

Cifar10 이미지셋과 CNN 성능향상 기법

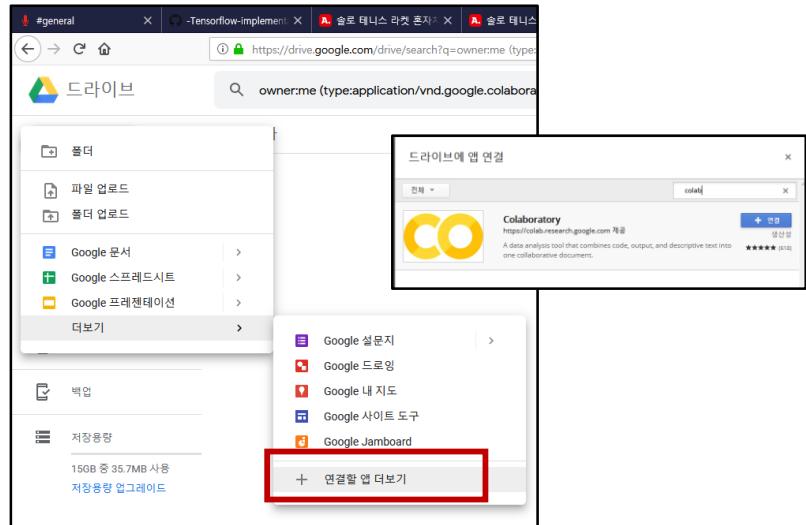
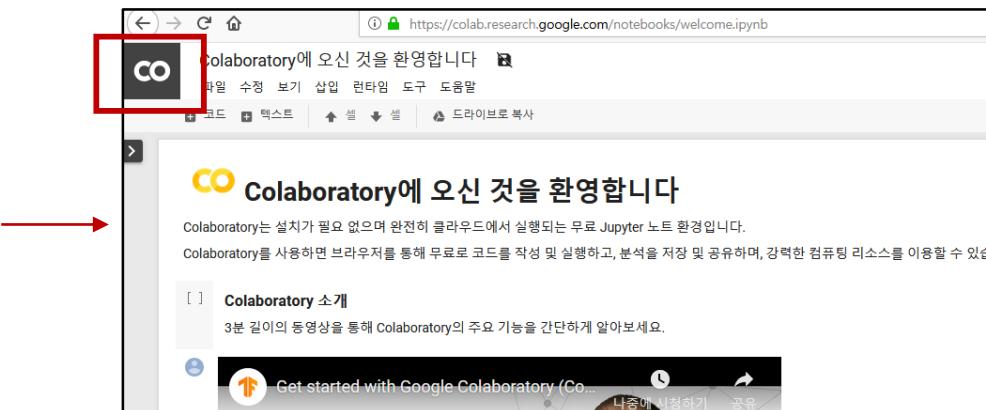
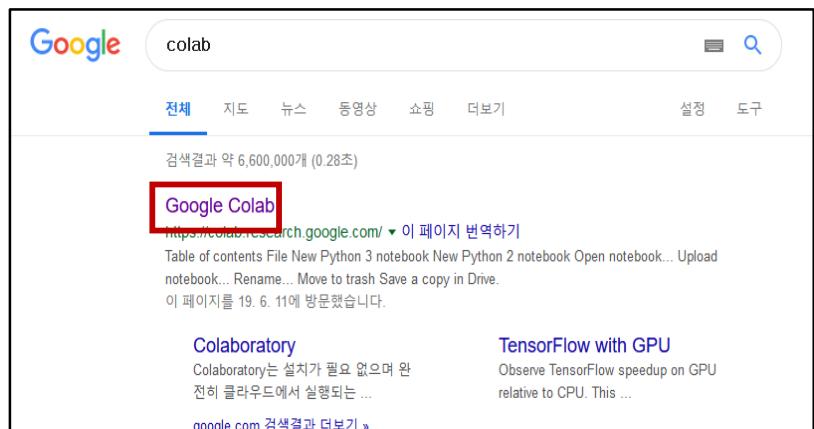
김도현

중앙대학교 소프트웨어학부

이메일: dohyun.cau@gmail.com

- PPT :
 - [workshop_kist_session1_cifar10.pptx]
 - [workshop_kist_session2_tensorflow.pptx]
 - [workshop_kist_session3_GAN.pptx]
- Source code : <https://github.com/ppooiiuuyh/-Tensorflow-implementations/tree/master/KICS-workshop>
 - [1_cnn_basic/cifar10_cnnbasic.py]
 - [2_cnn_deeper/cifar10_cnnbasic_deeper.py]
 - [4_cnn_deeper_init/cifar10_cnndeep_init.py]
 - [7_dropout/cifar10_cnndeep_dropout.py]
- Colab link :
 - [1_cnn_basic] : <https://colab.research.google.com/drive/1I25YqrFO66L-Y4mhNK0eFYEORS1Rg4yv>
 - [2_cnn_deeper] : <https://colab.research.google.com/drive/1qfPFG1hEl2QkavzG-M3tjLPd5i2j8zoG>
 - [4_cnn_deeper_init] : <https://colab.research.google.com/drive/1PpD1gnHlxTP6EebImKddIAkHh95YsJBF>
 - [7_dropout] : <https://colab.research.google.com/drive/1vb0gEf1SFs38qeTzKuNfDB2ejOOHuqN6>

Google CoLAB (GPU지원 online jupyter notebook)



- **강의 목표** : MNIST보다 어려운 Cifar10 데이터셋과 함께 CNN 성능 향상 기법들을 이해하고 구현한다.
- **목차**

1. Cifar10 Image Classification Dataset 소개

2. CNN 성능 향상을 위한 기법들

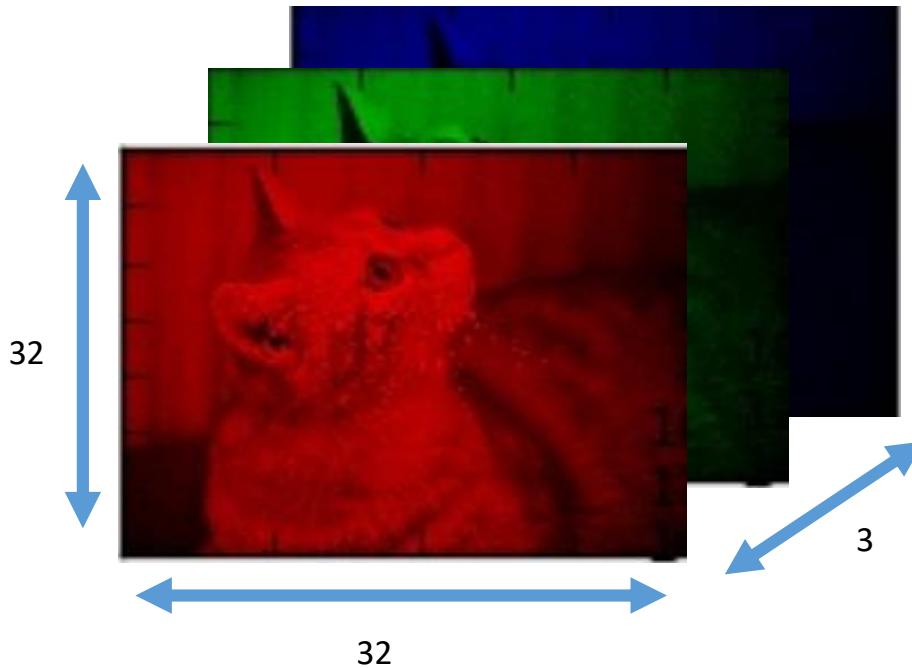
- ReLU
- Initializer
- Optimizer
- Batch Normalization
- Dropout

CIFAR10 Image Classification Dataset

	MNIST	CIFAR-10,100	IMAGENET
		 airplane automobile bird cat deer dog frog horse ship truck	
Num Channel	1 (Gray scale)	3 (R, G, B)	3 (R, G, B)
Num Classes	10	10 , 100	1000
Resolution	28 * 28	32 * 32	(256 * 256)
Num Training Set	50,000	50,000	200,000

CIFAR10 Image Classification Dataset

CIFAR 데이터셋



RGB 3channel 컬러 이미지 이기 때문에 $32 \times 32 \times 3$ 의 행렬로 표현

CIFAR 데이터셋 사용법

```
import tensorflow as tf
import numpy as np
from keras.datasets.cifar10 import load_data

(x_train, y_train), (x_test, y_test) = load_data()
print("x_train.shape :", x_train.shape)
print("y_train.shape :", y_train.shape)

print(x_train[0])
```

MNIST 데이터셋과 마찬가지로 라이브러리 지원.

케라스 설치 : pip install keras

CIFAR10 Image Classification Dataset

CIFAR 데이터셋 사용법

CODE :

```
import tensorflow as tf
import numpy as np
from keras.datasets.cifar10 import load_data

(x_train, y_train), (x_test, y_test) = load_data()

print("x_train.shape :", x_train.shape)
print("y_train.shape :", y_train.shape)

print(x_train[0])
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
  8192/170498071 [=====] - ETA: 1:04:44
  40960/170498071 [=====] - ETA: 25:58
 106496/170498071 [=====] - ETA: 14:59
 221184/170498071 [=====] - ETA: 9:36
 417792/170498071 [=====] - ETA: 5:38
 434176/170498071 [=====] - ETA: 6:07
 860160/170498071 [=====] - ETA: 3:20
 876544/170498071 [=====] - ETA: 3:37
 1728512/170498071 [=====] - ETA: 1:57
 1761280/170498071 [=====] - ETA: 2:05
 3465216/170498071 [=====] - ETA: 1:07
 3547136/170498071 [=====] - ETA: 1:10
 3891200/170498071 [=====] - ETA: 1:12
 7036928/170498071 [>-----] - ETA: 43s
 9658368/170498071 [>-----] - ETA: 34s
12034048/170498071 [=>-----] - ETA: 30s
14721024/170498071 [=>-----] - ETA: 25s
16064512/170498071 [=>-----] - ETA: 24s
16687104/170498071 [=>
```

*데이터셋 저장 위치 : `~/.keras/datasets/`

CIFAR10 Image Classification Dataset

CIFAR 데이터셋 사용법

CODE :

```
import tensorflow as tf
import numpy as np
from keras.datasets.cifar10 import load_data

(x_train, y_train), (x_test, y_test) = load_data()
print("x_train.shape :", x_train.shape)
print("y_train.shape :", y_train.shape)

print(x_train[0])
```

```
x_train.shape : (50000, 32, 32, 3)
y_train.shape : (50000, 1)
[[[ 59  62  63]
  [ 43  46  45]
  [ 50  48  43]
  ...
  [158 132 108]
  [152 125 102]
  [148 124 103]]]
```

```
[[ 16   20   20]
 [  0    0    0]
 [ 18    8    0]
 ...
```

CIFAR10 Image Classification Dataset

CIFAR 데이터셋 Sample

```
import tensorflow as tf
import numpy as np
from keras.datasets.cifar10 import load_data

from matplotlib import pyplot as plt

(x_train, y_train), (x_test, y_test) = load_data()
plt.imshow(x_train[0])
plt.show()
```



CIFAR10 Image Classification Dataset

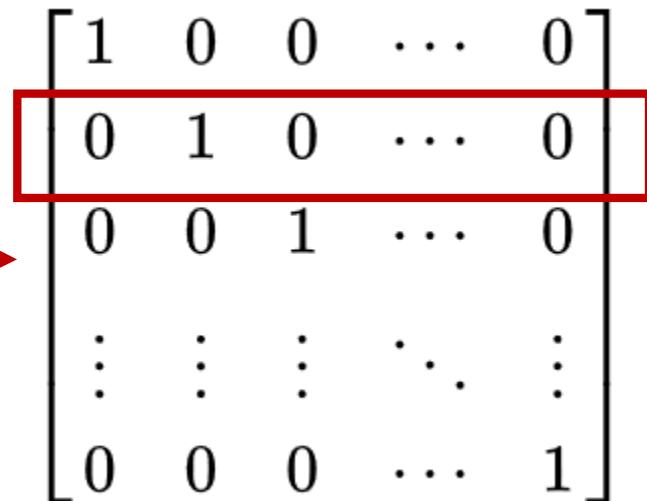
y_train, y_test onehot encoding

Code :

```
if __name__ == "__main__":
#=====
# 1. CIFAR-10 데이터 다운로드 및 데이터 로드
#=====
(x_train, y_train), (x_test, y_test) = load_data()

#one hot encoding
y_train_onehot = np.eye(10)[y_train]
y_test_onehot = np.eye(10)[y_test]
print(y_train_onehot.shape)

y_train_onehot = np.squeeze(y_train_onehot) # (50000,1,10) -> (50000,10)
y_test_onehot = np.squeeze(y_test_onehot) # (10000,1,10) -> (10000,10)
print(y_train_onehot)
```



1	0	0	...	0
0	1	0	...	0
0	0	1	...	0
:	:	:	..	:
0	0	0	...	1

Output :

```
Using TensorFlow backend.
[[4]
 [1]
 [1]
(50000, 1)
[[[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]]
[[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
(50000, 1, 10)
Tensor("concat_111_1:0")
```

```
(50000, 10)
[[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]]
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]]
(50000, 10)
[[[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]]
```

Keras dataset을 사용하는 경우 onehot encoding이 되어있지 않으므로 처리해주어야 한다

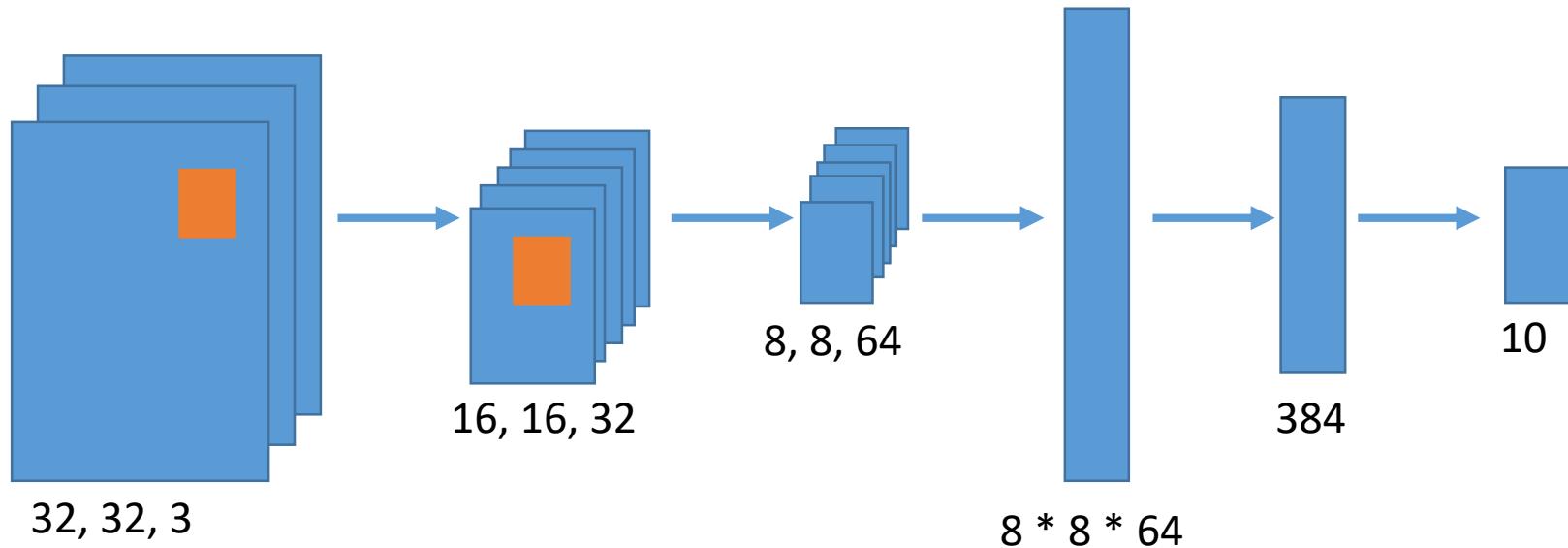
CIFAR10 with Basic CNN

```
1 import tensorflow as tf
2 import numpy as np
3 from keras.datasets.cifar10 import load_data
4
5
6 if __name__ == "__main__":
7 #=====
8 # 1. CIFAR-10 데이터 다운로드 및 데이터 로드
9 #=====
10 (x_train, y_train), (x_test, y_test) = load_data()
11
12 #one hot encoding
13 y_train_onehot = np.eye(10)[y_train]
14 y_test_onehot = np.eye(10)[y_test]
15
16 y_train_onehot = np.squeeze(y_train_onehot) # (50000,1,10) -> (50000,10)
17 y_test_onehot = np.squeeze(y_test_onehot) # (10000,1,10) -> (10000,10)
18
19 #=====
20 # 2. 인풋, 레이블을 입력받기 위한 플레이스홀더 정의
21 #=====
22 x = tf.placeholder(tf.float32, shape=[None, 32, 32, 3])
23 y = tf.placeholder(tf.float32, shape=[None, 10])
24
```

