Deep learning

CHAPTER 5 합성곱 필터의 다층화를 통한 성능 향상

Ha-Jin Yu, Dept. of Computer Science, University of Seoul
서울시립대학교 컴퓨터과학부 유하진

2019.

HJYU@UOS.AC.KR

CHAPTER 5 합성곱 필터의 다층화를 통한 성능 향상

5.1 합성곱 신경망의 완성

5.1.1 다층형 합성곱 필터를 이용한 특징 추출

5.1.2 텐서플로를 이용한 다층 CNN 구현

5.2 그 밖의 주제

5.2.1 CIFAR-10(컬러 사진 이미지) 분류를 위한 확장

5.2.2 ‘A Neural Network Playground’를 이용한 직감적 이해

5.1 합성곱 신경망의 완성

- Image → convolution and pooling → filter 개수 만큼의 새로운 image data = feature (특징)
- Convolution and pooling 다시 적용
  → 더 새로운 특징

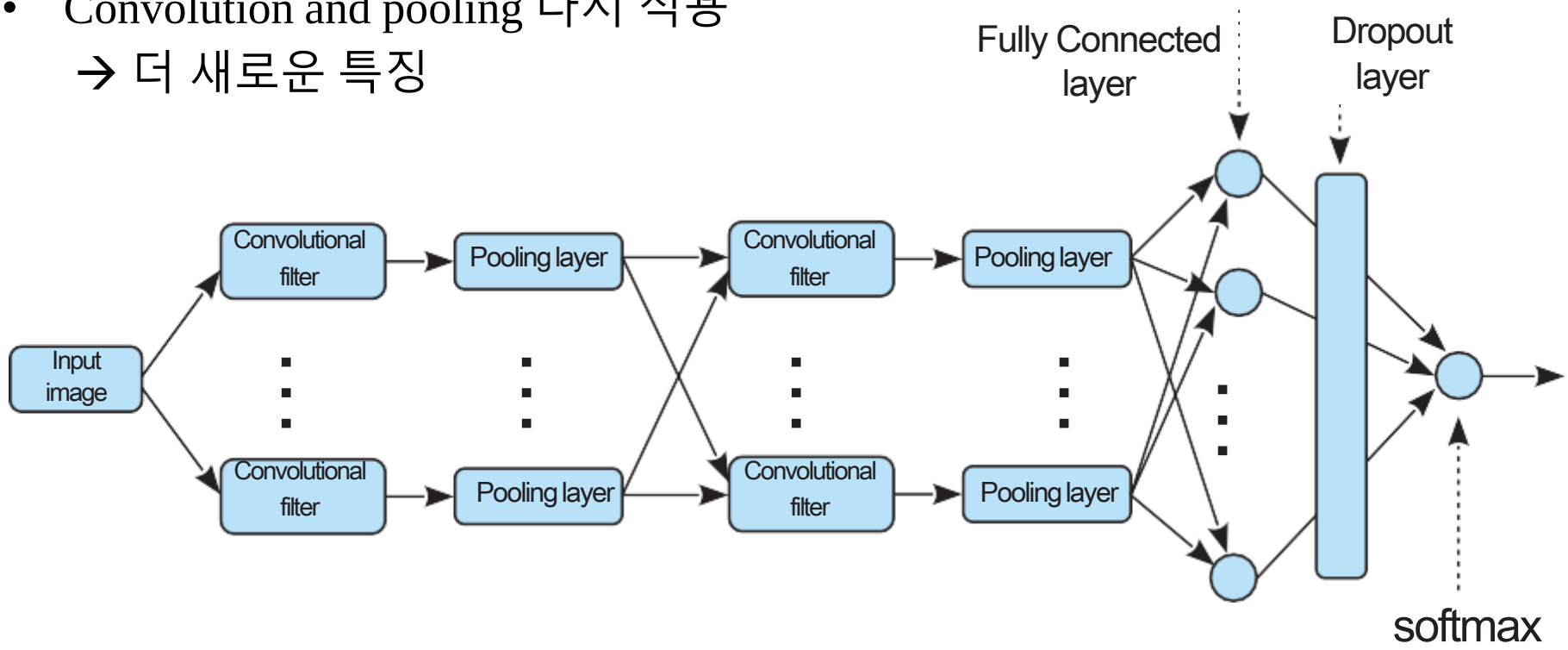
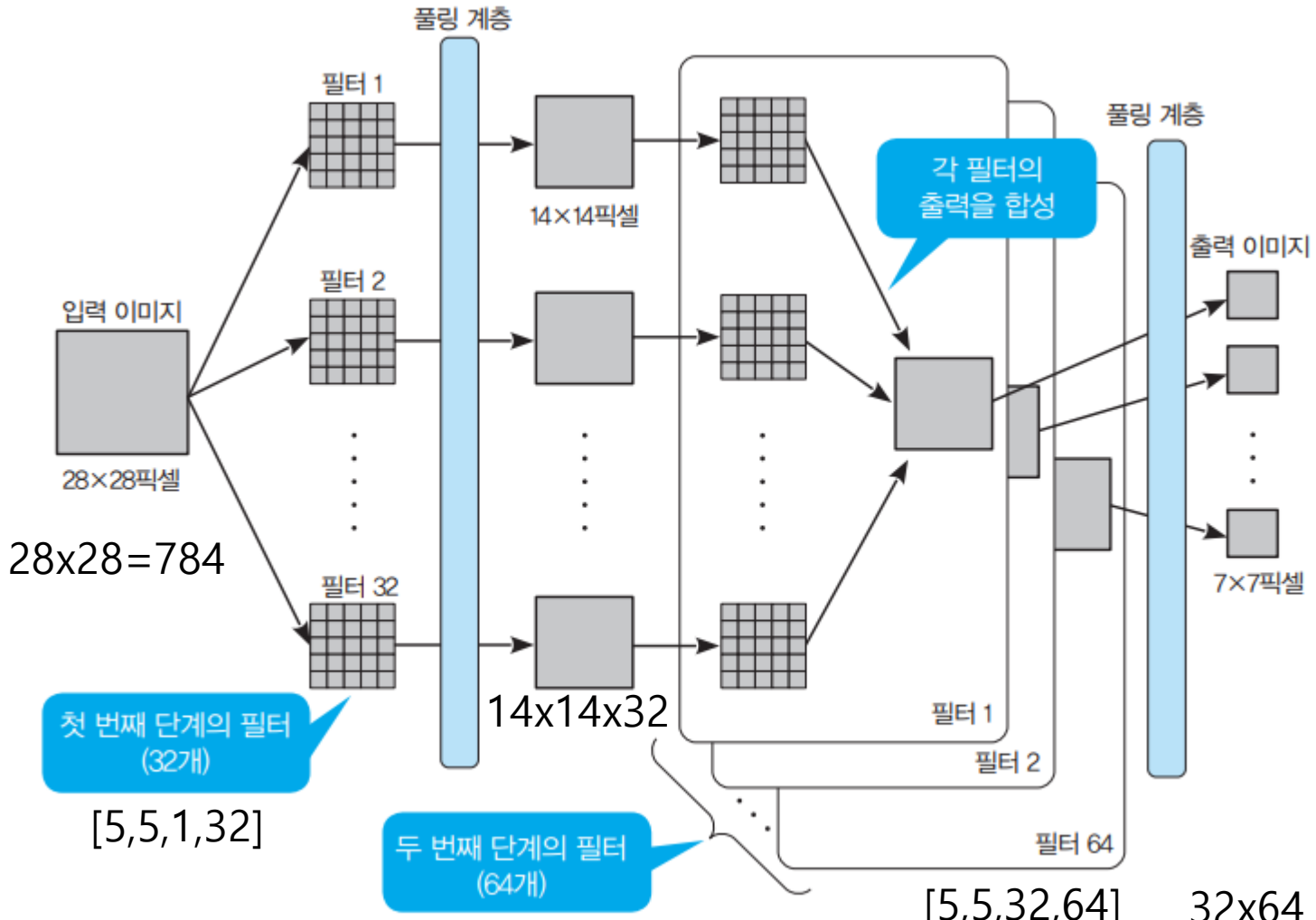


