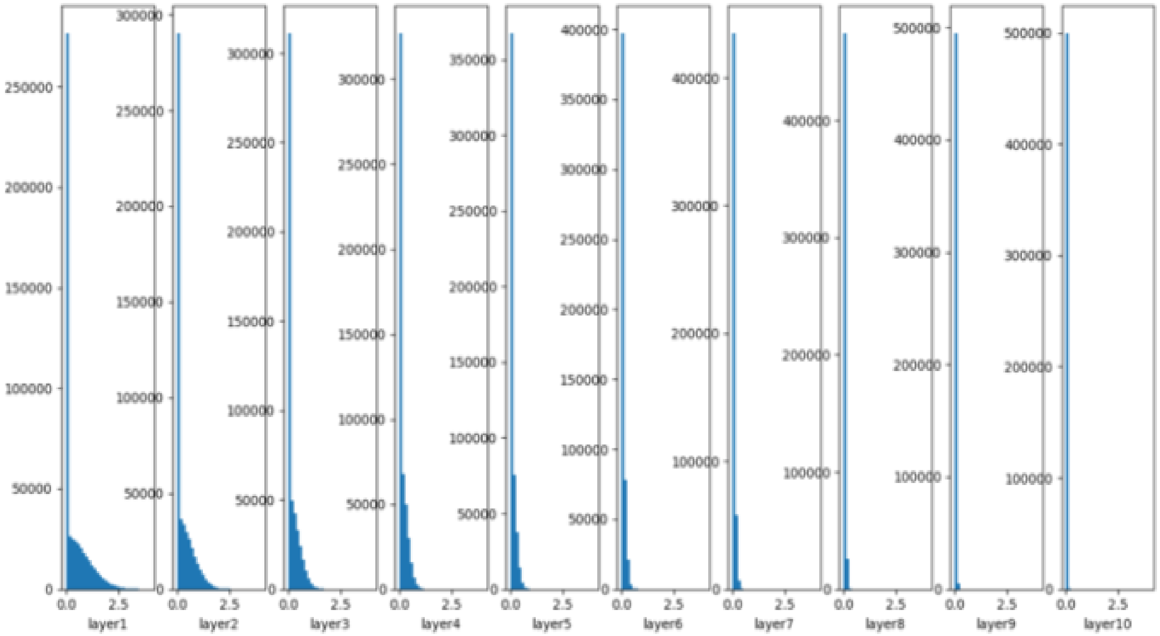


Xavier initialization : 히스토그램

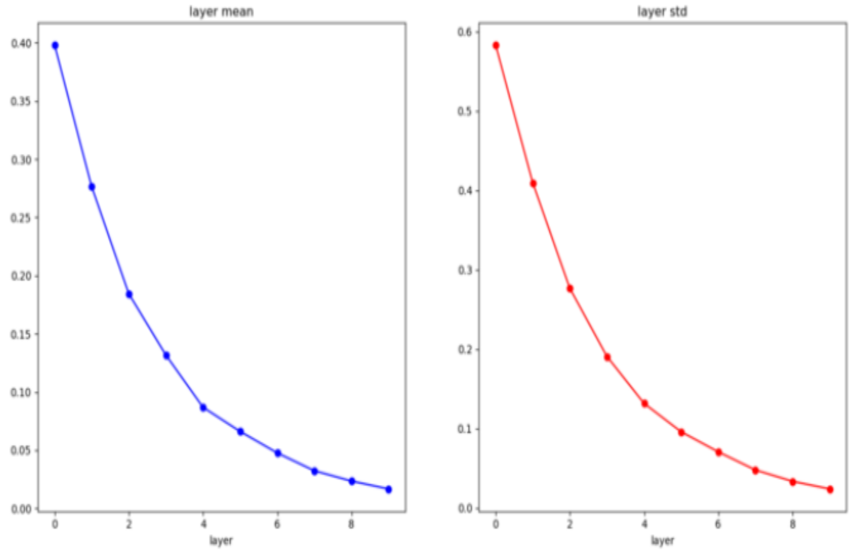


Weight Initialization

But when using 'ReLU' activation function, problem arises!