CIFAR10 Image Classification Dataset

CIFAR10 with Basic CNN

```
26  =====  
27  # 3. 모델 정의  
28  =====
```



CIFAR10 Image Classification Dataset

CIFAR10 with Basic CNN

```
26 =====
27 # 3. 모델 정의
28 =====
29     x_image = x
30     W_conv1 = tf.Variable(tf.truncated_normal(shape=[3,3,3, 32], stddev=0.01))
31     b_conv1 = tf.Variable(tf.constant(0.1, shape=[32]))
32     h_conv1 = tf.nn.sigmoid(tf.nn.conv2d(x_image, W_conv1, strides=[1, 1, 1, 1], padding='SAME') + b_conv1)
33     h_pool1 = tf.nn.max_pool(h_conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
34
35     W_conv2 = tf.Variable(tf.truncated_normal(shape=[3,3,32, 64], stddev=0.01))
36     b_conv2 = tf.Variable(tf.constant(0.1, shape=[64]))
37     h_conv2 = tf.nn.sigmoid(tf.nn.conv2d(h_pool1, W_conv2, strides=[1, 1, 1, 1], padding='SAME') + b_conv2)
38     h_pool2 = tf.nn.max_pool(h_conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
39
40     h_conv2_flat = tf.reshape(h_pool2, [-1, 8 * 8 * 64])
41     W_fc1 = tf.Variable(tf.truncated_normal(shape=[8 * 8 * 64, 384], stddev=0.01))
42     b_fc1 = tf.Variable(tf.constant(0.1, shape=[384]))
43     h_fc1 = tf.nn.sigmoid(tf.matmul(h_conv2_flat, W_fc1) + b_fc1)
44
45     W_fc2 = tf.Variable(tf.truncated_normal(shape=[384, 10], stddev=0.01))
46     b_fc2 = tf.Variable(tf.constant(0.1, shape=[10]))
47     logits = tf.matmul(h_fc1, W_fc2) + b_fc2
48     y_pred = tf.nn.softmax(logits)
```

CIFAR10 with Basic CNN

```
51  =====
52  # 4. 비용함수 정의
53  =====
54      loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=logits))
55      train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)
56
57      # 정확도를 계산하는 연산.
58      correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y, 1))
59      accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
60
61
```

CIFAR10 Image Classification Dataset

CIFAR10 with Basic CNN

```
62  =====
63  # 5. 학습
64  =====
65      with tf.Session() as sess:
66      #-----
67      # 5.1 세션, 변수 초기화
68      #-----
69          sess.run(tf.global_variables_initializer())
70
71      #-----
72      # 5.2 Training loop
73      #-----
74          total_epoch = 300
75          for e in range(total_epoch):
76              #.....
77              # 5.2.1 학습
78              #.....
79              total_size = x_train.shape[0]
80              batch_size = 128
```

CIFAR10 with Basic CNN

```
82         loss_list = []
83         train_accuracy_list = []
84         for i in range(int(total_size / batch_size)):
85             #== batch load
86             batch_x = x_train[i*batch_size:(i+1)*batch_size]
87             batch_y = y_train_onehot[i*batch_size:(i+1)*batch_size]
88
89             #== train
90             sess.run(train_step, feed_dict={x: batch_x, y: batch_y})
91
92             #== logging
93             train_accuracy = accuracy.eval(feed_dict={x: batch_x, y: batch_y})
94             loss_print = loss.eval(feed_dict={x: batch_x, y: batch_y})
95             train_accuracy_list.append(train_accuracy)
96             loss_list.append(loss_print)
97
98             print("반복(Epoch):", e, "트레이닝 데이터 정확도:", np.mean(train_accuracy_list),
```

CIFAR10 with Basic CNN

```
100      # .....
101      # 5.2.2 평가
102      # .....
103      # 매epoch 마다 test 데이터셋에 대한 정확도와 loss를 출력.
104          test_total_size = x_test.shape[0]
105          test_batch_size = 128
106
107          test_accuracy_list = []
108          for i in range(int(test_total_size / test_batch_size)):
109              #== test batch load
110              test_batch_x = x_test[i*test_batch_size:(i+1)*test_batch_size]
111              test_batch_y = y_test_onehot[i*test_batch_size:(i+1)*test_batch_size]
112
113              #== logging
114              test_accuracy = accuracy.eval(feed_dict={x: test_batch_x, y: test_batch_y})
115              test_accuracy_list.append(test_accuracy)
116              print("테스트 데이터 정확도:",np.mean(test_accuracy_list))
117              print()
```

CIFAR10 Image Classification Dataset

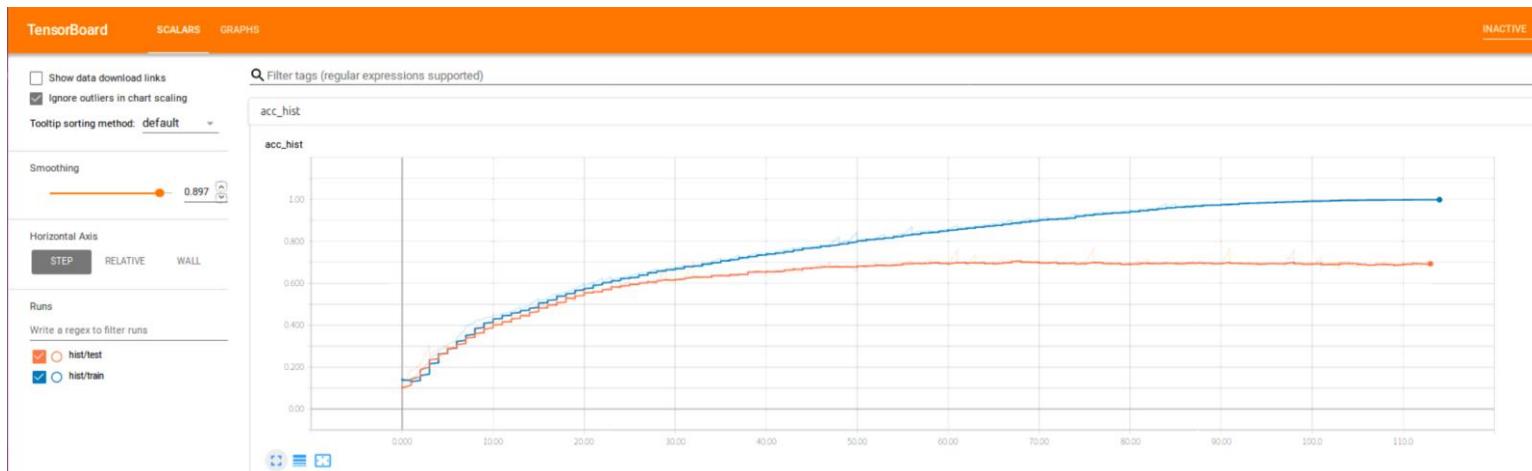
CIFAR10 with Basic CNN : Result

반복(Epoch): 61 트레이닝 데이터 정확도: 0.8639824 손실 함수(loss): 0.45344216
테스트 데이터 정확도: 0.6916066

반복(Epoch): 62 트레이닝 데이터 정확도: 0.86604565 손실 함수(loss): 0.44477302
테스트 데이터 정확도: 0.6885016

반복(Epoch): 63 트레이닝 데이터 정확도: 0.87059295 손실 함수(loss): 0.43496877
테스트 데이터 정확도: 0.69340944

반복(Epoch): 64 트레이닝 데이터 정확도: 0.8745793 손실 함수(loss): 0.4233058
테스트 데이터 정확도: 0.68880206



- 목차

1. Cifar10 Image Classification Dataset 소개

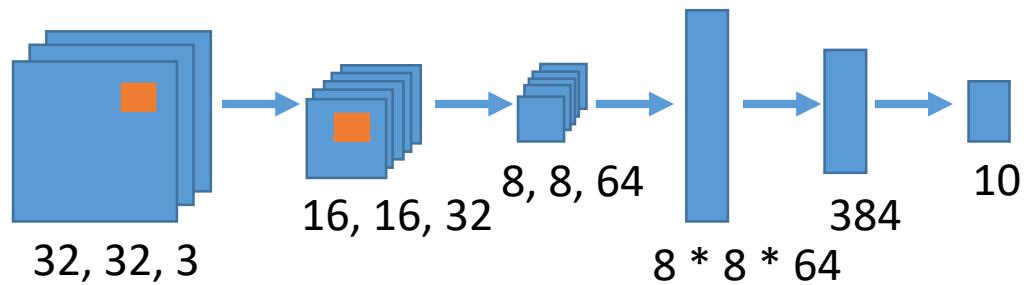
2. CNN 성능 향상을 위한 기법들 ←

- ReLU
- Initializer
- Optimizer
- Batch Normalization
- Dropout
- Regularization

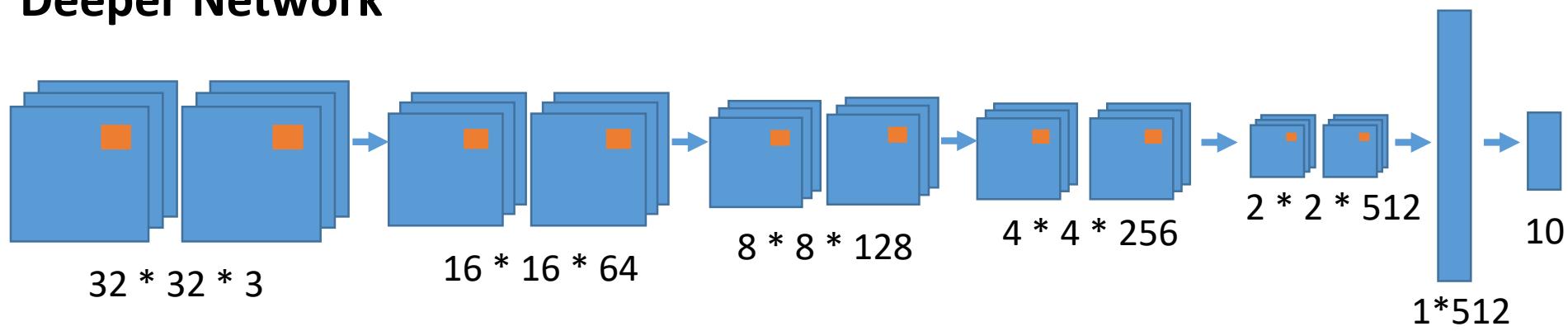
(Advanced)

- Data Augmentation
- Ensemble

기존 Network



Deeper Network



CNN 성능 향상 기법 : Deeper Network

```
26 #=====
27 # 3. 모델 정의
28 #=====
29     x_image = x
30
31
32     def conv(X, in_ch, out_ch, name):
33         with tf.variable_scope(name) as scope:
34             W_conv = tf.Variable(tf.truncated_normal(shape=[3, 3, in_ch, out_ch], stddev=0.01))
35             b_conv = tf.Variable(tf.constant(0.1, shape=[out_ch]))
36             h_conv = tf.nn.sigmoid(tf.nn.conv2d(X, W_conv, strides=[1, 1, 1, 1], padding='SAME') + b_conv)
37             return h_conv
38
39     h_conv1 = conv(x_image, 3, 64, "Conv1")
40     h_conv2 = conv(h_conv1, 64, 64, "Conv2")
41     h_conv2_pool = tf.nn.max_pool(h_conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
42
43     h_conv3 = conv(h_conv2_pool, 64, 128, "Conv3")
44     h_conv4 = conv(h_conv3, 128, 128, "Conv4")
45     h_conv4_pool = tf.nn.max_pool(h_conv4, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
46
47     h_conv5 = conv(h_conv4_pool, 128, 256, "Conv5")
48     h_conv6 = conv(h_conv5, 256, 256, "Conv6")
49     h_conv6_pool = tf.nn.max_pool(h_conv6, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
```

CNN 성능 향상 기법 : Deeper Network

```
51     h_conv7 = conv(h_conv6_pool, 256, 512, "Conv7")
52     h_conv8 = conv(h_conv7, 512, 512, "Conv8")
53     h_conv8_pool = tf.nn.max_pool(h_conv8, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
54
55     h_conv9 = conv(h_conv8_pool, 512, 512, "Conv9")
56     h_conv10 = conv(h_conv9, 512, 512, "Conv10")
57     h_conv10_pool = tf.nn.max_pool(h_conv10, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
58
59     h_conv10_pool_flat = tf.reshape(h_conv10_pool, [-1, 1 * 1 * 512])
60     W_fc = tf.Variable(tf.truncated_normal(shape=[512, 10], stddev=0.01))
61     b_fc = tf.Variable(tf.constant(0.1, shape=[10]))
62     logits = tf.matmul(h_conv10_pool_flat, W_fc) + b_fc
63     y_pred = tf.nn.softmax(logits)
64
```

Deeper Network : Result

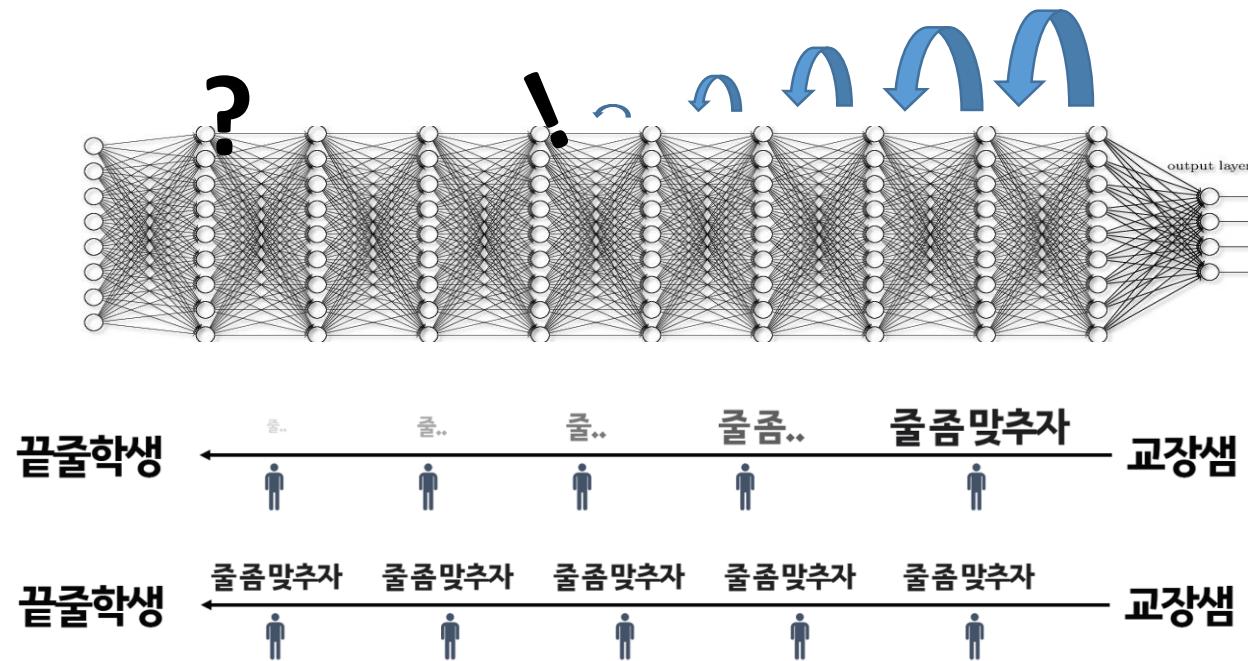
반복(Epoch): 0 트레이닝 데이터 정확도: 0.10078125 손실 합수(loss): 2.3039994
테스트 데이터 정확도: 0.099859774

반복(Epoch): 1 트레이닝 데이터 정확도: 0.099679485 손실 합수(loss): 2.3039432
테스트 데이터 정확도: 0.099859774

반복(Epoch): 2 트레이닝 데이터 정확도: 0.09969952 손실 합수(loss): 2.3039484
테스트 데이터 정확도: 0.099859774

학습이 진행되지 않음.