그림 5-1 CNN의 전체 모습

5.1.1 다층형 합성곱 필터를 이용한 특징 추출



[filter_size,input_layers,output_layers]

최종 출력 : 64개의 7x7 images

그림 5-2 2단계 합성곱 필터의 구성

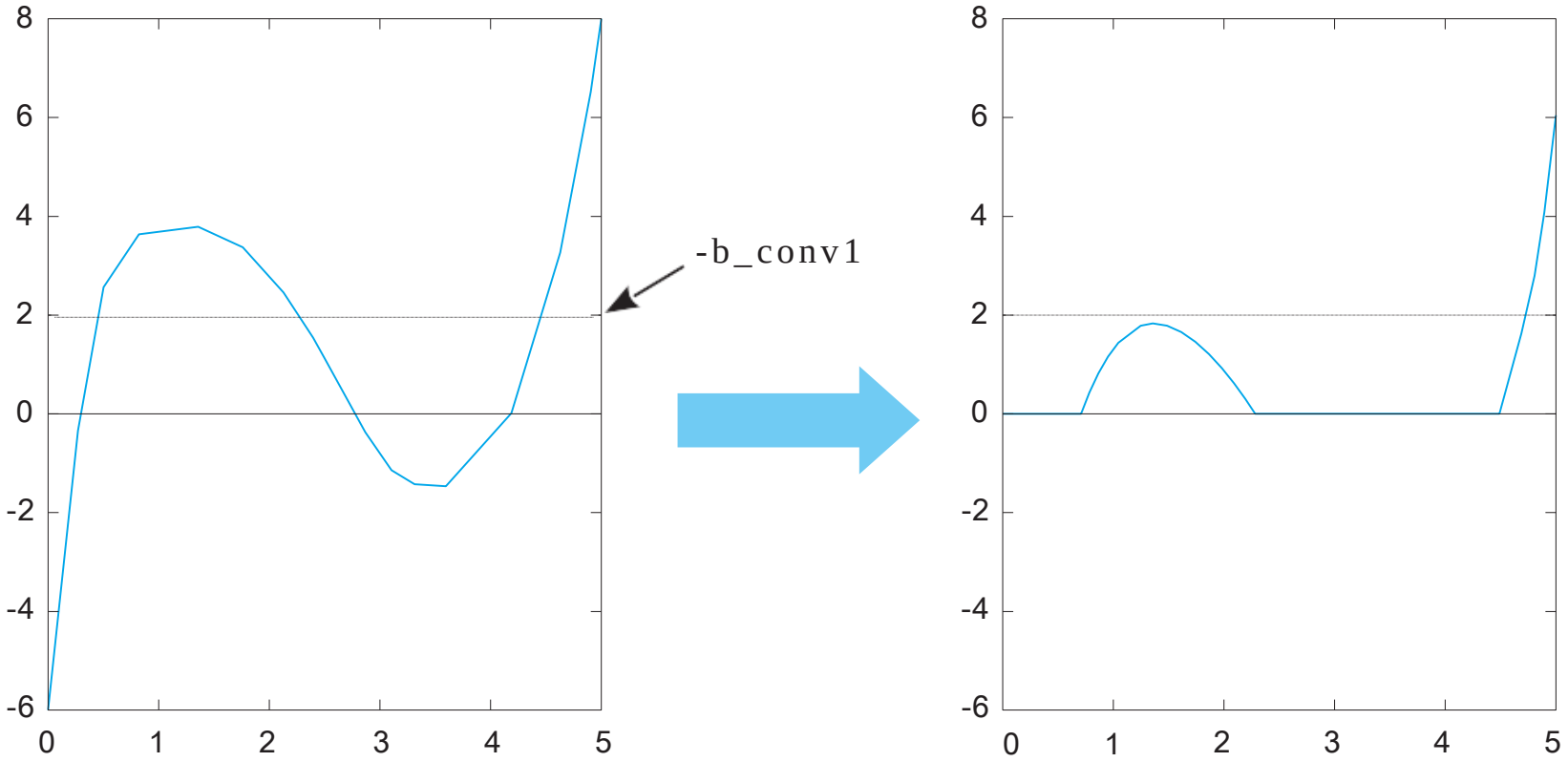
MNIST double layer CNN classification.ipynb

```
# first convolution and pooling layer
num_filters1 = 32
x = tf.placeholder(tf.float32, [None, 784])
x_image = tf.reshape(x, [-1,28,28,1])
W_conv1 = tf.Variable(tf.truncated_normal([5,5,1,num_filters1], stddev=0.1))
h_conv1 = tf.nn.conv2d(x_image, W_conv1, strides=[1,1,1,1], padding='SAME')
b_conv1 = tf.Variable(tf.constant(0.1, shape=[num_filters1]))
h_conv1_cutoff = tf.nn.relu(h_conv1 + b_conv1)
h_pool1 = tf.nn.max_pool(h_conv1_cutoff, ksize=[1,2,2,1],
          strides=[1,2,2,1], padding='SAME')
```