Xavier initialization + ReLU : 히스토그램

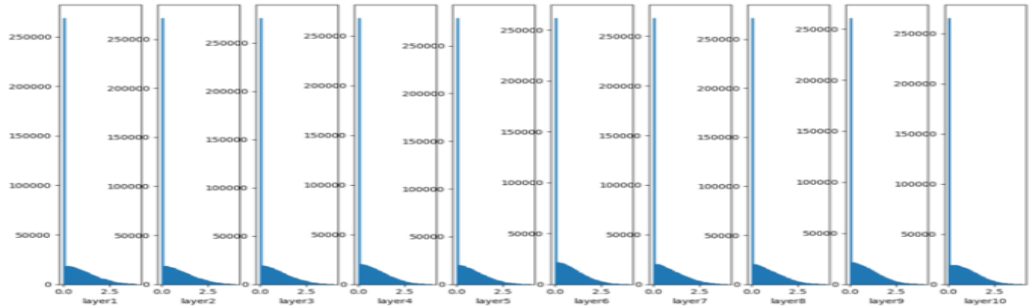


Xavier initialization + ReLU : 평균과 표준편차

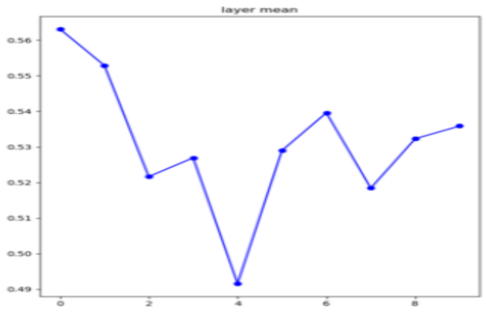
Weight Initialization

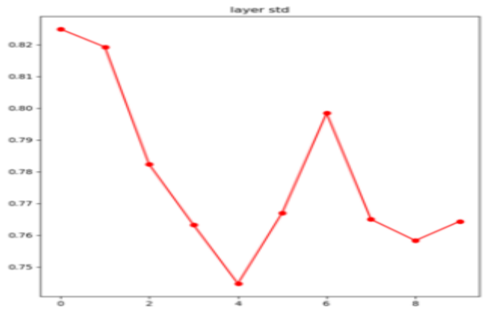
He Initialization
[He et al.. 2015]

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2)
```



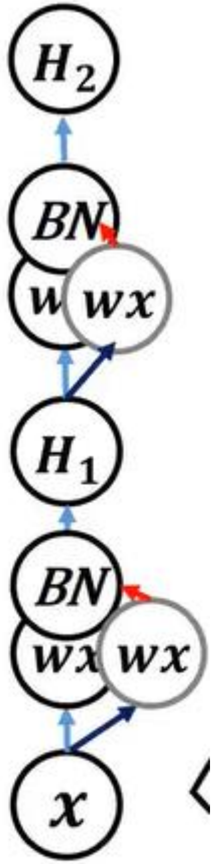
He initialization : 히스토그램





Batch Normalization

-> 배치 단위로 Scaling



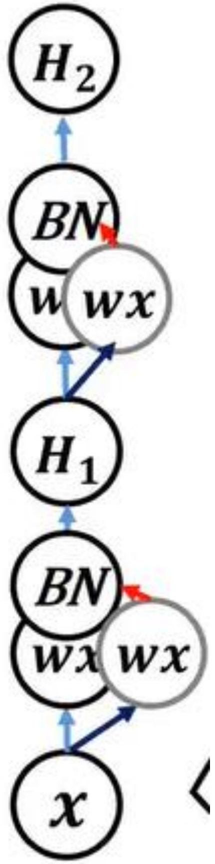
"You want unit gaussian activations? Just make them so!"

->각 층의 활성화 값 분포가
정규분포를 따르면
학습이 원활하게 수행된다.

->강제로
각 layer의
활성화를
적당히
퍼뜨린다.