경사 감소 소멸의 원인

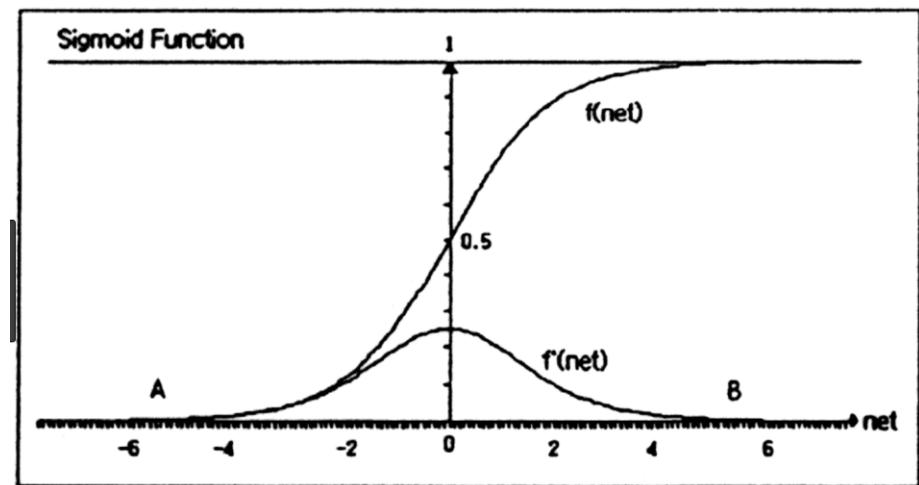


- 역전파는 미분의 반복적 수행
- 하지만 연속적인 미분으로 경사(gradient)가 소멸되어 버려서 미처 최전단의 파라미터까지 학습이 되지 못함

경사 감소 소멸의 원인 : Sigmoid 활성화 함수

- Sigmoid 활성화 함수는 이러한 경사감소를 가속화 시킨다.

$$\begin{aligned}
 F(x) &= \frac{1}{1+e^{-x}} &< f(x) \\
 F'(x) &= \frac{f'(x)g(x) - f(x)g'(x)}{g^2(x)} \\
 &= \frac{0 - (-e^{-x})}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} \frac{e^{-x}}{1+e^{-x}} \\
 &= F(x) \frac{1+e^{-x} - 1}{1+e^{-x}} = F(x) \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}} \right) \\
 &= F(x)(1 - F(x))
 \end{aligned}$$



(a) Sigmoid 함수의 미분

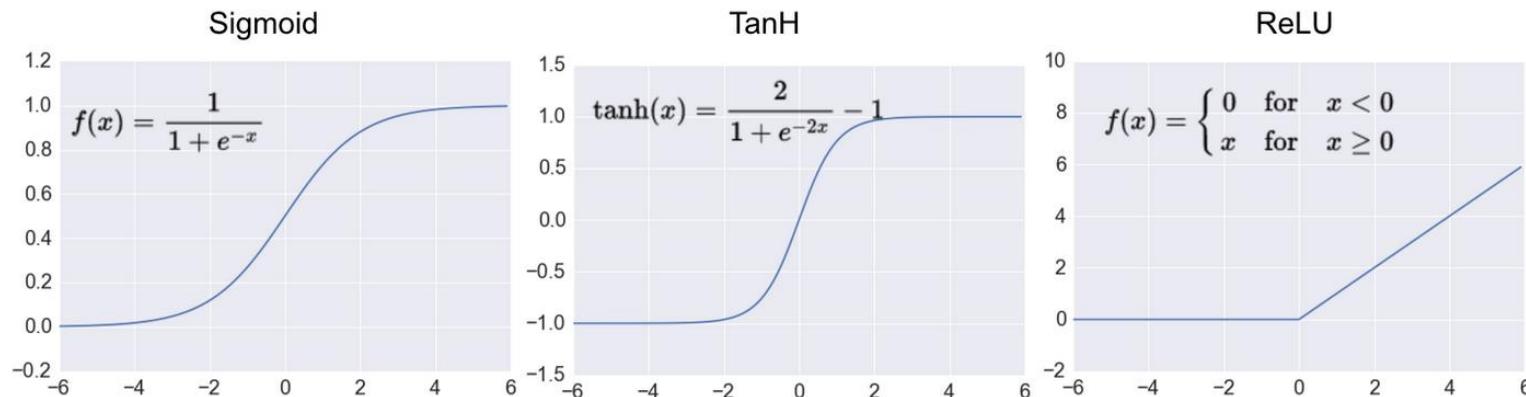
(b) Sigmoid 함수의 그래프

Sigmoid 함수는 미분할수록 그 값이 급격하게 작아진다

경사 감소 소멸의 원인 : Sigmoid 활성화 함수

- 해결 방안 1 : 활성화 함수를 사용하지 않는다. (불가능)
- 해결 방안 2 : Sigmoid의 역할을 하면서도 미분해도 값이 작아지지 않는 다른 활성화 함수를 사용한다.

=> ReLU 활성화 함수, tanh 활성화 함수



- 목차

- 1. Cifar10 Image Classification Dataset 소개**

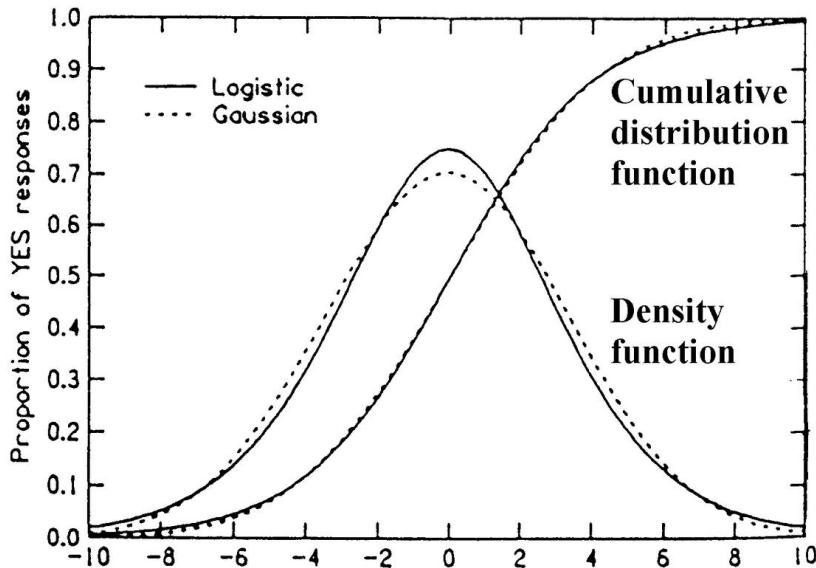
- 2. CNN 성능 향상을 위한 기법들**

- ReLU 
- Initializer
- Optimizer
- Batch Normalization
- Dropout
- Regularization

(Advanced)

- Data Augmentation
- Ensemble

Sigmoid 활성화 함수의 사용 이유 : 계단함수



$$\begin{aligned}
 F(x) &= \frac{1}{1+e^{-x}} &< f(x) \\
 &< g(x) \\
 F'(x) &= \frac{f'(x)g(x)-f(x)g'(x)}{g^2(x)} \\
 &= \frac{0-(-e^{-x})}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} \\
 &= F(x) \frac{1+e^{-x}-1}{1+e^{-x}} = F(x) \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}} \right) \\
 &= F(x)(1-F(x))
 \end{aligned}$$

Sigmoid 함수의 미분은 자기 자신으로 재귀적 표현 가능

$$f(x) = \int_{-\infty}^x \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

따라서 미분은 훨씬 간단하지만 기능적으로는 거의 동일한 Sigmoid 함수를 개발

Sigmoid 활성화 함수의 사용 이유 : 계단함수

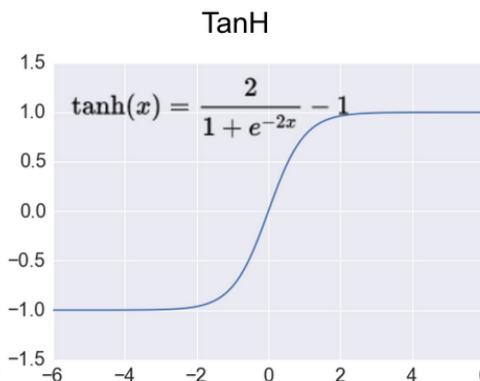
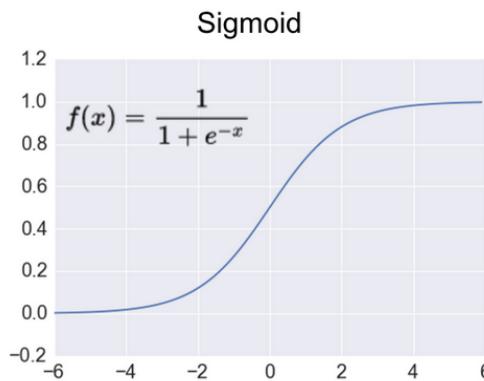
- 하지만 Sigmoid 함수는 경사감소 소실 문제를 가속화 시킴

=> 따라서 아래와 같은 대체 가능한 함수 필요

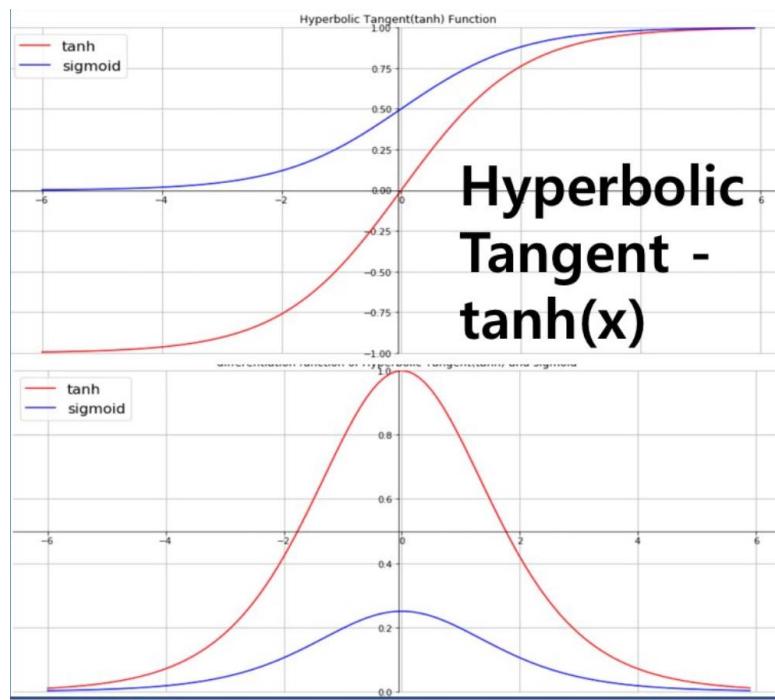
- 확률적 표현이 가능하고
- 미분이 간단하고
- 비선형 함수이며
- 미분해도 값이 줄어들지 않아야 한다

$$\begin{aligned}
 F(x) &= \frac{1}{1+e^{-x}} &< f(x) \\
 &< g(x) \\
 F'(x) &= \frac{f'(x)g(x)-f(x)g'(x)}{g^2(x)} \\
 &= \frac{0-(-e^{-x})}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} \frac{e^{-x}}{1+e^{-x}} \\
 &= F(x) \frac{1+e^{-x}-1}{1+e^{-x}} = F(x) \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}} \right) \\
 &= F(x)(1-F(x))
 \end{aligned}$$

=> 새로운 활성화 함수 tanh (hyperbolic tangent) 개발

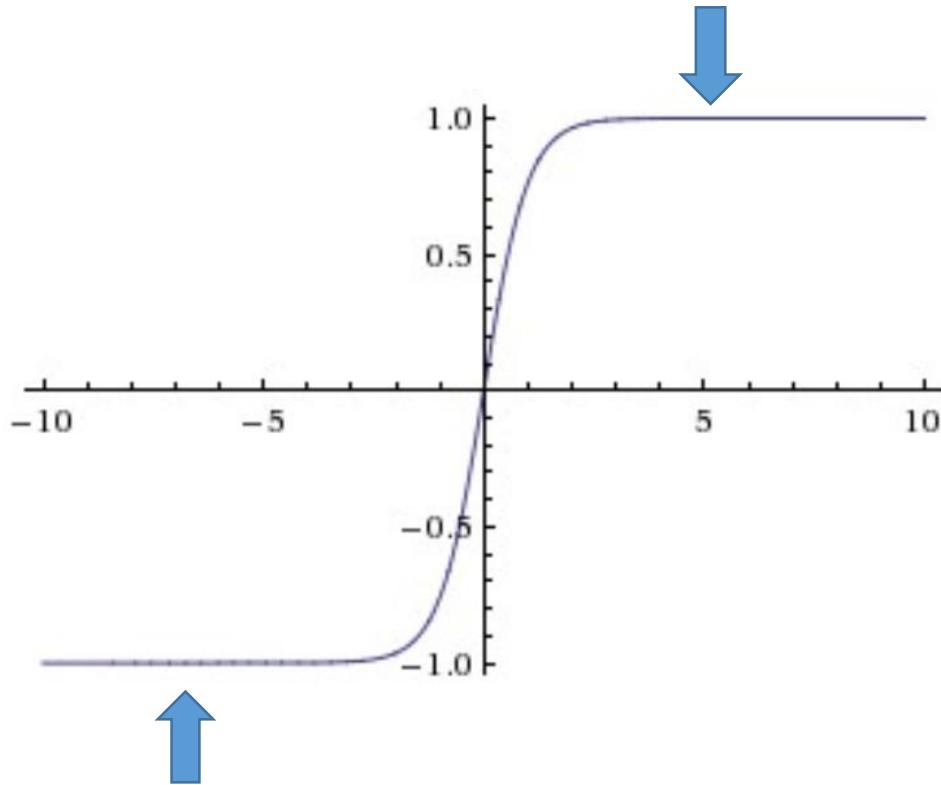


Tanh (hyperbolic tangent) 활성화 함수의 장점



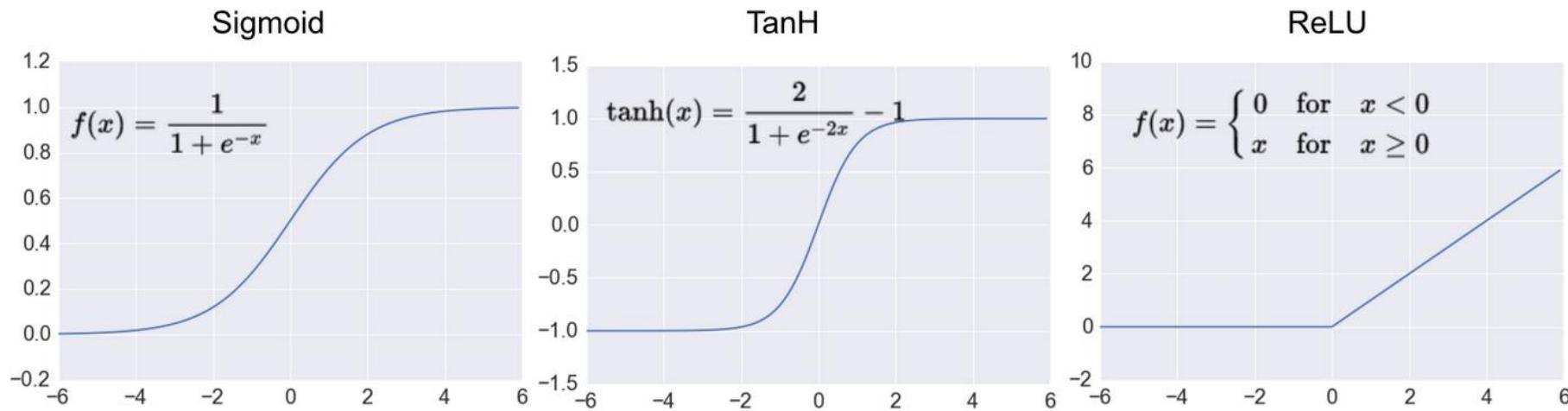
- Tanh 함수는 미분해도 경사가 감소하지 않는다
- Zero – centric 하여 학습이 더 안정적으로 수행 될 수 있도록 해준다

Tanh (hyperbolic tangent) 활성화 함수의 단점



- 매우 크거나 매우 작은 값에 대하여 경사가 매우 작아진다

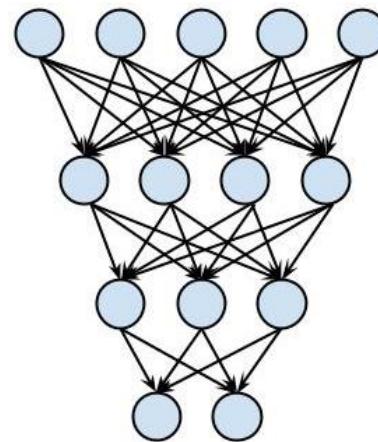
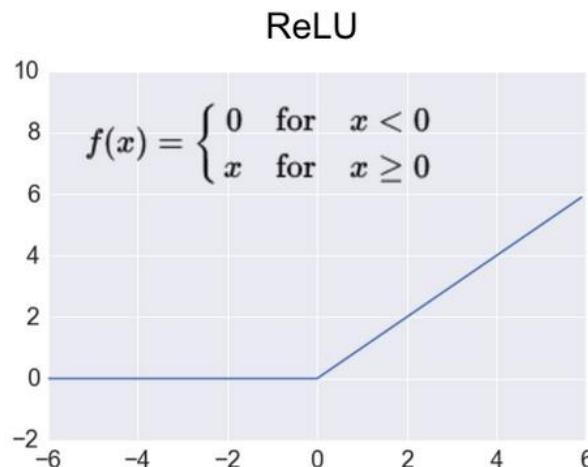
ReLU 활성화 함수



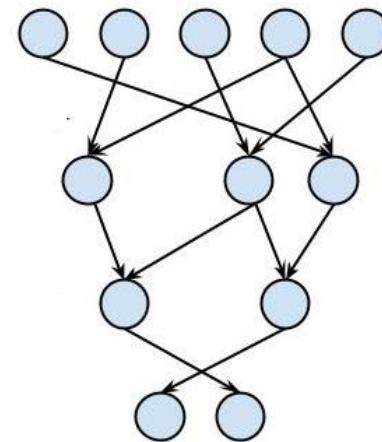
- 비선형 함수이다
- 계단함수로 사용 가능하다
- 미분이 간단하다
- 미분해도 경사 감소 문제가 발생하지 않는다
- 입력 값이 매우 커져도 경사 감소 문제가 발생하지 않는다