Initialize all to 0.1

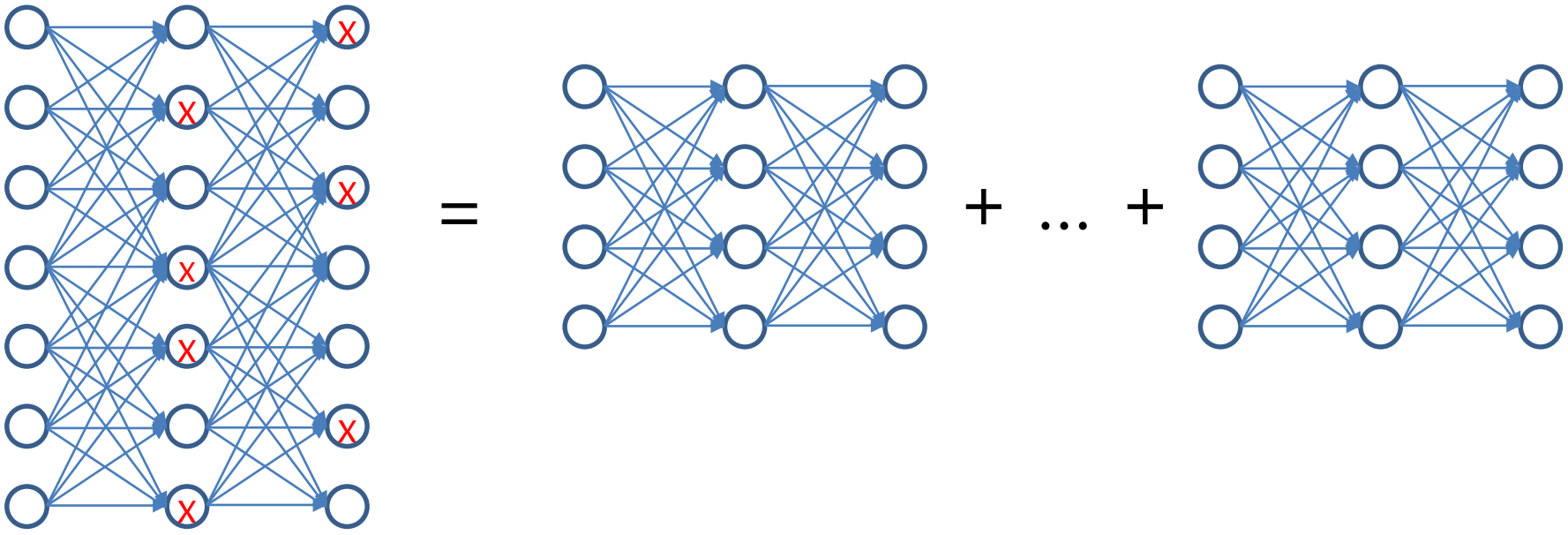
relu

그림 5-3 활성화 함수 ReLU로 픽셀값을 잘라 내는 예



Review: Dropout

- 문제: Overfitting 해결
- Dropout : Training example 마다 일부 hidden unit 을 없는 것으로 계산
- 다층 신경망의 유닛 중 일부를 확률적으로 선택하여 학습
- Randomly, 확률 $p = 0.5$ 적용
- 결국 2^n 개의 서로 다른 모델을 사용하는 효과



```
# CNN-04] 두 번째 단계의 합성곱 필터와 풀링 계층을 정의한다.
num_filters2 = 64
W_conv2 = tf.Variable(
        tf.truncated_normal([5,5,num_filters1,num_filters2], stddev=0.1))
h_conv2 = tf.nn.conv2d(h_pool1, W_conv2,
        strides=[1,1,1,1], padding='SAME')
b_conv2 = tf.Variable(tf.constant(0.1, shape=[num_filters2]))
h_conv2_cutoff = tf.nn.relu(h_conv2 + b_conv2)
h_pool2 = tf.nn.max_pool(h_conv2_cutoff, ksize=[1,2,2,1],
        strides=[1,2,2,1], padding='SAME')
```

[filter_size,input_layers,output_layers]

5.1.2 텐서플로를 이용한 다층 CNN 구현

```
# CNN-05] 전 결합층, 드롭아웃 계층, 소프트맥스 함수를 정의한다.
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*num_filters2])
num_units1 = 7*7*num_filters2
num_units2 = 1024
w2 = tf.Variable(tf.truncated_normal([num_units1, num_units2]))
b2 = tf.Variable(tf.constant(0.1, shape=[num_units2]))
hidden2 = tf.nn.relu(tf.matmul(h_pool2_flat, w2) + b2)
keep_prob = tf.placeholder(tf.float32)
hidden2_drop = tf.nn.dropout(hidden2, keep_prob)
w0 = tf.Variable(tf.zeros([num_units2, 10]))
b0 = tf.Variable(tf.zeros([10]))
p = tf.nn.softmax(tf.matmul(hidden2_drop, w0) + b0)
```

Dropout probability

```
sess.run(train_step, feed_dict={x:batch_xs, t:batch_ts, keep_prob:0.5})
```

Node를 잘라내는 비율에 따라 노드로 부터의 출력을 크게 … ex) x 2

```
# CNN-06] 오차 함수 loss, 트레이닝 알고리즘 train_step, 정답률 accuracy을 정의.
t = tf.placeholder(tf.float32, [None, 10])
loss = -tf.reduce_sum(t * tf.log(p))
train_step = tf.train.AdamOptimizer(0.0001).minimize(loss)
correct_prediction = tf.equal(tf.argmax(p, 1), tf.argmax(t, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

신경망이 복잡해 질수록 작은 학습률

Parameter optimization