Batch Normalization

-> 배치 단위로 Scaling



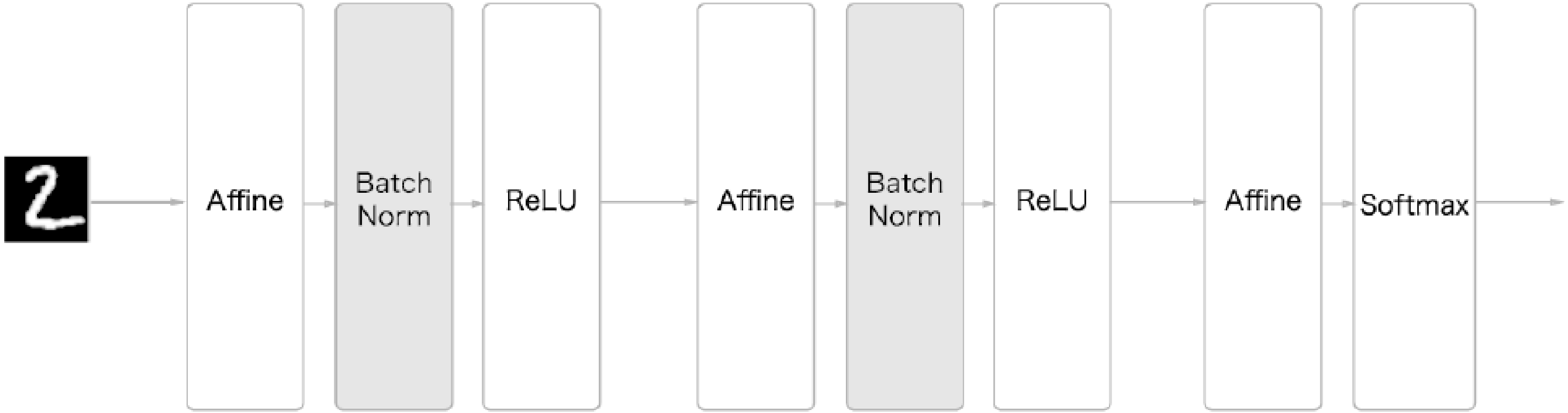
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$ // mini-batch mean

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

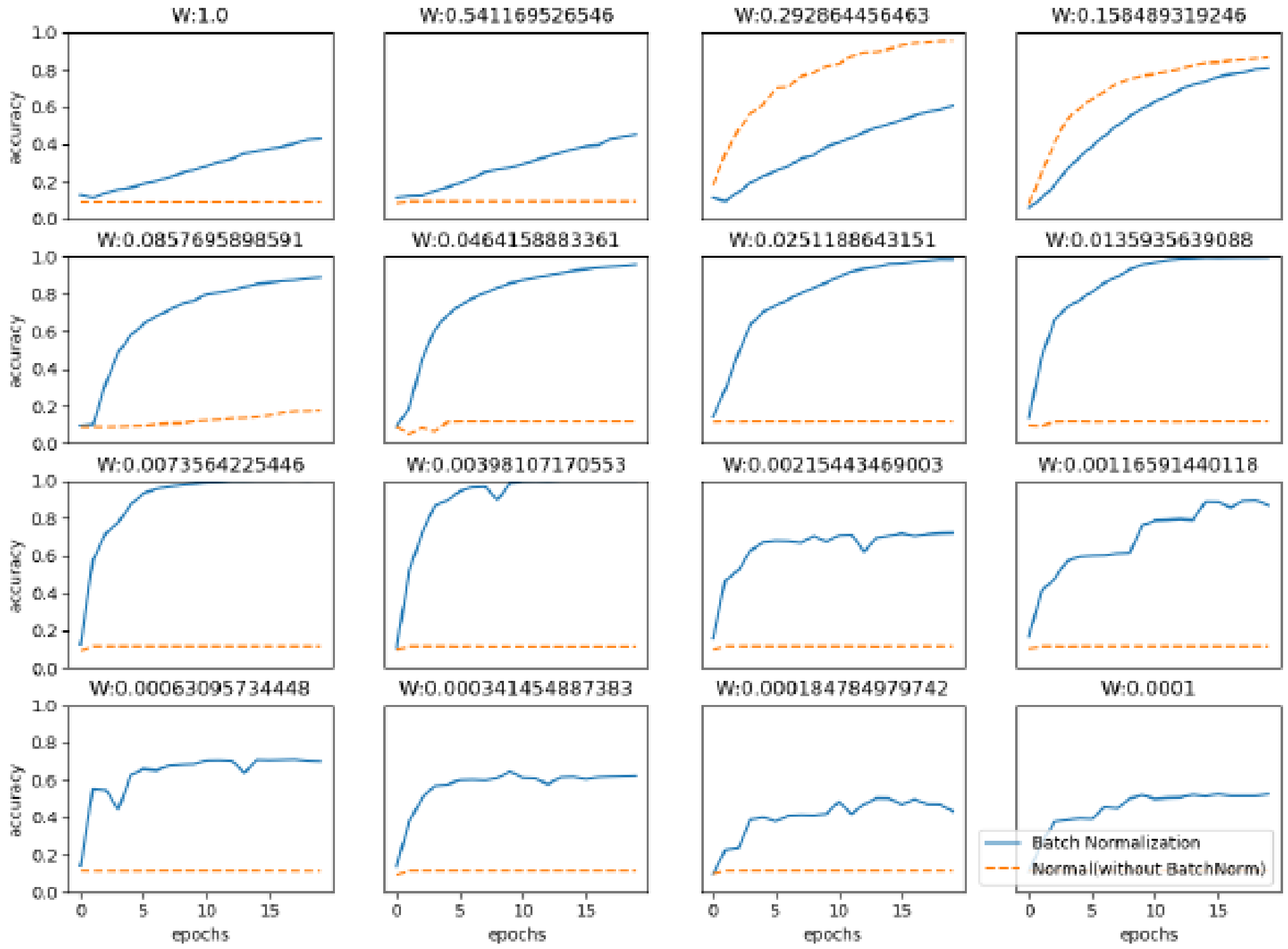