ReLU 활성화 함수 - Sparsity

- 단, 0 이하 값에 대하여는 학습이 진행되지 않는다 [단점]
Zero – centric 하지 않다 [단점]
- 이는 신경망의 Sparsity를 높인다 [장점]



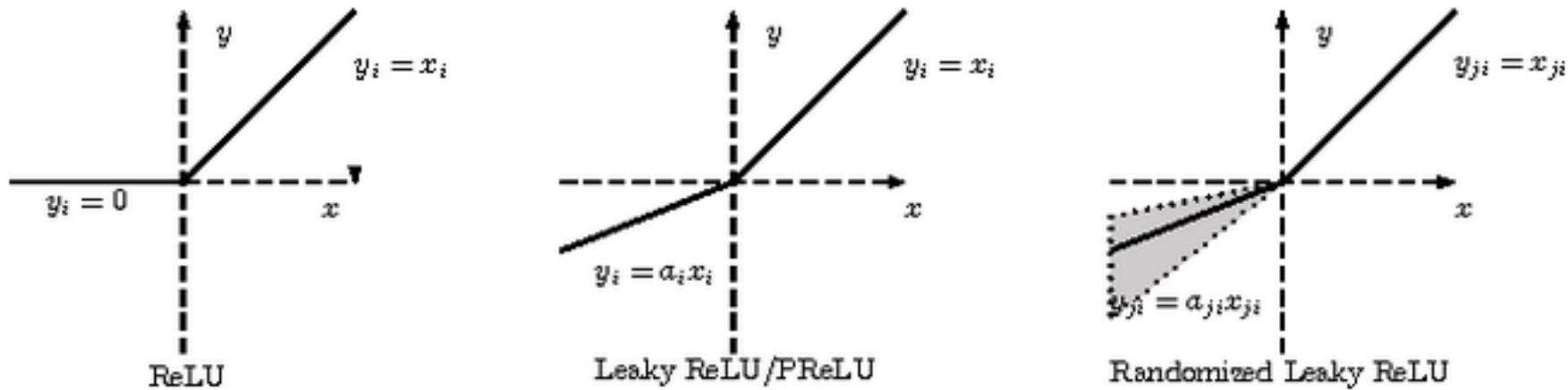
(a) Dense network



(b) Sparse network

- 대부분 파라미터가 0이 되어 신경망의 overfitting 문제가 완화 된다

Leaky ReLU



ReLU 함수의 장점을 살리고 단점을 보완하는 활성화 함수들이 개발 됨

ReLU 함수 구현 및 결과

- ReLU 함수 라이브러리 사용

```
L1 = tf.sigmoid(tf.matmul(X, W1) + b1)
```



```
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
```

```
#=====
# 3. 모델 정의
#=====

x_image = x

def conv(X, in_ch, out_ch, name):
    with tf.variable_scope(name) as scope:
        W_conv = tf.Variable(tf.truncated_normal(shape=[3, 3, in_ch, out_ch], stddev=0.01))
        b_conv = tf.Variable(tf.constant(0.1, shape=[out_ch]))
        h_conv = tf.nn.relu(tf.nn.conv2d(X, W_conv, strides=[1, 1, 1, 1], padding='SAME') + b_conv)
    return h_conv
```

ReLU 함수 사용 구현 및 사용 결과

```
반복(Epoch): 0 트레이닝 데이터 정확도: 0.11991186 손실 함수(loss): 2.2979372
테스트 데이터 정확도: 0.099859774
```

```
반복(Epoch): 1 트레이닝 데이터 정확도: 0.11634615 손실 함수(loss): 2.299668
테스트 데이터 정확도: 0.099859774
```

```
반복(Epoch): 2 트레이닝 데이터 정확도: 0.11394231 손실 함수(loss): 2.3004184
테스트 데이터 정확도: 0.099859774
```

여전히 학습이 진행되지 않는다

- 목차

- 1. Cifar10 Image Classification Dataset 소개

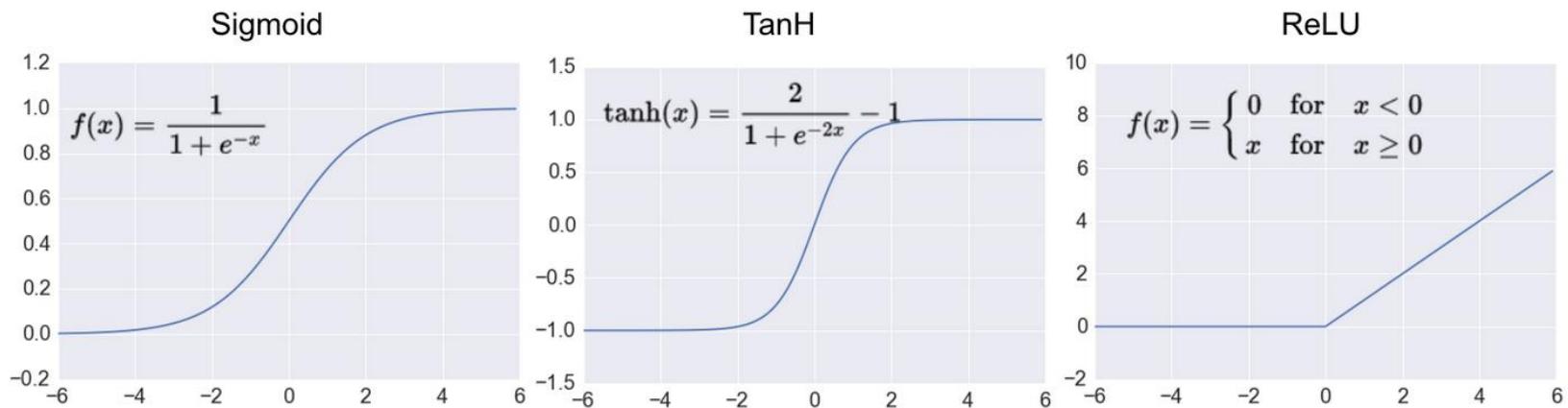
- 2. CNN 성능 향상을 위한 기법들

- ReLU
 - Initializer 
 - Optimizer
 - Batch Normalization
 - Dropout
 - Regularization

(Advanced)

- Data Augmentation
 - Ensemble

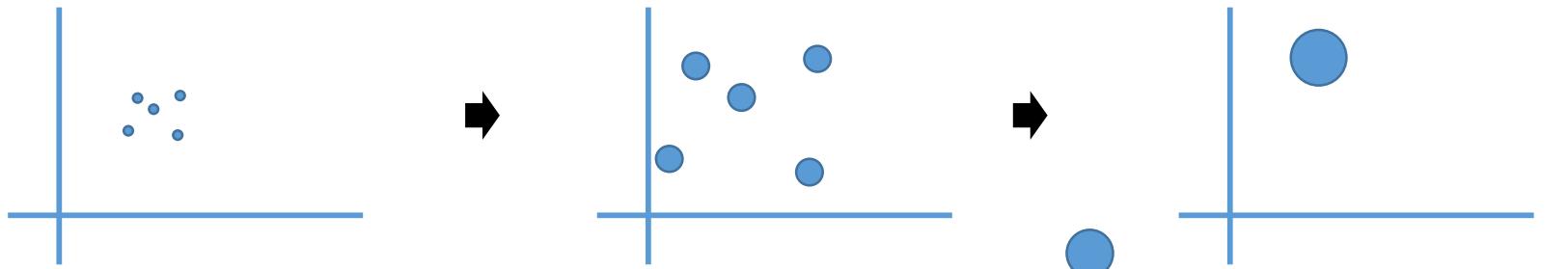
가중치 초기화의 중요성



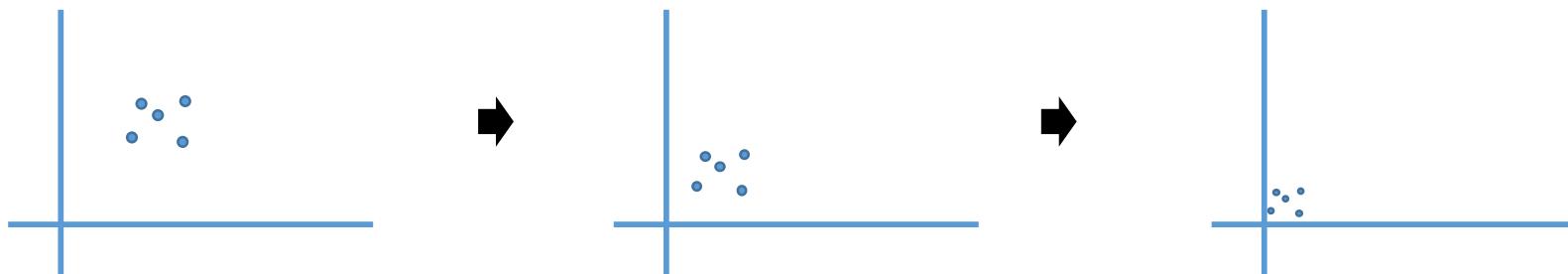
- 가중치가 너무 작거나 크면 Gradient는 0에 가까운 값을 가지게 되며 이 역시 경사 감소 소멸 현상의 원인이 된다.
- 따라서 활성화 함수를 개선하는 것 뿐만 아니라 가중치를 적절하게 초기화 하는 것을 통해서도 경사 감소 소멸 문제를 완화할 수 있다.

가중치 초기화의 중요성

- 가중치가 큰 경우 값이 과하게 분산될 수 있다



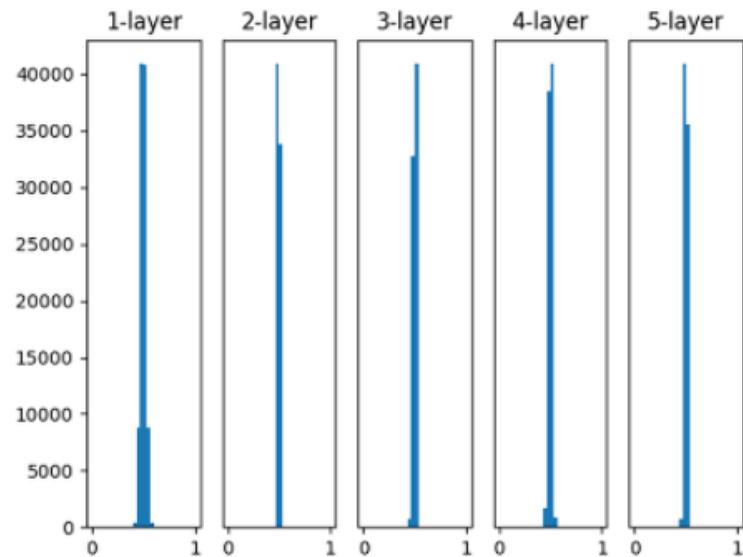
- 가중치가 작은 경우 값이 지나치게 0에 수렴할 수 있다



가중치 초기화의 방법

- 2. 작은 표준편차를 사용하여 초기화 – 평균0, 표준편차가 0.01 (기존 방법)

```
32     def conv(X, in_ch, out_ch, name):  
33         with tf.variable_scope(name) as scope:  
34             W_conv = tf.Variable(tf.truncated_normal(shape=[3, 3, in_ch, out_ch], stddev=0.01))  
35             b_conv = tf.Variable(tf.constant(0.1, shape=[out_ch]))  
36             h_conv = tf.nn.sigmoid(tf.nn.conv2d(X, W_conv, strides=[1, 1, 1, 1], padding='SAME') + b_conv)  
37         return h_conv
```



가중치 초기화의 방법

- 3. 발전된 방법 : Xavier Initialization

$$w = \text{np.random.randn}(n_input, n_output) / \text{sqrt}(n_input)$$

=> 단순히 작은 값의 표준편차를 갖는 형태로 초기화 하는 것이 아닌, 표준 정규 분포를 입력개수의 표준편차로 나누어 주는 방법을 사용

```
32 def conv(X, in_ch, out_ch, name):  
33     with tf.variable_scope(name) as scope:  
34         W_conv = tf.Variable(tf.truncated_normal(shape=[3, 3, in_ch, out_ch], stddev=0.01))  
35         b_conv = tf.Variable(tf.constant(0.1, shape=[out_ch]))  
36         h_conv = tf.nn.sigmoid(tf.nn.conv2d(X, W_conv, strides=[1, 1, 1, 1], padding='SAME') + b_conv)  
37     return h_conv
```



```
3 from tensorflow.contrib.layers import xavier_initializer_conv2d  
  
34 def conv(X, in_ch, out_ch, name):  
35     with tf.variable_scope(name) as scope:  
36         W_conv = tf.get_variable(name='weights', shape=[3, 3, in_ch, out_ch], initializer=xavier_initializer_conv2d())  
37         b_conv = tf.Variable(tf.constant(0.1, shape=[out_ch]))  
38         h_conv = tf.nn.relu(tf.nn.conv2d(X, W_conv, strides=[1, 1, 1, 1], padding='SAME') + b_conv)  
39     return h_conv
```

- 목차

1. Cifar10 Image Classification Dataset 소개

2. CNN 성능 향상을 위한 기법들

- ReLU
- Initializer
- Optimizer 
- Batch Normalization
- Dropout
- Regularization

(Advanced)

- Data Augmentation
- Ensemble

AdamOptimizer 사용

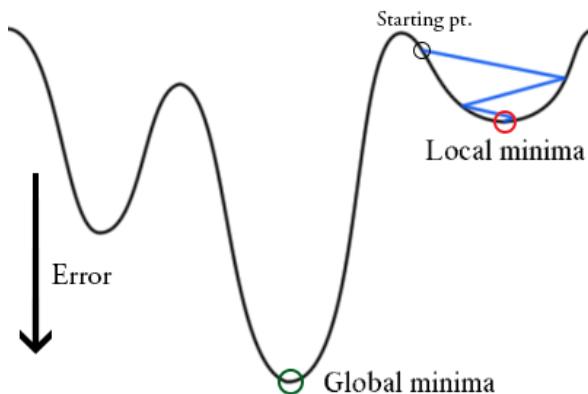
```
51 =====  
52 # 4. 비용함수 정의  
53 =====  
54     loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=logits))  
55     train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)
```



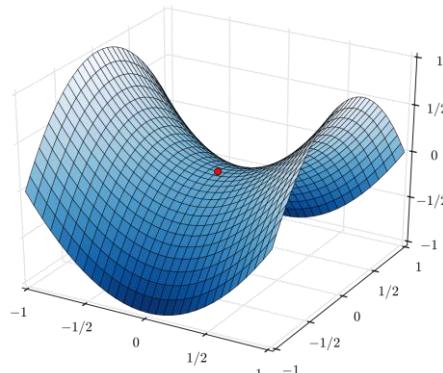
```
71 =====  
72 # 4. 비용함수 정의  
73 =====  
74     loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y, logits=logits))  
75     train_step = tf.train.AdamOptimizer(0.0001).minimize(loss)
```

Local minima

- Local minima에 갇혀서 학습이 진행되지 않을 수 있다

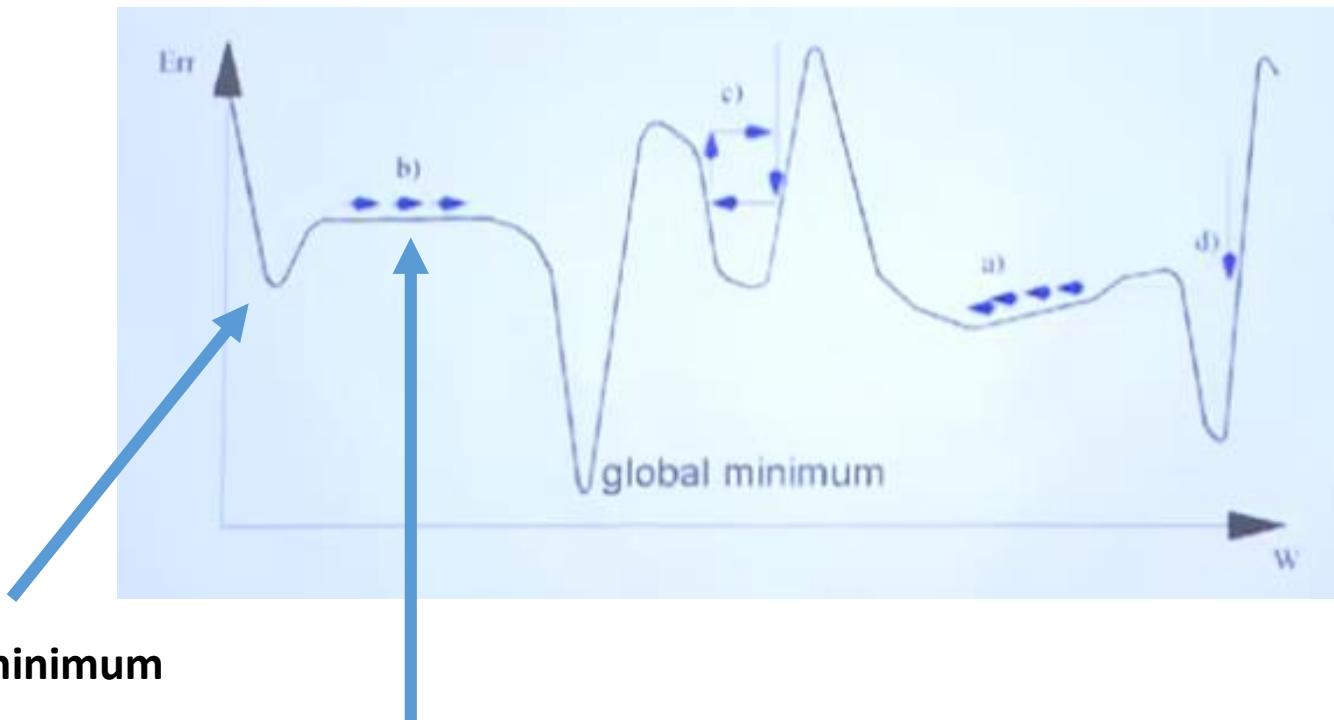


- 그러나 최근 연구에 따르면 이러한 Local minima는 차원수가 증가함에 따라 기하급수적으로 등장확률이 희박해지며 대부분은 Saddle point 이기 때문에 크게 문제가 되지 않는다