```
i = 0
for _ in range(20000):
    i += 1
    batch_xs, batch_ts = mnist.train.next_batch(50)
    sess.run(train_step,
             feed_dict={x:batch_xs, t:batch_ts, keep_prob:0.5})
    if i % 500 == 0:
        loss_vals, acc_vals = [], []
        for c in range(4):
            start = int(len(mnist.test.labels) / 4 * c)
            end = int(len(mnist.test.labels) / 4 * (c+1))
            loss_val, acc_val = sess.run([loss, accuracy],
                feed_dict={x:mnist.test.images[start:end],
                           t:mnist.test.labels[start:end],
                           keep_prob:1.0})
            loss_vals.append(loss_val)
            acc_vals.append(acc_val)
        loss_val = np.sum(loss_vals)
        acc_val = np.mean(acc_vals)
        print ('Step: %d, Loss: %f, Accuracy: %f'
               % (i, loss_val, acc_val))
```

신경망이 복잡해 질수록 작은 batch size

Dropout: Training 할 때는 반만 사용

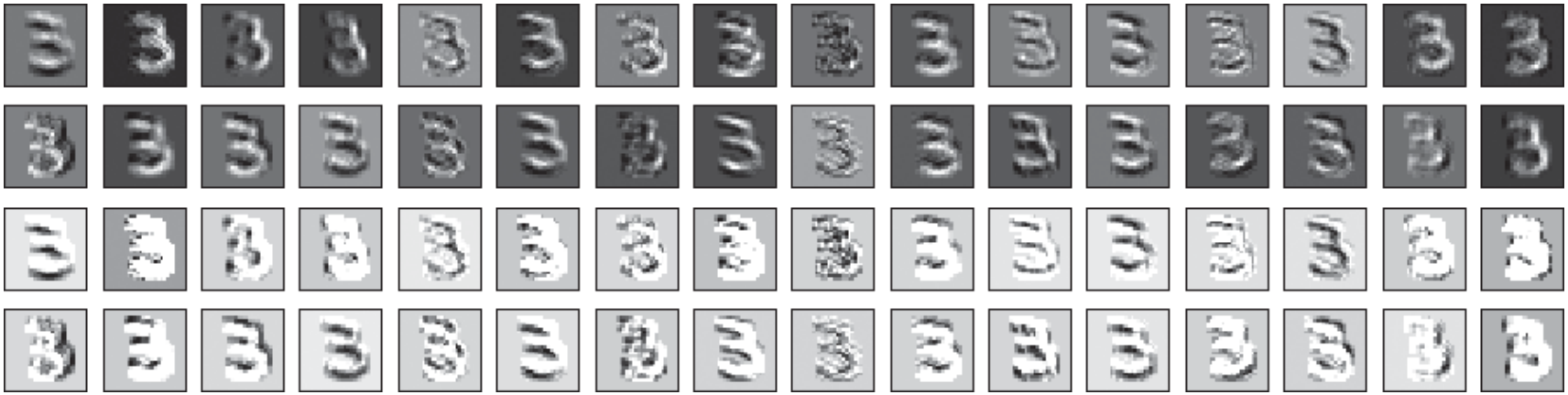
모든 데이터를 한꺼번에 평가하면 대량의 메모리가 필요하므로 4회로 나누어 평가

Dropout: Test 할 때는 모두 사용

최종적으로 test set에 대해 약 99%의 정답률

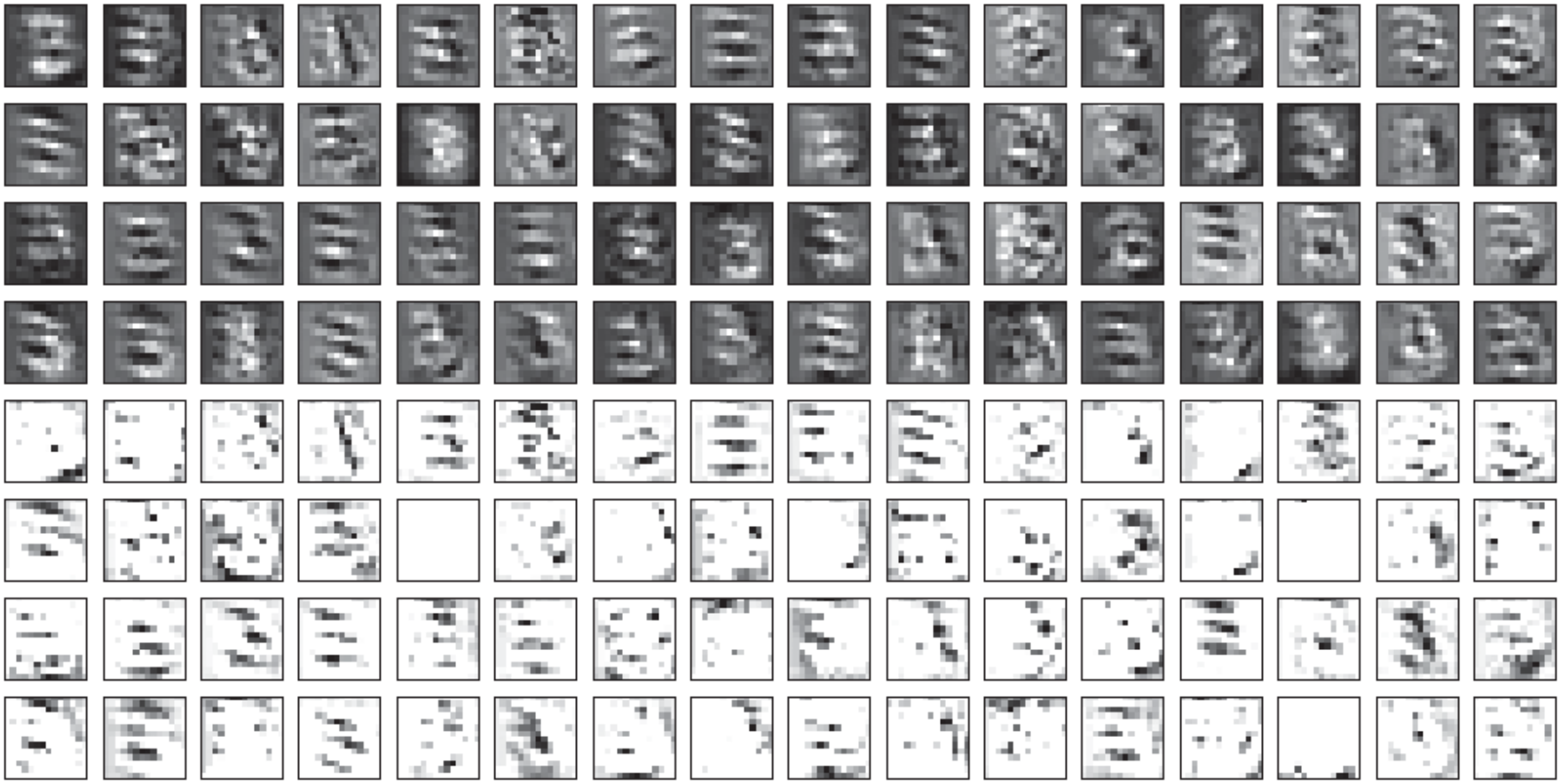
```
# HWR-09] 첫 번째 단계의 필터를 적용한 이미지를 출력한다.
# 여기서는 작은 픽셀값을 잘라 내기 전후의 각각의 이미지를 출력한다.

batch_xs, batch_ts = mnist.train.next_batch(50)
conv1_vals, cutoff1_vals = sess.run(
        [h_conv1, h_conv1_cutoff], feed_dict={x: batch_xs, keep_prob:1.0})
fig = plt.figure(figsize=(16,4))
for f in range(num_filters1):
        subplot = fig.add_subplot(4, 16, f+1)
        subplot.set_xticks([])
        subplot.set_yticks([])
        subplot.imshow(conv1_vals[0,:,:,f],
                cmap=plt.cm.gray_r, interpolation='nearest')
for f in range(num_filters1):
        subplot = fig.add_subplot(4, 16, num_filters1+f+1)
        subplot.set_xticks([])
        subplot.set_yticks([])
        subplot.imshow(cutoff1_vals[0,:,:,f],
                cmap=plt.cm.gray_r, interpolation='nearest')
```

그림 5-6 첫 번째 단계의 필터를 적용한 이미지 데이터