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$ // normalize

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

Batch Normalization



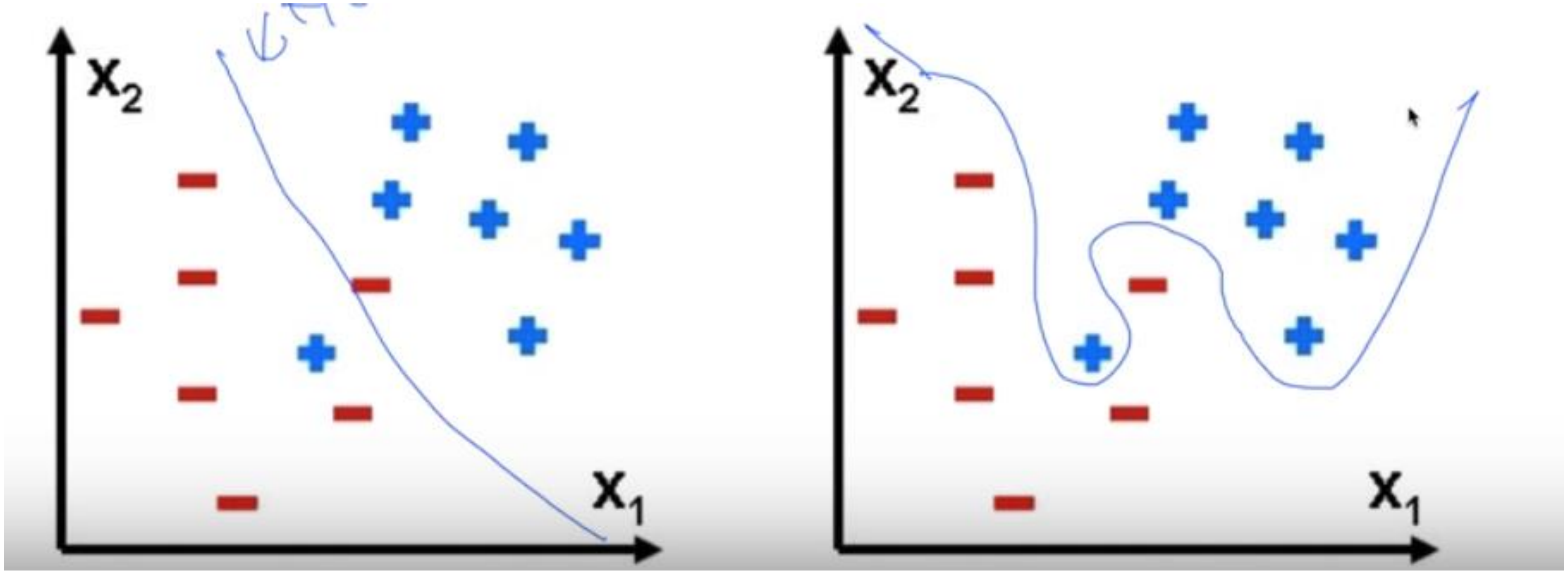
Batch Normalization



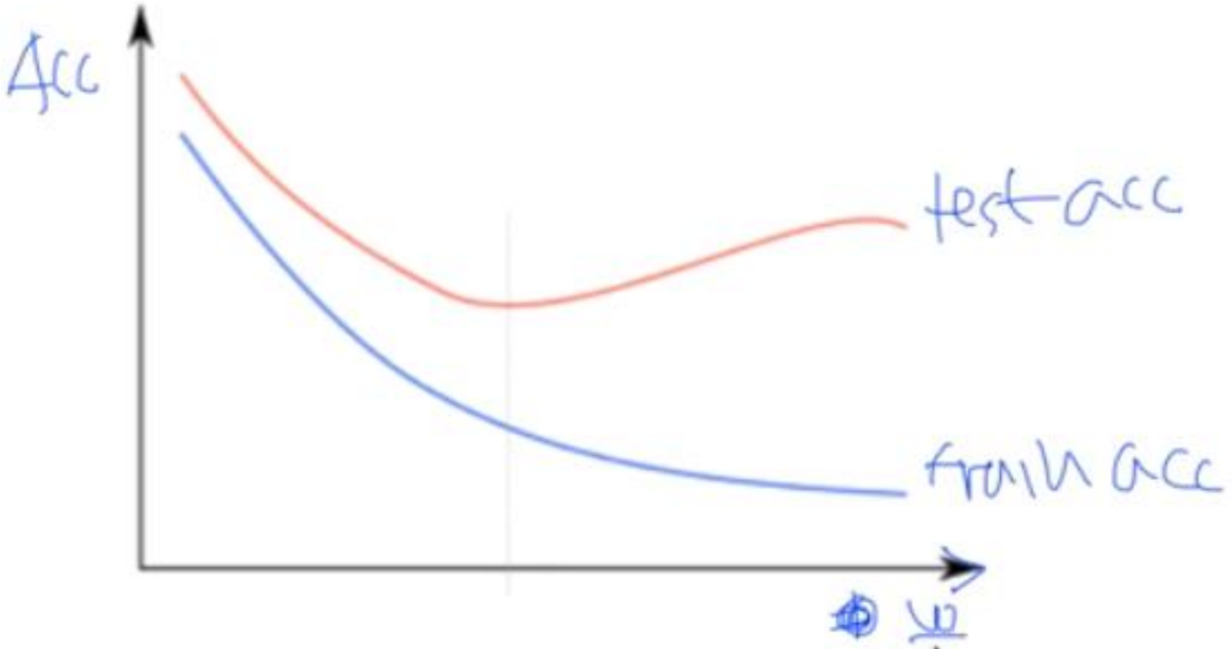
장점

1. Improve gradient flow through the network
2. allows higher learning rate