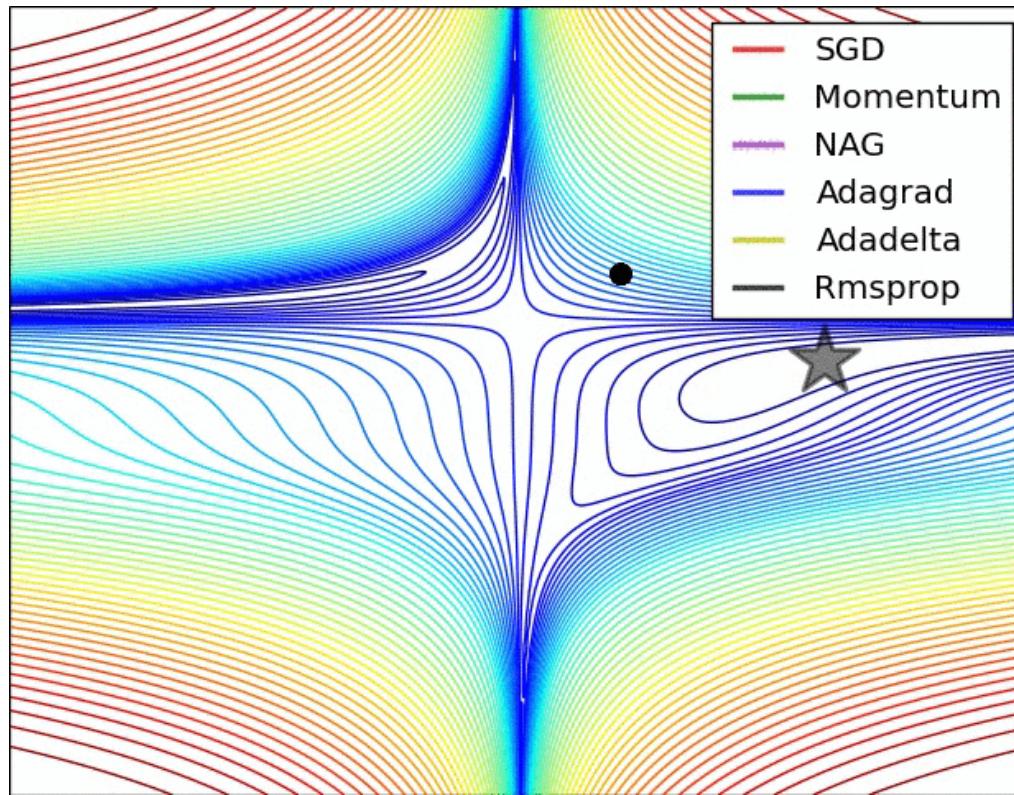


오차 곡면의 평평한 구간 문제

- 실제로는 경사가 작은 평평한 구간이 더욱 큰 문제로 작용



Momentum (관성)



진행 방향에 따른 Momentum (관성) 을 주어 경사가 급감해도 학습이 진행될 수 있도록 한다

AdamOptimizer 사용

```
51 =====  
52 # 4. 비용함수 정의  
53 =====  
54     loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=logits))  
55     train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)
```



```
71 =====  
72 # 4. 비용함수 정의  
73 =====  
74     loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y, logits=logits))  
75     train_step = tf.train.AdamOptimizer(0.0001).minimize(loss)
```

ReLU + Xavier initializer + AdamOptimizer 성능 평가

```
반복(Epoch): 202 트레이닝 데이터 정확도: 1.0 손실 합수(loss): 0.000331494
테스트 데이터 정확도: 0.78725964
```

```
반복(Epoch): 203 트레이닝 데이터 정확도: 1.0 손실 합수(loss): 0.0003098271
테스트 데이터 정확도: 0.79086536
```

```
반복(Epoch): 204 트레이닝 데이터 정확도: 1.0 손실 합수(loss): 0.00039825856
테스트 데이터 정확도: 0.7800481
```

```
반복(Epoch): 205 트레이닝 데이터 정확도: 1.0 손실 합수(loss): 0.00035657172
테스트 데이터 정확도: 0.78675884
```

```
반복(Epoch): 206 트레이닝 데이터 정확도: 0.99998 손실 합수(loss): 0.00048114886
테스트 데이터 정확도: 0.7807492
```

- 목차

- 1. Cifar10 Image Classification Dataset 소개

- 2. CNN 성능 향상을 위한 기법들

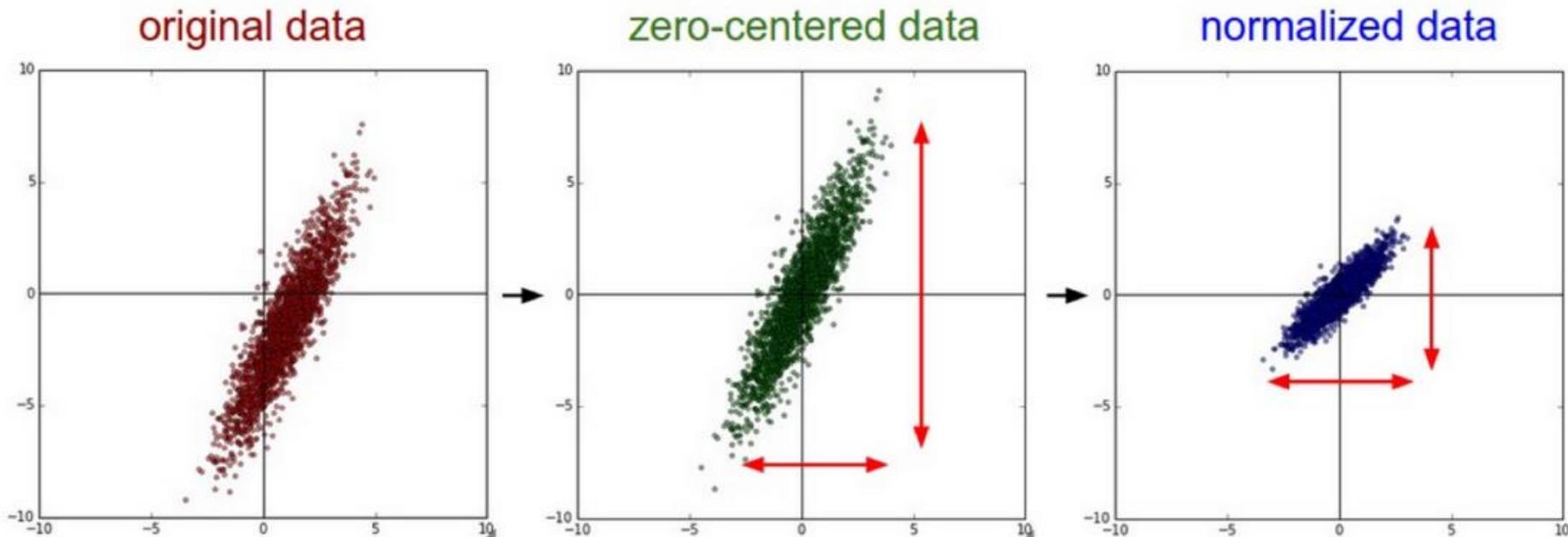
- ReLU
 - Initializer
 - Optimizer
 - Batch Normalization 
 - Dropout
 - Regularization

(Advanced)

- Data Augmentation
 - Ensemble

Dataset Normalization (Input normalization) 이란

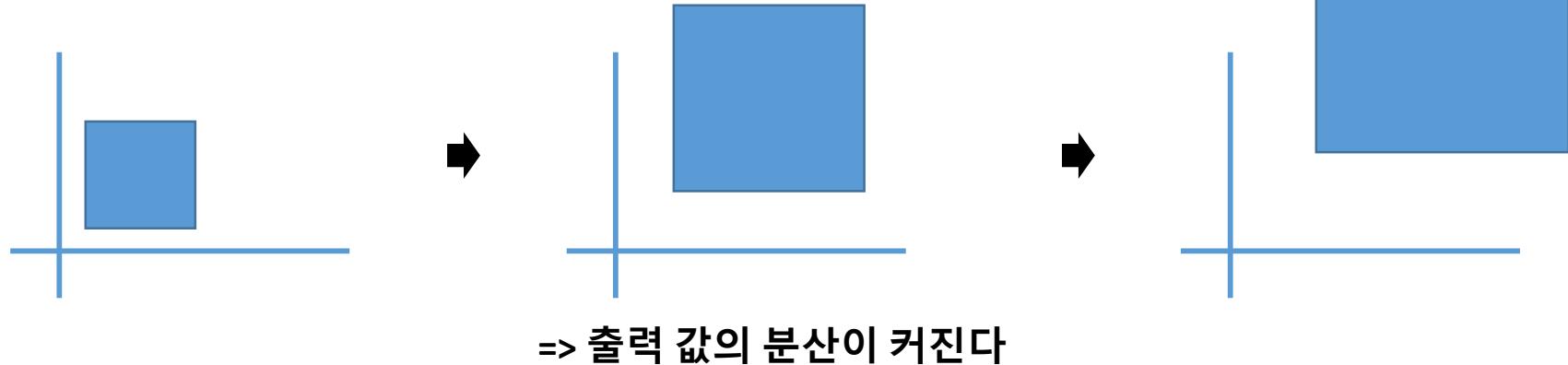
- 입력 값의 표준화를 통해서도 출력 값의 분산이 커지는 현상을 완화할 수 있다



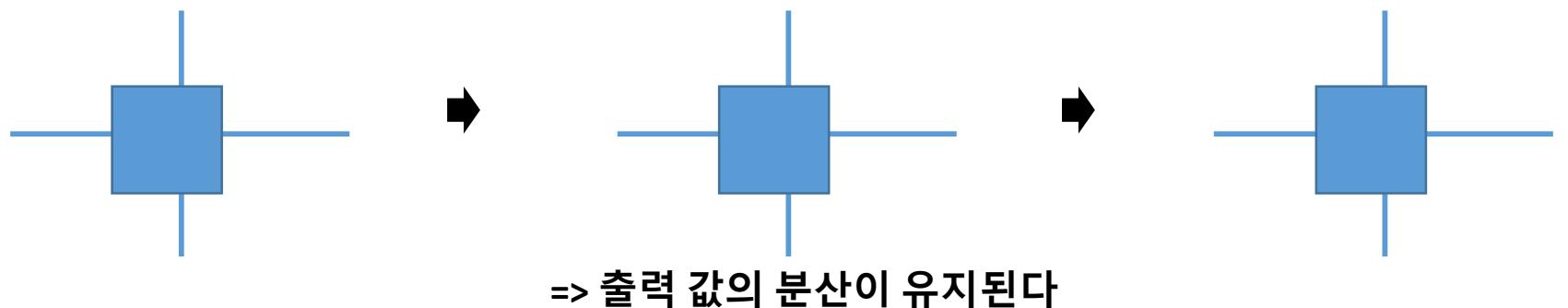
값의 범위가 0~255 사이였던 기존 입력 값의 분포를 -1~1 사이로 표준화

Dataset Normalization의 중요성

- 표준화가 되어있지 않은 경우



- 표준화가 잘 되어있는 경우



Dataset Normalization 구현

```
7      =====
8  # 1. CIFAR-10 데이터 다운로드 및 데이터 로드
9  =====
10     (x_train, y_train), (x_test, y_test) = load_data()
11
12     #input normalization
13     x_train_zerocentered = x_train - np.mean(x_train) #평균이 0에 가깝도록 한다
14     x_train = x_train_zerocentered/np.max(x_train_zerocentered) #값의 범위 -1 ~ 1이 되도록 한다
15
16     print(np.mean(x_train))
17     print(x_train[0])
18
19     x_test_zerocentered = x_test - np.mean(x_test)
20     x_test = x_test_zerocentered/ np.max(x_test_zerocentered)
21
```

Dataset Normalization 구현

```
x_train.shape : (50000, 32, 32, 3)
y_train.shape : (50000, 1)
[[[ 59  62  63]
 [ 43  46  45]
 [ 50  48  43]
 ...
 [158 132 108]
 [152 125 102]
 [148 124 103]]]

[[ 16  20  20]
 [  0   0   0]
 [ 18   8   0]
 ...
 ...]
```

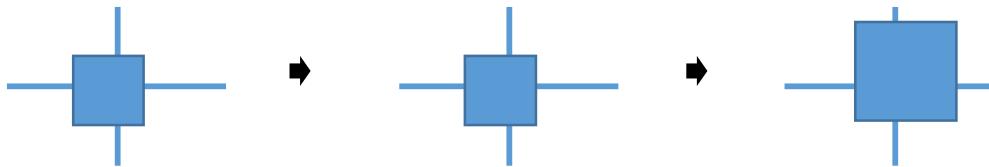
(a) 기준 : 0~255 사이의 값

```
mean : -1.5143442055887134e-17
[[[-0.45950142 -0.43716212 -0.42971568]
 [-0.57864444 -0.55630509 -0.56375153]
 [-0.52651935 -0.54141222 -0.57864444]
 ...
 [ 0.27769572  0.08408839 -0.09462607]
 [ 0.23301711  0.03196334 -0.13930468]
 [ 0.20323137  0.02451691 -0.13185825]]
```

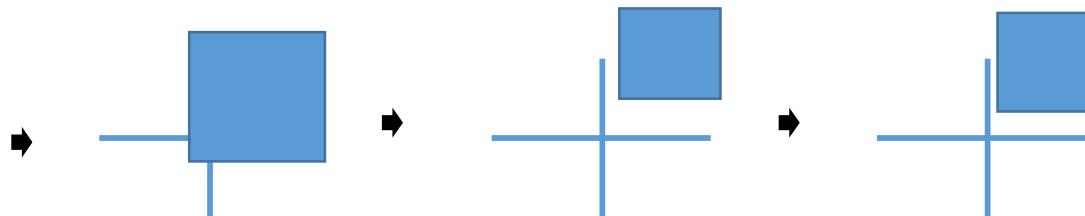
(b) 표준화 : -1~1 사이의 값

Batch Normalization 이란

- 표준화가 잘 되어있어도 레이어를 거치면서 어느정도는 흩어 질 수 있다.