```
# HWR-10] 두 번째 단계의 필터를 적용한 이미지를 출력한다.
conv2_vals, cutoff2_vals = sess.run(
        [h_conv2, h_conv2_cutoff], feed_dict={x: batch_xs, keep_prob:1.0})
fig = plt.figure(figsize=(16,8))
for f in range(num_filters2):
        subplot = fig.add_subplot(8, 16, f+1)
        subplot.set_xticks([])
        subplot.set_yticks([])
        subplot.imshow(conv2_vals[0,:,:,f],
                cmap=plt.cm.gray_r, interpolation='nearest')
for f in range(num_filters2):
        subplot = fig.add_subplot(8, 16, num_filters2+f+1)
        subplot.set_xticks([])
        subplot.set_yticks([])
        subplot.imshow(cutoff2_vals[0,:,:,f],
                cmap=plt.cm.gray_r, interpolation='nearest')
```

그림 5-7 두 번째 단계의 필터를 적용한 이미지 데이터



Term project

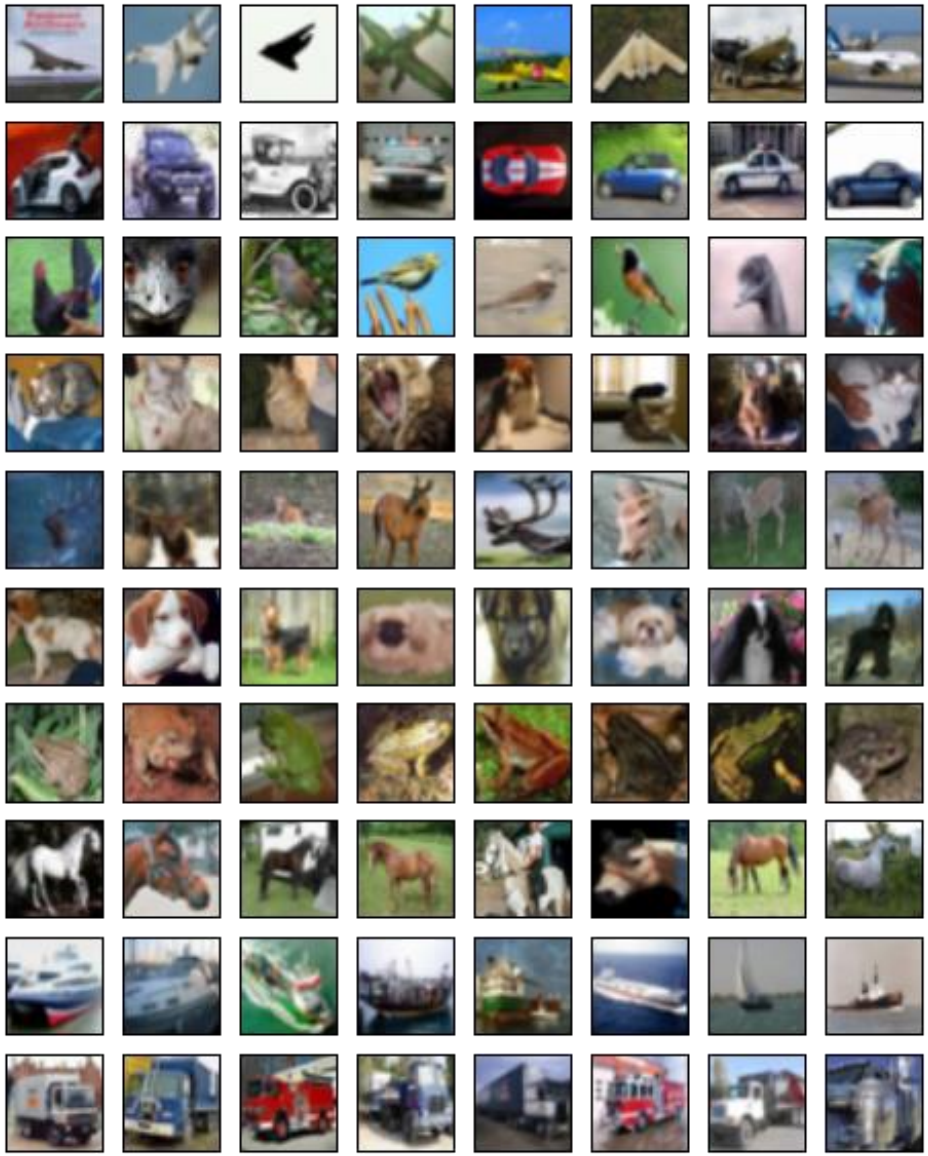
- MNIST 성능 향상 → test data에 대하여 오류 1% 이하로 개선
  - Tensorflow code를 여러가지 방법으로 확장 (자유 선택)
  - Convolution Hidden layer 수 증가 (필수)
  - Node 수 증가
  - Filter 수
  - Activation function
  - Data augmentation
  - …
- 제출 : 보고서, ipynb file

5.2.1 CIFAR-10 분류를 위한 확장

- CIFAR-10 classification is a common benchmark problem in machine learning.
- To classify RGB 32x32 pixel images
- 10 categories:
  - airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

- http://www.cs.toronto.edu/~kriz/cifar-10-binary.tar.gz
- http://www.cs.toronto.edu/~kriz/cifar.html
- https://github.com/tensorflow/models/tree/master/tutorials/image