3. reduces the strong dependence on initialization
모두 목표가 결과값을 정규분포로 만들어서 학습을 잘 하도록 하기 위한 것이기 때문에
4. acts as a form of regularization

Overfitting



Overfitting



- Very high accuracy on the training dataset (eg: 0.99)
- Poor accuracy on the test data set (0.85)

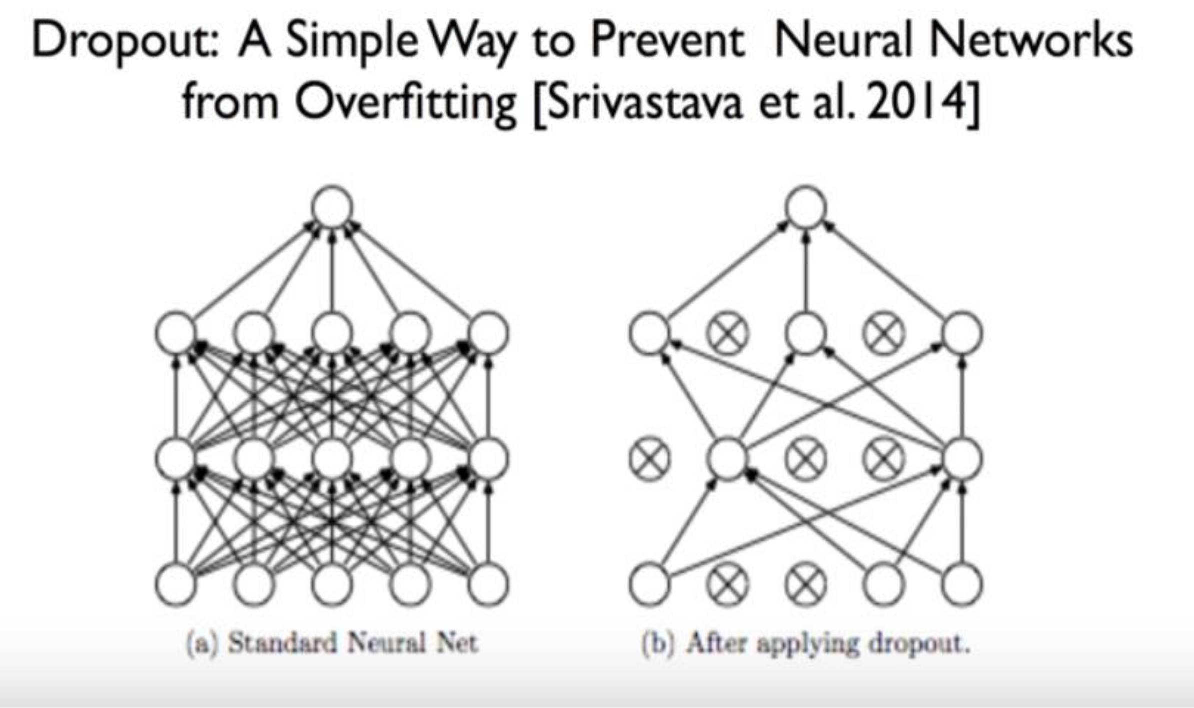
Overfitting

Solution:
1. increase # training set
2. erase # features
3. regularization (v)

$$cost + \lambda \sum W^2$$

Dropout

= Killing nodes
(Randomly set some neurons into zero)



Dropout


Forces the network to have a redundant representation.