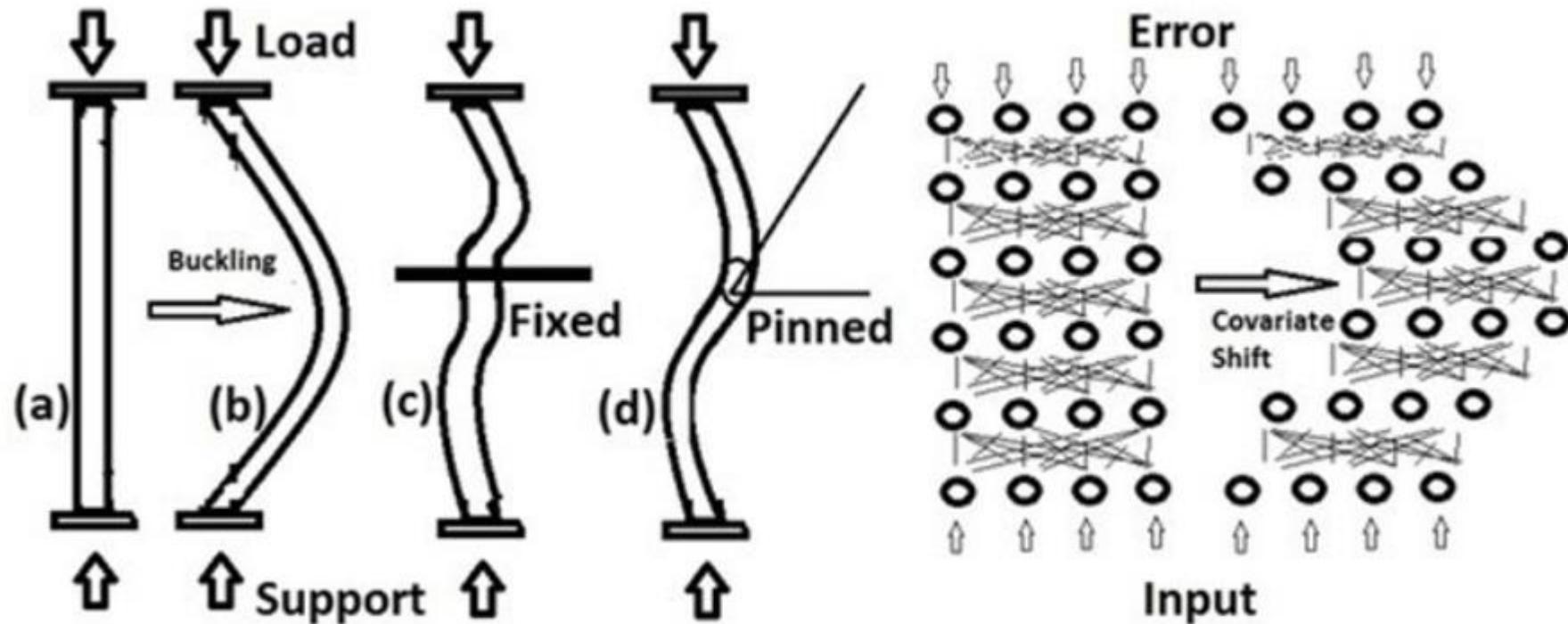
=> 처음에는 어느정도 유지될 수 있지만



=> 흩어짐이 누적되면서 전체 결과에 영향을 줄 수 있다

Batch Normalization 이란

- 해결책 : 입력 데이터 뿐만 아니라 배치단위로 매 레이어에서 표준화를 한다



=> 매 레이어마다 부분적으로 보정을 수행하여 전체적으로 왜곡이 일어나는 것을 방지

Batch Normalization 을 사용한 학습

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

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

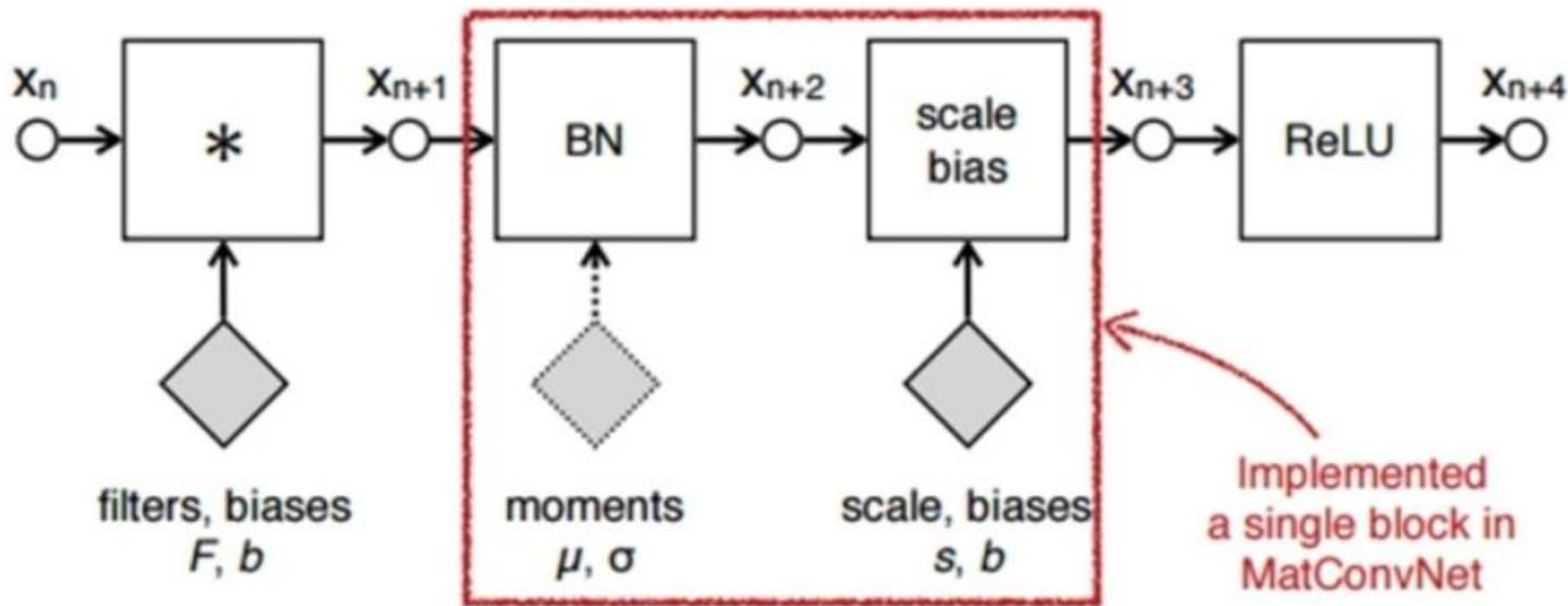
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

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

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

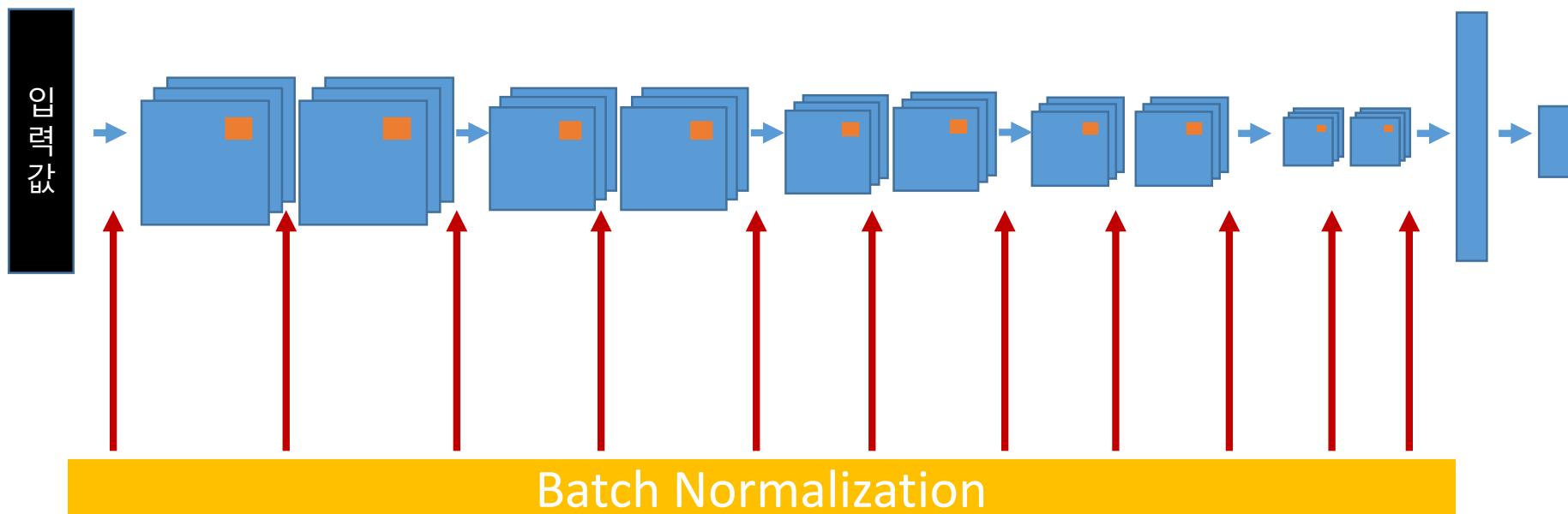
CNN 성능 향상 기법 : Batch Normalization

Batch Normalization 을 사용한 학습

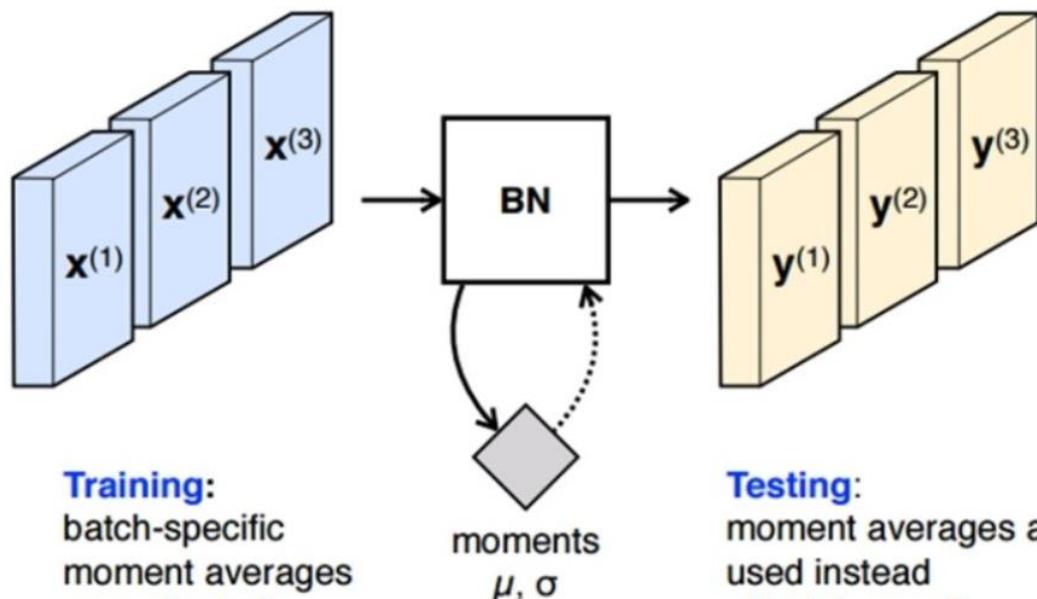


CNN 성능 향상 기법 : Batch Normalization

Batch Normalization 을 사용한 학습



Batch Normalization 을 사용한 테스트



- 학습시에는 미니 배치 별로 모멘텀을 저장
- 평가시에는 저장된 모멘텀의 평균을 사용

Batch Normalization 구현

```
21 =====
22 # 2. 인풋, 아웃풋데이터를 입력받기 위한 플레이스홀더 정의
23 =====
24     x = tf.placeholder(tf.float32, shape=[None, 32, 32, 3])
25     y = tf.placeholder(tf.float32, shape=[None, 10])
26
27     #batch norm 을위한 train pahse 정의
28     trainphase = tf.placeholder(tf.bool)
29
30 =====
31 # 3. 모델 정의
32 =====
33     x_image = x
34
35     def conv(X, in_ch, out_ch, name):
36         with tf.variable_scope(name) as scope:
37             W_conv = tf.get_variable(name='weights', shape=[3, 3, in_ch, out_ch], initializer=xavier_initializer_conv2d())
38             h_bn = tf.layers.batch_normalization(tf.nn.conv2d(X, W_conv, strides=[1, 1, 1, 1], padding='SAME'), training =trainphase)
39             h_conv = tf.nn.relu(h_bn)
40         return h_conv
```

Batch Normalization 구현

```
74 =====
75 # 4. 비용함수 정의
76 =====
77     loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=logits))
78     update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
79     with tf.control_dependencies(update_ops):
80         train_step = tf.train.AdamOptimizer(0.0001).minimize(loss)
81
82     # 정확도를 계산하는 연산.
83     correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y, 1))
84     accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
85
```

Batch Normalization 구현

```
97      #.....
98      # 5.2.1 학습
99      #.....
100     total_size = x_train.shape[0]
101     batch_size = 128
102
103     loss_list = []
104     train_accuracy_list = []
105     for i in range(int(total_size / batch_size)):
106         #== batch load
107         batch_x = x_train[i*batch_size:(i+1)*batch_size]
108         batch_y = y_train_onehot[i*batch_size:(i+1)*batch_size]
109
110         #== train
111         sess.run(train_step, feed_dict={x: batch_x, y: batch_y, trainphase : True})
112
113         #== logging
114         train_accuracy = accuracy.eval(feed_dict={x: batch_x, y: batch_y, trainphase : False})
115         loss_print = loss.eval(feed_dict={x: batch_x, y: batch_y, trainphase : False})
116         train_accuracy_list.append(train_accuracy)
117         loss_list.append(loss_print)
118
119         print("반복(Epoch):", e, "트레이닝 데이터 정확도:", np.mean(train_accuracy_list), "손실 함수(loss):",np.mean(loss_list))
```

- 목차

1. Cifar10 Image Classification Dataset 소개

2. CNN 성능 향상을 위한 기법들

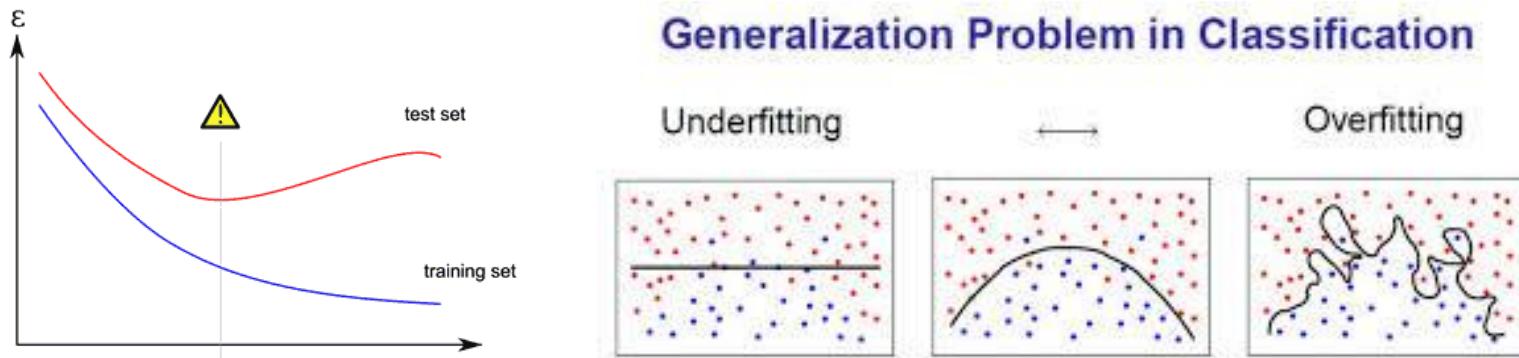
- ReLU
- Initializer
- Optimizer
- Batch Normalization
- Dropout 
- Regularization

(Advanced)

- Data Augmentation
- Ensemble

Overfitting 문제

- Training set의 노이즈에 까지 overfit 되어 test set에 대한 정확도가 낮아지는 문제



망이 깊어질 수록 (파라미터가 많아 질 수록) 심해진다

Overfitting 문제

- Overfitting 문제 해결을 위한 3가지 핵심 기법

1. 학습 데이터를 많이 확보한다

=> Data augmentation

2. feature의 수를 줄이거나 여러 feature를 같이 사용한다

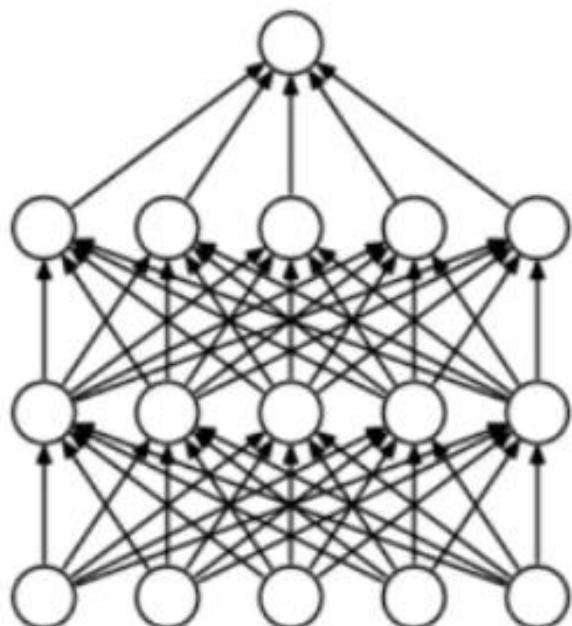
=> Dropout, Ensemble

3. 파라미터의 수를 줄인다

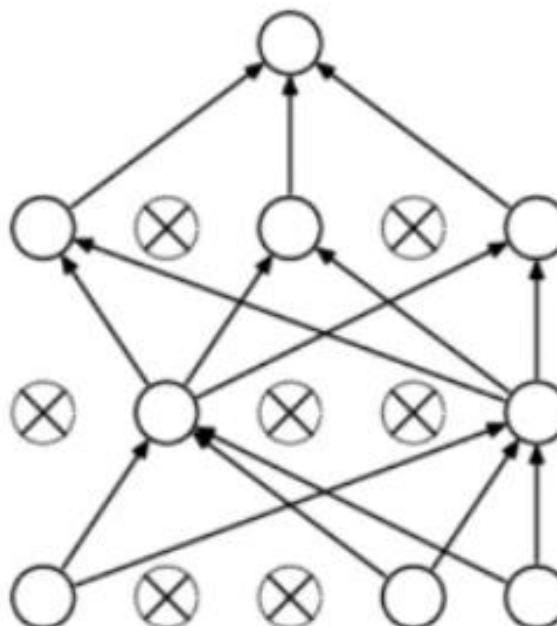
=> Regularization

Dropout

- 임의의 확률로 일부 neuron을 학습에서 제외한다 (가중치 값을 0으로 만든다)



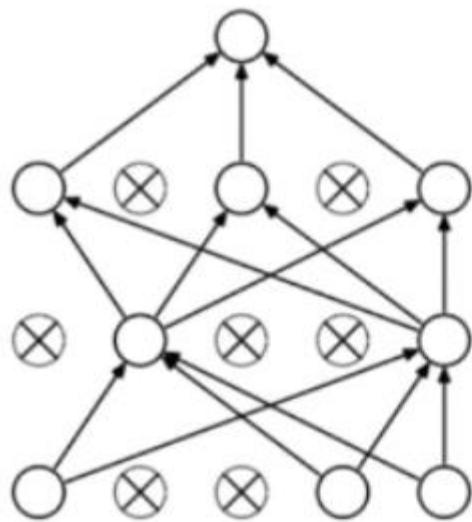
(a) Standard Neural Net



(b) After applying dropout.