그림 5-8 CIFAR-10의 데이터 세트(일부)



비행기

자동차

새

고양이

사슴

개

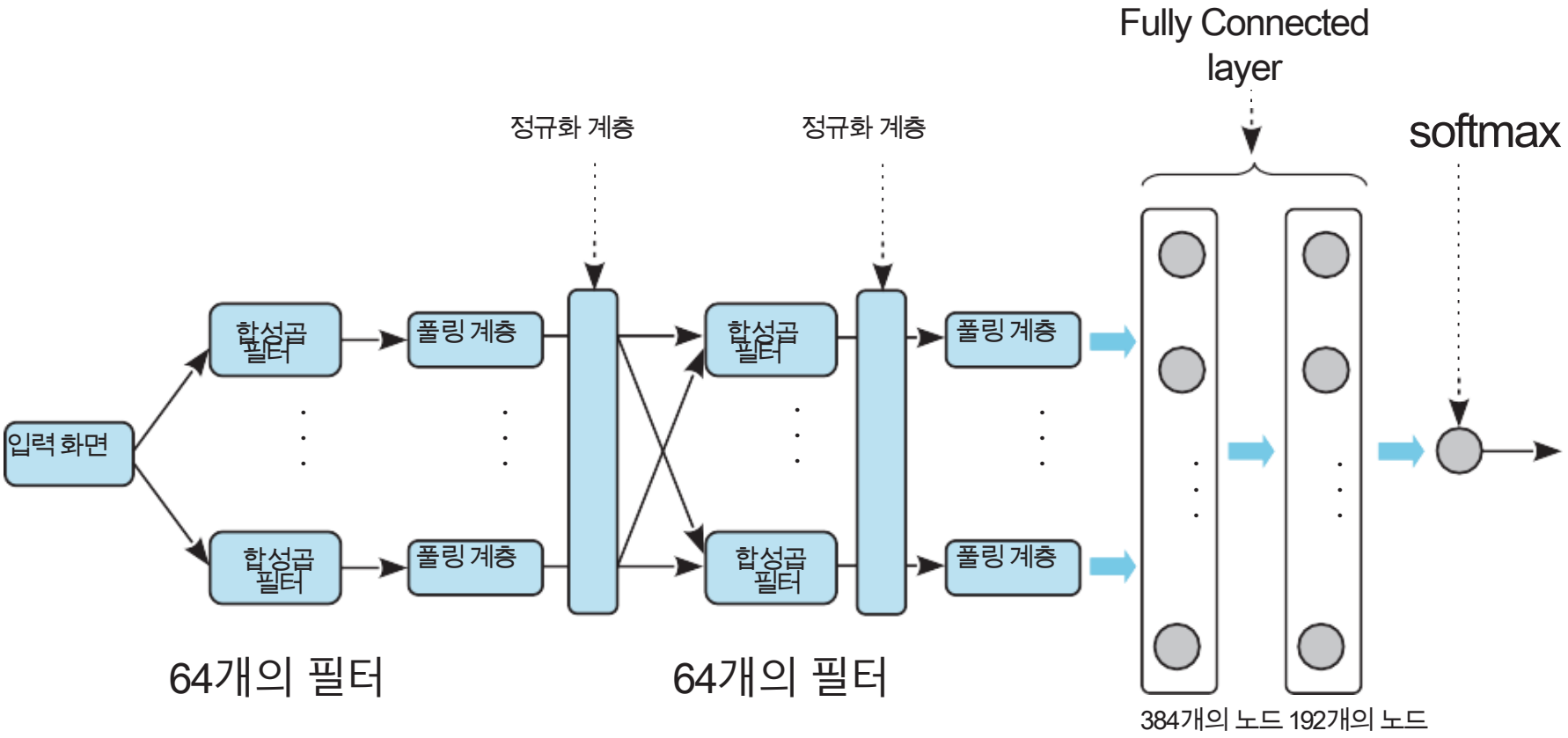
개구리

말

배

트럭

그림 5-9 CIFAR-10의 데이터 세트를 분류하는 CNN 구성



- 정규화 계층: Pixel 값이 극단적으로 커지지 않도록 pixel 값의 범위를 압축

Image preprocessing for test data

- Cropping : image 주변을 잘라낸다.
- Whitening : 하나의 image data에 포함되어 있는 각 pixel 값의 범위를 조정해서 평균 0, 표준편차 1인 범위 안에 들어가도록 조절
- x_i : pixel value
- m : mean
- s^2 : varience

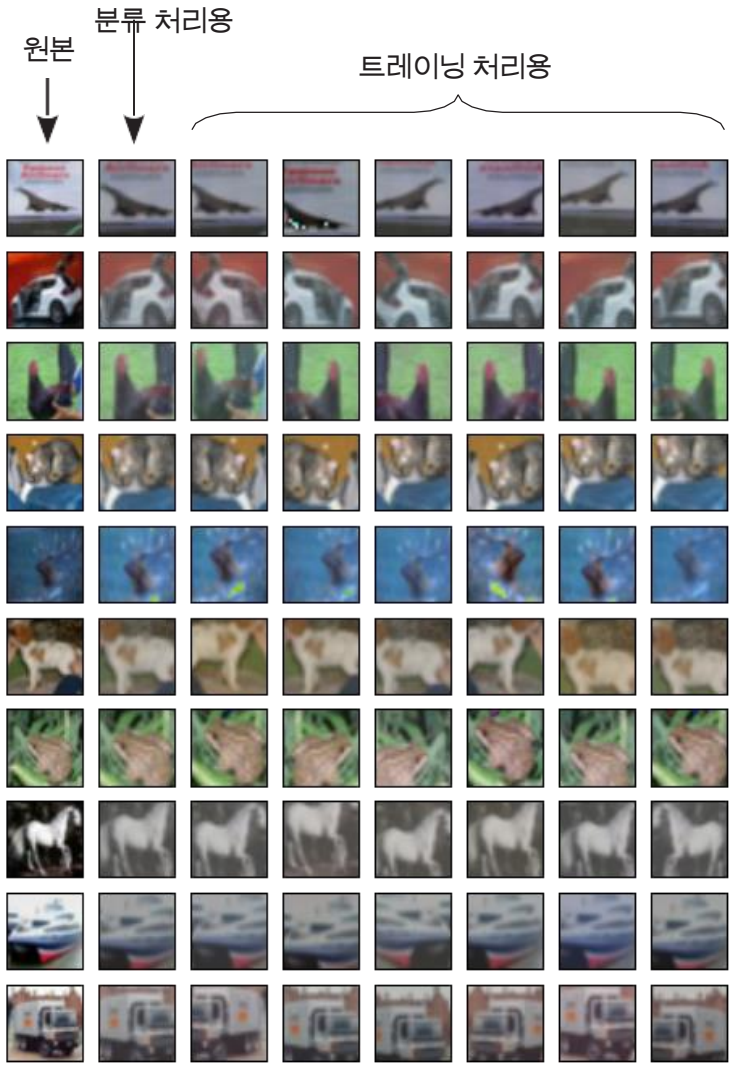
$$m = \frac{1}{N}\sum_{i=1}^{N} x_i, \quad s^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - m)^2$$

$$x_i \rightarrow \frac{x_i - m}{\sqrt{s^2}}$$

Image preprocessing for training

- Random cropping : image 주변을 random 하게 잘라냄.