has an ear
has a tail
is furry
has claws
mischievous look
cat score

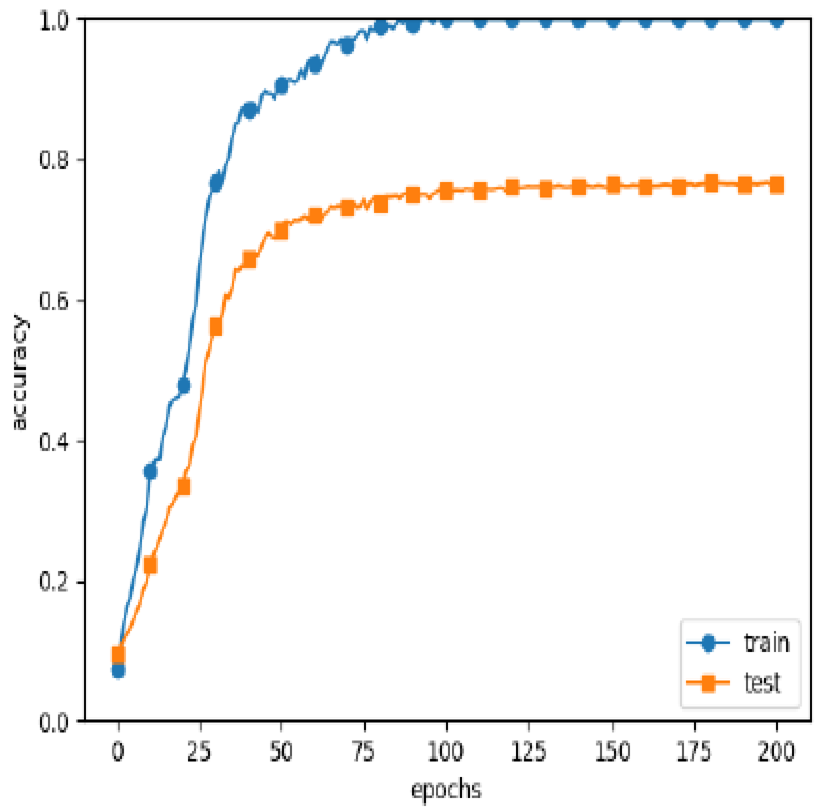
Dropout

TensorFlow implementation

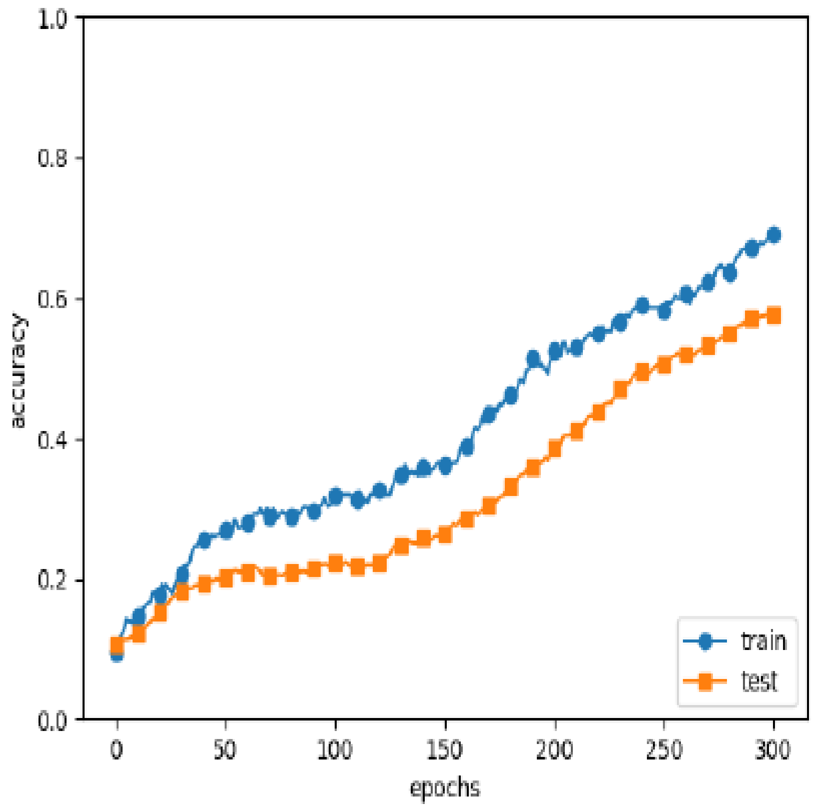
```
dropout_rate = tf.placeholder("float")
_L1 = tf.nn.relu(tf.add(tf.matmul(X, W1), B1))
L1 = tf.nn.dropout(_L1, dropout_rate)
```

```
TRAIN:
  sess.run(optimizer, feed_dict={X: batch_xs, Y: batch_ys,
  dropout_rate: 0.7})

EVALUATION:
  print "Accuracy:", accuracy.eval({X: mnist.test.images, Y:
  mnist.test.labels, dropout_rate: 1})
```

Dropout



1.0
0.8
0.6
0.4
0.2
0.0
accuracy
0
25
50
75
100
125
150
175
200
epochs
train
test



1.0
0.8
0.6
0.4
0.2
0.0
accuracy
0
50
100
150
200
250
300
epochs
train
test