Dropout의 학습과 평가

- 학습시에는 이를 통하여 중복된 feature를 줄이며, overfitting 문제를 완화한다



Forces the network to have a redundant representation.



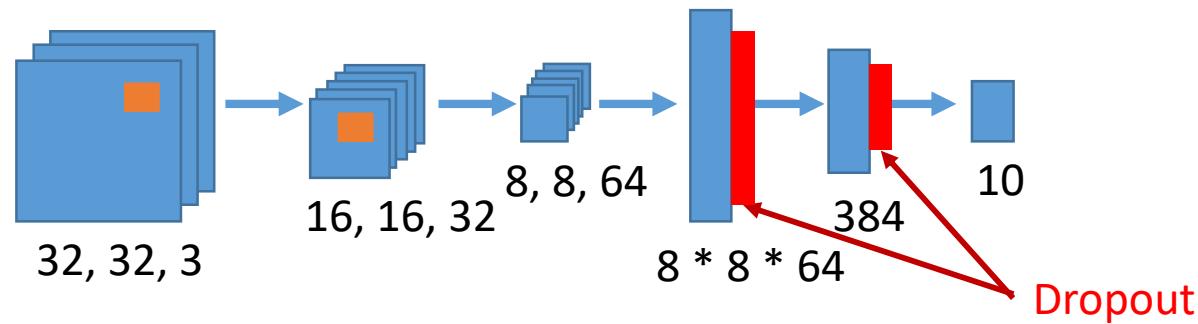
- 테스트 시에는 이렇게 학습된 모든 neuron들을 다 사용한다

Dropout 구현

- Tensorflow 라이브러리 사용

```
66     h_conv10_pool_flat_dropout = tf.nn.dropout(h_conv10_pool_flat , keep_prob=keepprob)
```

- 일반적으로 CNN의 경우 Fully connected layer에서 주로 사용된다



- 하지만 Batch normalization 0|Dropout의 역할도 수행하기 때문에 둘 중 하나만 사용

Dropout 구현

- Keepprob placeholder 정의

```
21 =====
22 # 2. 인풋, 아웃풋데이터를 입력받기 위한 플레이스홀더 정의
23 =====
24     x = tf.placeholder(tf.float32, shape=[None, 32, 32, 3])
25     y = tf.placeholder(tf.float32, shape=[None, 10])
26
27     #batch norm 을위한 train pahse 정의
28     trainphase = tf.placeholder(tf.bool)
29     #dropout 을 위한 keepprob 정의
30     keepprob = tf.placeholder(tf.float32)
```

- 모델 정의

```
65     h_conv10_pool_flat = tf.reshape(h_conv10_pool, [-1, 1 * 1 * 512])
66     h_conv10_pool_flat_dropout = tf.nn.dropout(h_conv10_pool_flat , keep_prob=keepprob)
67     W_fc = tf.get_variable(name='weights', shape=[512, 10], initializer=xavier_initializer())
68     b_fc = tf.Variable(tf.constant(0.1, shape=[10]))
69     logits = tf.matmul(h_conv10_pool_flat, W_fc) + b_fc
70     y_pred = tf.nn.softmax(logits)
```

Dropout 구현

- 학습

```
100      #.....
101      # 5.2.1 학습
102      #.....
103      total_size = x_train.shape[0]
104      batch_size = 128
105
106      loss_list = []
107      train_accuracy_list = []
108      for i in range(int(total_size / batch_size)):
109          #== batch load
110          batch_x = x_train[i*batch_size:(i+1)*batch_size]
111          batch_y = y_train_onehot[i*batch_size:(i+1)*batch_size]
112
113          #== train
114          sess.run(train_step, feed_dict={x: batch_x, y: batch_y, trainphase : True , keepprob:0.7})
115
116          #== logging
117          train_accuracy = accuracy.eval(feed_dict={x: batch_x, y: batch_y, trainphase : False , keepprob:1})
118          loss_print = loss.eval(feed_dict={x: batch_x, y: batch_y, trainphase : False , keepprob:1})
119          train_accuracy_list.append(train_accuracy)
120          loss_list.append(loss_print)
121
122          print("반복(Epoch):", e, "트레이닝 데이터 정확도:", np.mean(train_accuracy_list), "손실 함수(loss):",np.mean(loss_list))
```

반복(Epoch): 251 트레이닝 데이터 정확도: 1.0 손실 함수(loss): 0.0643254
테스트 데이터 정확도: 0.8421474

반복(Epoch): 252 트레이닝 데이터 정확도: 1.0 손실 함수(loss): 0.06303058
테스트 데이터 정확도: 0.8427484

반복(Epoch): 253 트레이닝 데이터 정확도: 1.0 손실 함수(loss): 0.06169955
테스트 데이터 정확도: 0.8420473

반복(Epoch): 254 트레이닝 데이터 정확도: 1.0 손실 함수(loss): 0.0602972
테스트 데이터 정확도: 0.8415465

반복(Epoch): 255 트레이닝 데이터 정확도: 1.0 손실 함수(loss): 0.05878525
테스트 데이터 정확도: 0.8407452

- 목차

- 1. Cifar10 Image Classification Dataset 소개

- 2. CNN 성능 향상을 위한 기법들

- ReLU
 - Initializer
 - Optimizer
 - Batch Normalization
 - Dropout
 - Regularization 

(Advanced)

- Data Augmentation
 - Ensemble

Regularization

- Loss 에 Weight들의 합을 Regularizer 로 더해주면 전체 weight들이 과도하게 커지는 것을 완화 할 수 있다

- L2-regularization (weight decay): regularization parameter

$$\mathcal{L}_{new} = \mathcal{L} + \frac{\lambda}{2} W^2$$

- L1-regularization:

$$\mathcal{L}_{new} = \mathcal{L} + \frac{\lambda}{2} |W|$$

Lambda 는 일반적으로 0.0005와 같은 매우 작은 값을 사용

Regularization L1 vs L2

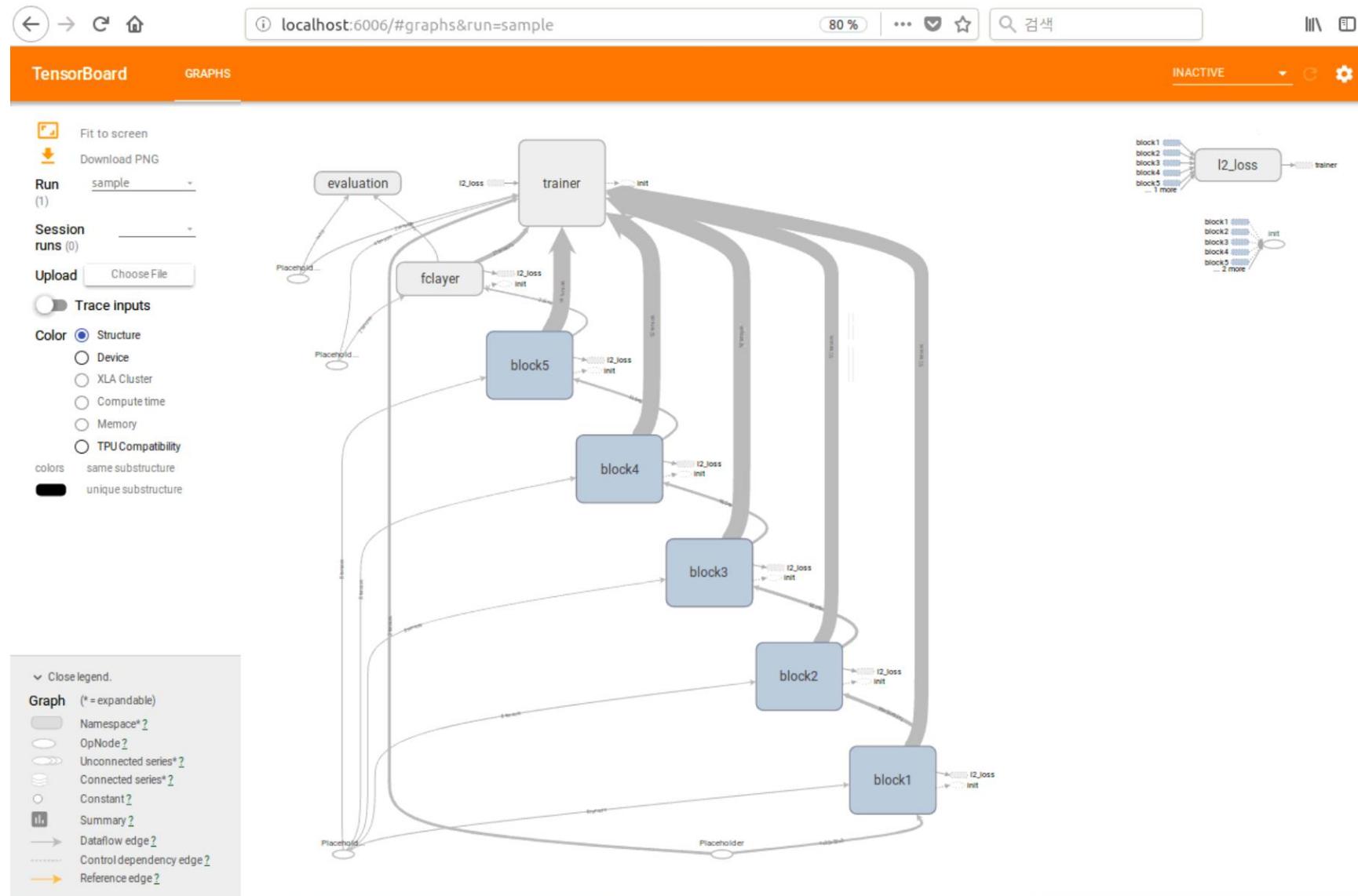
L2 loss function	L1 loss function
Not very robust	Robust
Stable solution	Unstable solution
Always one solution	Possibly multiple solutions

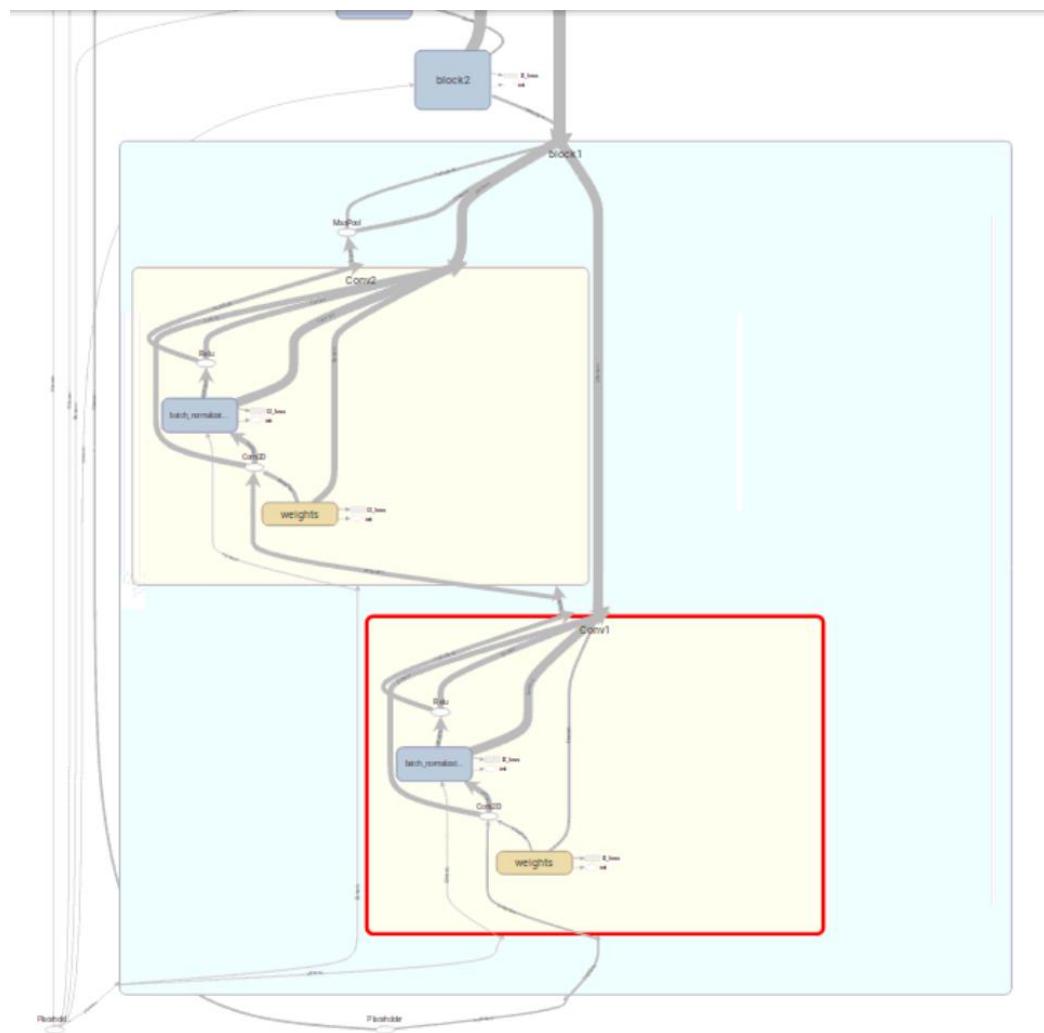
일반적인 경우 L2 Loss를 사용한다

Regularization 구현

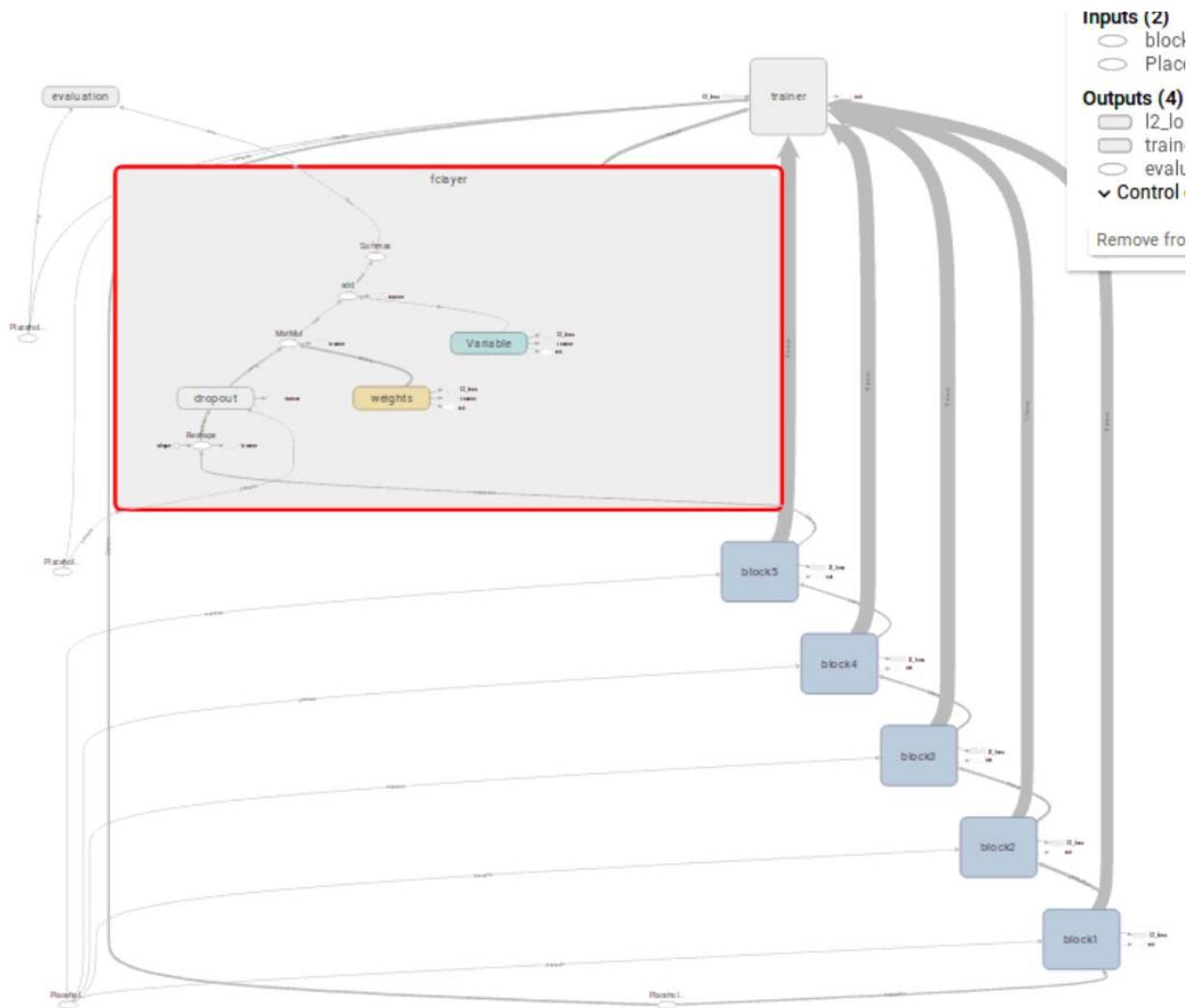
```
71  =====
72  # 4. 비용함수 정의
73  =====
74      vars    = tf.trainable_variables()
75  lossL2 = tf.add_n([ tf.nn.l2_loss(v) for v in vars ]) * 0.0005
76
77  loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y, logits=logits)) + lossL2
78  train_step = tf.train.AdamOptimizer(0.0001).minimize(loss)
79
80  # 정확도를 계산하는 연산.
81  correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y, 1))
82  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
83
```

최종 구현 및 모델 성능 평가





최종 구현 및 모델 성능 평가



*youtube : <https://www.youtube.com/watch?v=5vPsJw4UiQk>

```
1 import tensorflow as tf
2 import numpy as np
3 from keras.datasets.cifar10 import load_data
4
5
6 if __name__ == "__main__":
7 #=====
8 # 1. CIFAR-10 데이터 다운로드 및 데이터 로드
9 #=====
10 (x_train, y_train), (x_test, y_test) = load_data()
11
12 #one hot encoding
13 y_train_onehot = np.eye(10)[y_train]
14 y_test_onehot = np.eye(10)[y_test]
15
16 y_train_onehot = np.squeeze(y_train_onehot) # (50000,1,10) -> (50000,10)
17 y_test_onehot = np.squeeze(y_test_onehot) # (10000,1,10) -> (10000,10)
18
19 #=====
20 # 2. 인풋, 레이블을 입력받기 위한 플레이스홀더 정의
21 #=====
22 x = tf.placeholder(tf.float32, shape=[None, 32, 32, 3])
23 y = tf.placeholder(tf.float32, shape=[None, 10])
24
```

최종 구현 및 모델 성능 평가

```
32 #=====
33 # 3. 모델 정의
34 #=====
35     x_image = x
36
37     def conv(X, in_ch, out_ch, name):
38         with tf.variable_scope(name) as scope:
39             W_conv = tf.get_variable(name='weights', shape=[3, 3, in_ch, out_ch], initializer=xavier_initializer_conv2d())
40             h_bn = tf.layers.batch_normalization(tf.nn.conv2d(X, W_conv, strides=[1, 1, 1, 1], padding='SAME'), training =trainphase)
41             h_conv = tf.nn.relu(h_bn)
42             return h_conv
43
44     with tf.variable_scope("block1") as scope:
45         h_conv1 = conv(x_image,3,64,"Conv1")
46         h_conv2 = conv(h_conv1,64,64,"Conv2")
47         h_conv2_pool = tf.nn.max_pool(h_conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
48
49     with tf.variable_scope("block2") as scope:
50         h_conv3 = conv(h_conv2_pool,64,128,"Conv3")
51         h_conv4 = conv(h_conv3, 128,128 , "Conv4")
52         h_conv4_pool = tf.nn.max_pool(h_conv4, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
53
```

최종 구현 및 모델 성능 평가

```
53
54     with tf.variable_scope("block3") as scope:
55         h_conv5 = conv(h_conv4_pool, 128, 256, "Conv5")
56         h_conv6 = conv(h_conv5, 256, 256, "Conv6")
57         h_conv6_pool = tf.nn.max_pool(h_conv6, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
58
59     with tf.variable_scope("block4") as scope:
60         h_conv7 = conv(h_conv6_pool, 256, 512, "Conv7")
61         h_conv8 = conv(h_conv7, 512, 512, "Conv8")
62         h_conv8_pool = tf.nn.max_pool(h_conv8, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
63
64     with tf.variable_scope("block5") as scope:
65         h_conv9 = conv(h_conv8_pool, 512, 512, "Conv9")
66         h_conv10 = conv(h_conv9, 512, 512, "Conv10")
67         h_conv10_pool = tf.nn.max_pool(h_conv10, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
68
69     with tf.variable_scope("fclayer") as scope:
70         h_conv10_pool_flat = tf.reshape(h_conv10_pool, [-1, 1 * 1 * 512])
71         h_conv10_pool_flat_dropout = tf.nn.dropout(h_conv10_pool_flat, keep_prob=keepprob)
72         W_fc = tf.get_variable(name='weights', shape=[512, 10], initializer=xavier_initializer())
73         b_fc = tf.Variable(tf.constant(0.1, shape=[10]))
74         logits = tf.matmul(h_conv10_pool_flat_dropout, W_fc) + b_fc
75         y_pred = tf.nn.softmax(logits)
```

최종 구현 및 모델 성능 평가

```
79  =====
80 # 4. 비용함수 정의
81 =====
82     with tf.name_scope("l2_loss") as scope :
83         vars = tf.trainable_variables()
84         lossL2 = tf.add_n([tf.nn.l2_loss(v) for v in vars]) * 0.0005
85
86     with tf.name_scope("trainer") as scope:
87         loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y, logits=logits)) + lossL2
88         train_step = tf.train.AdamOptimizer(0.0001).minimize(loss)
89
90     # 정확도를 계산하는 연산.
91     with tf.name_scope("evaluation") as scope:
92         correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y, 1))
93         accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
94
95
96 =====
97 # 5. 학습
98 =====
99     with tf.Session() as sess:
100     -----
101     # 5.1 세션, 변수 초기화
102     -----
103     sess.run(tf.global_variables_initializer())
104     writer = tf.summary.FileWriter("./board/sample", sess.graph)
```

최종 구현 및 모델 성능 평가

```
98      -----
99      # 5.2 Training loop
100     -----
101     total_epoch = 300
102     for e in range(total_epoch):
103         #.....
104         # 5.2.1 학습
105         #.....
106         total_size = x_train.shape[0]
107         batch_size = 128
108
109         loss_list = []
110         train_accuracy_list = []
111         for i in range(int(total_size / batch_size)):
112             #== batch load
113             batch_x = x_train[i*batch_size:(i+1)*batch_size]
114             batch_y = y_train_onehot[i*batch_size:(i+1)*batch_size]
115
116             #== train
117             sess.run(train_step, feed_dict={x: batch_x, y: batch_y, trainphase : True , keepprob:0.7})
118
119             #== logging
120             train_accuracy = accuracy.eval(feed_dict={x: batch_x, y: batch_y, trainphase : False , keepprob:1})
121             loss_print = loss.eval(feed_dict={x: batch_x, y: batch_y, trainphase : False , keepprob:1})
122             train_accuracy_list.append(train_accuracy)
123             loss_list.append(loss_print)
124
125             print("반복(Epoch):", e, "트레이닝 데이터 정확도:", np.mean(train_accuracy_list), "손실 함수(loss):",np.mean(loss_list))
126
```

반복(Epoch): 251 트레이닝 데이터 정확도: 1.0 손실 함수(loss): 0.0643254
테스트 데이터 정확도: 0.8421474

반복(Epoch): 252 트레이닝 데이터 정확도: 1.0 손실 함수(loss): 0.06303058
테스트 데이터 정확도: 0.8427484

반복(Epoch): 253 트레이닝 데이터 정확도: 1.0 손실 함수(loss): 0.06169955
테스트 데이터 정확도: 0.8420473

반복(Epoch): 254 트레이닝 데이터 정확도: 1.0 손실 함수(loss): 0.0602972
테스트 데이터 정확도: 0.8415465

반복(Epoch): 255 트레이닝 데이터 정확도: 1.0 손실 함수(loss): 0.05878525
테스트 데이터 정확도: 0.8407452

Translation Invariance

Translation Invariance



Rotation/Viewpoint Invariance



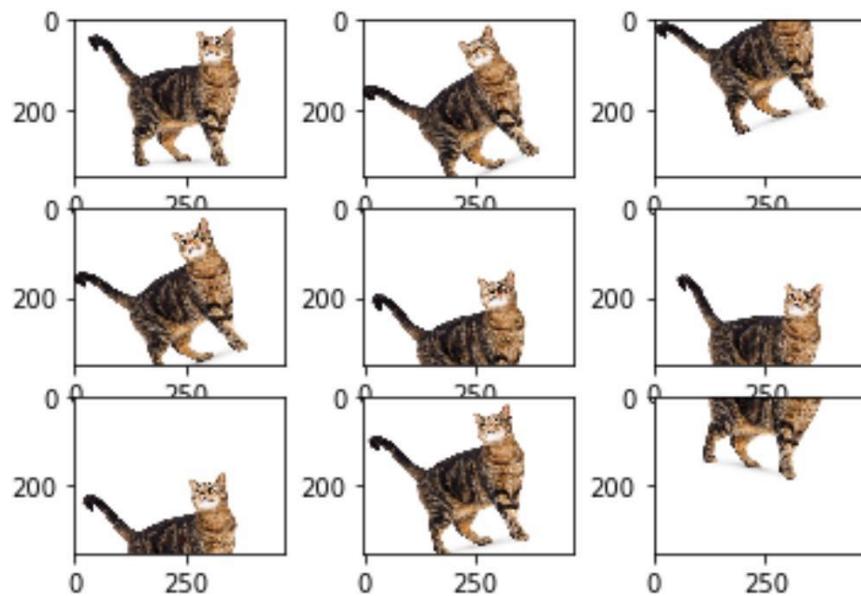
Size Invariance



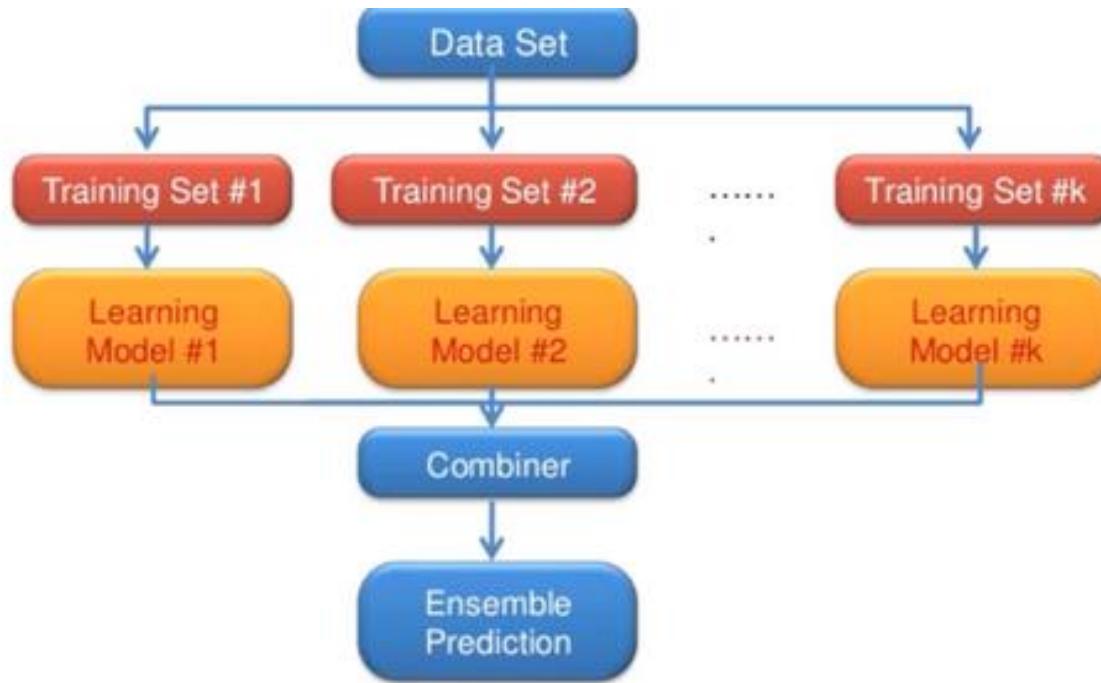
Illumination Invariance



- 왜곡된 이미지에 대하여도 강경한 성능을 얻기 위해서는 이러한 변화요소를 반영하는 다양한 학습 데이터를 필요로 한다
- 하지만 이러한 변화를 포함하는 학습데이터를 모으는 것은 쉽지 않다
- 기존의 학습데이터로부터 인공적으로 추가 학습 데이터를 생성한다



- 여러 모델들의 분류결과를 다수결로 취합하여 최종 결과로 사용한다
- 효과는 좋으나 사용하는 모델 수에 비례하여 처리시간이 배로 증가한다



여러 모델에서의 Softmax 레이어 출력 값을 평균 내어 사용한다

Advanced : Ensemble 구현

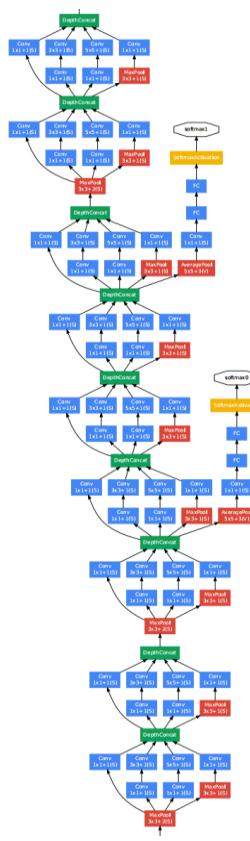
```
95 =====
96 # 5. 학습
97 =====
98 -----
99 # 5.1 세션, 변수 초기화
100 -----
101     num_ensembles = 2
102     sess_list = [ tf.Session() for i in range(num_ensembles) ]
103     for i in range(num_ensembles) : sess_list[i].run(tf.global_variables_initializer())
104
105 -----
106 # 5.2 Training loop
107 -----
108     total_epoch = 300
109     for e in range(total_epoch):
110         #.....
111         # 5.2.1 학습
112         #.....
113         total_size = x_train.shape[0]
114         batch_size = 128
115
116         for i in range(int(total_size / batch_size)):
117             #== batch load
118             batch_x = x_train[i*batch_size:(i+1)*batch_size]
119             batch_y = y_train_onehot[i*batch_size:(i+1)*batch_size]
120
121             #== train
122             for ens in range(num_ensembles):
123                 sess_list[ens].run(train_step, feed_dict={x: batch_x, y: batch_y, trainphase : True , keepprob:0.7})
124
```

Advanced : Ensemble 구현

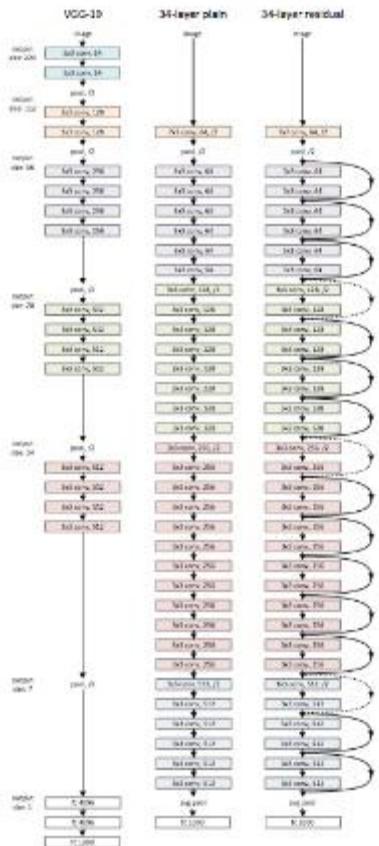
```
126      # .....
127      # 5.2.2 평가
128      #
129      # 매epoch 마다 test 데이터셋에 대한 정확도와 loss를 출력.
130      test_total_size = x_test.shape[0]
131      test_batch_size = 128
132
133      test_accuracy_list = []
134      for i in range(int(test_total_size / test_batch_size)):
135          # == test batch load
136          test_batch_x = x_test[i * test_batch_size:(i + 1) * test_batch_size]
137          test_batch_y = y_test_onehot[i * test_batch_size:(i + 1) * test_batch_size]
138
139          # == logging
140          predictions = []
141          test_accuracy = []
142          for ens in range(num_ensembles):
143              predictions.append(sess_list[ens].run(y_pred, feed_dict={x: test_batch_x, y: test_batch_y, trainphase : True , keepprob:0.7
144              test_accuracy.append(sess_list[ens].run(accuracy,
145                                  feed_dict={output_pred: np.mean(predictions[0:ens + 1], axis=0),
146                                  y: test_batch_y, trainphase : True , keepprob:0.7}))
147              test_accuracy_list.append(test_accuracy)
148
149          for ens in range(num_ensembles) :
150              print("[Ens:",ens,"] 테스트 데이터 정확도:",np.mean(test_accuracy_list[ens]))
151          print()
```

```
반복[epoch] : 211
[Ens: 0 ] 테스트 데이터 정확도: 0.859375
[Ens: 1 ] 테스트 데이터 정확도: 0.8546875
[Ens: 2 ] 테스트 데이터 정확도: 0.840625
[Ens: 3 ] 테스트 데이터 정확도: 0.890625
[Ens: 4 ] 테스트 데이터 정확도: 0.9265625
```

Advanced : More Deeper Network Architecture



(a) GoogLeNet



(b) ResNet

Revolution of Depth