- Random flipping : random 하게 좌우 반전
- 밝기와 contrast 를 random 하게 변경
- Whitening (Data normalization)

그림 5-10 전처리를 한 이미지 데이터의 예



5.2.2 'A Neural Network Playground'를 이용한 직감적 이해

- http://playground.tensorflow.org
- Javascript로 구현

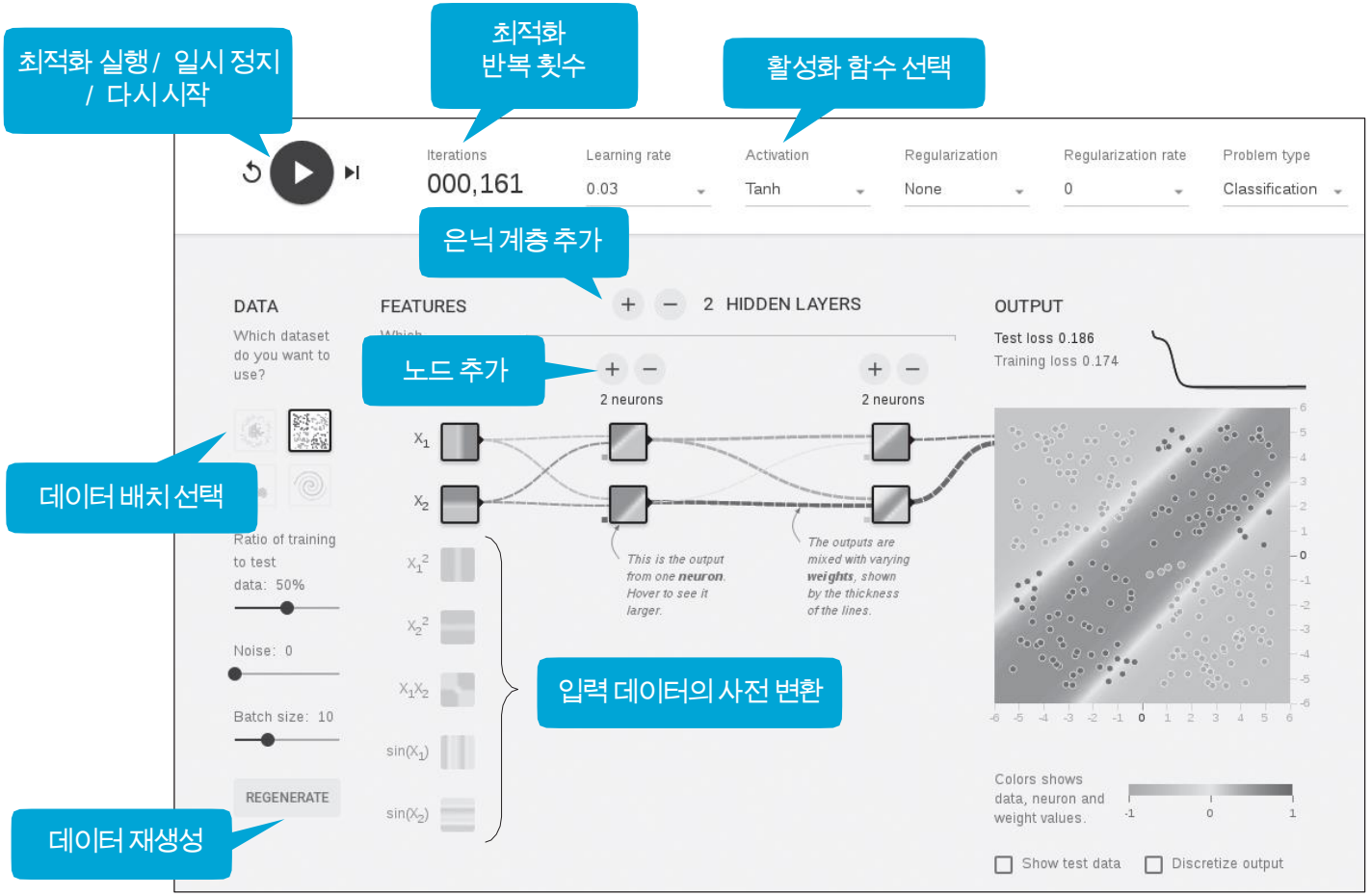
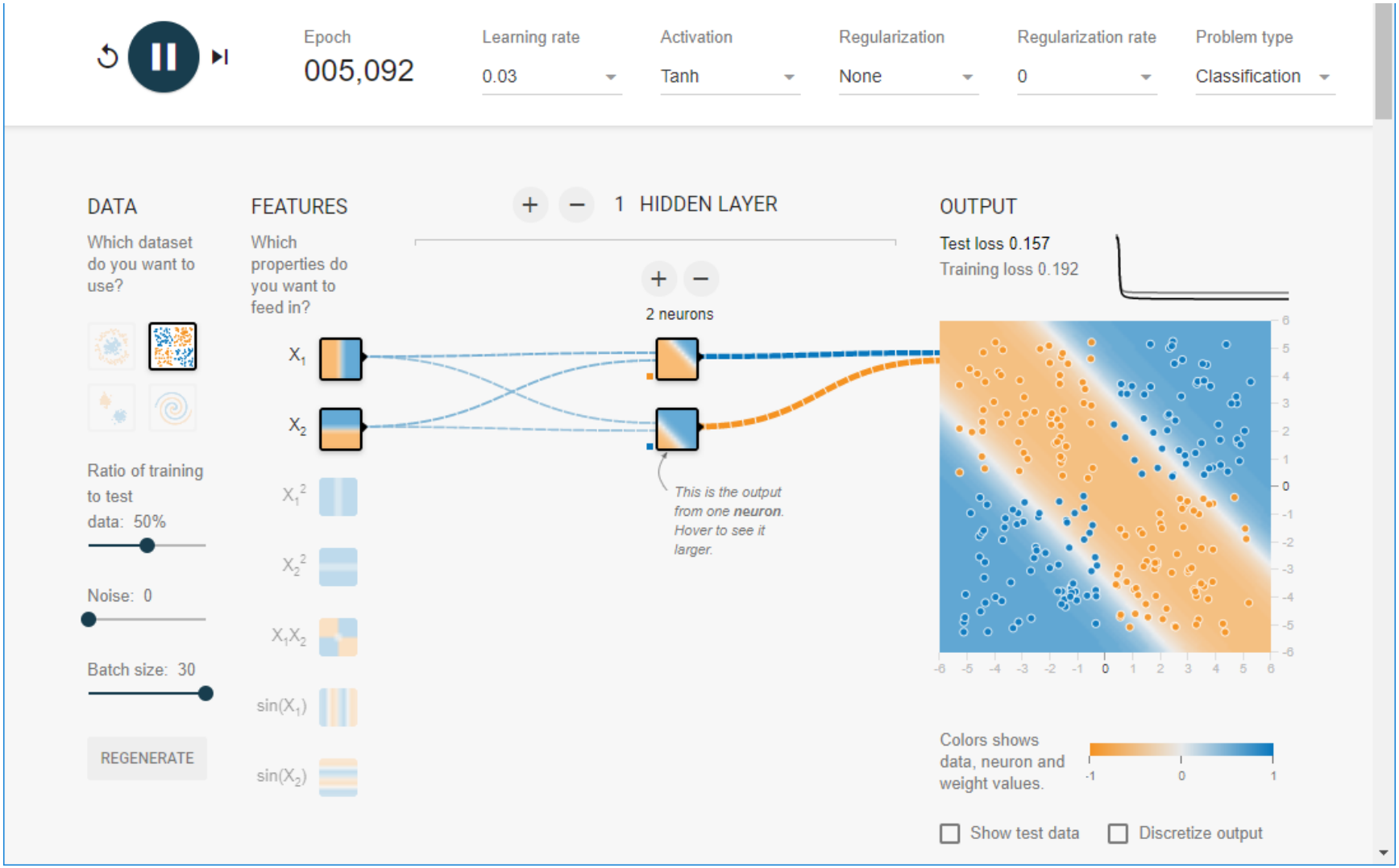


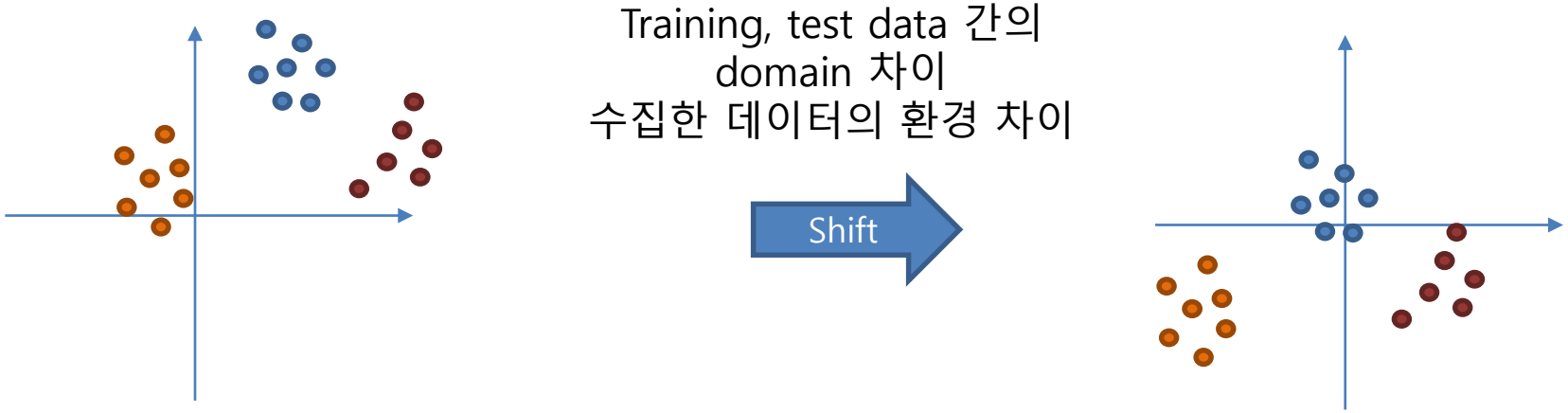
그림 5-11 'A Neural Network Playground'의 제어 화면

Local minimun



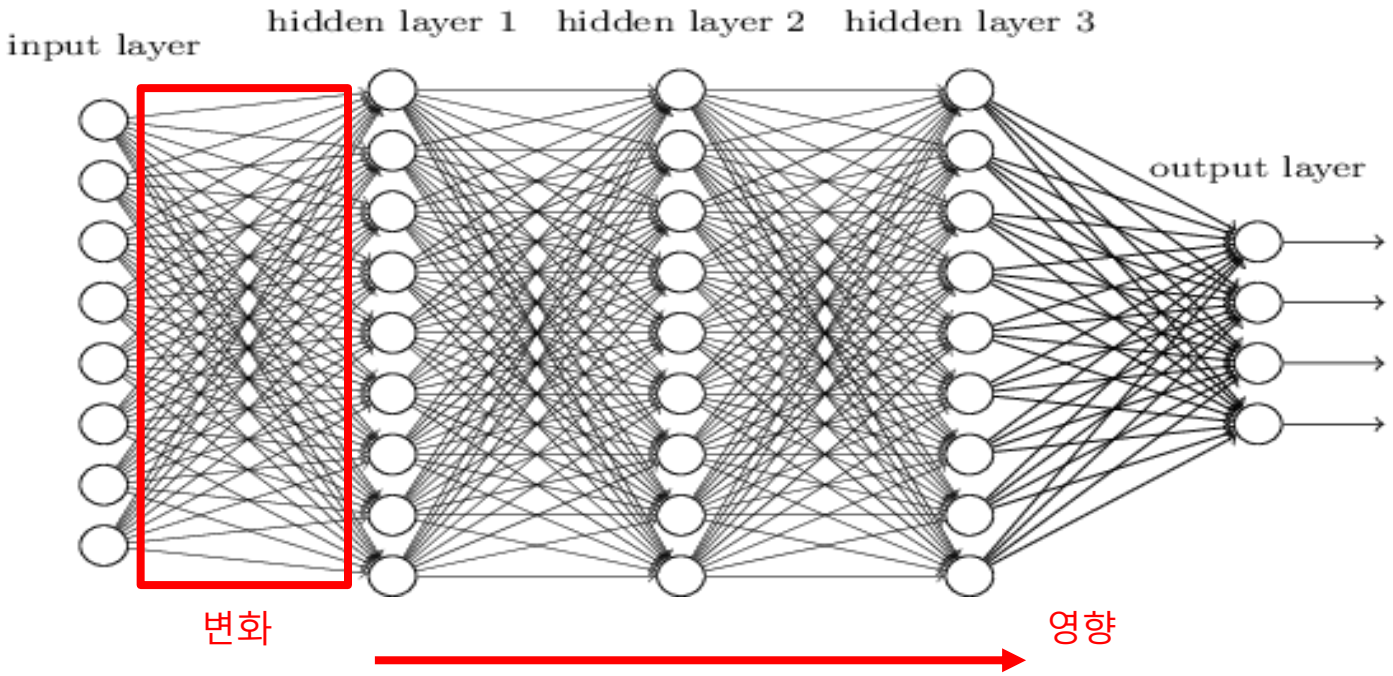
Batch Normalization (2015)

- Accelerating Deep Network Training by Reducing Internal Covariate Shift
- Covariate shift
  - Input data의 Covariate 변화 현상
  - Data normalization, model adaptation 등으로 문제 해결



Internal Covariate shift

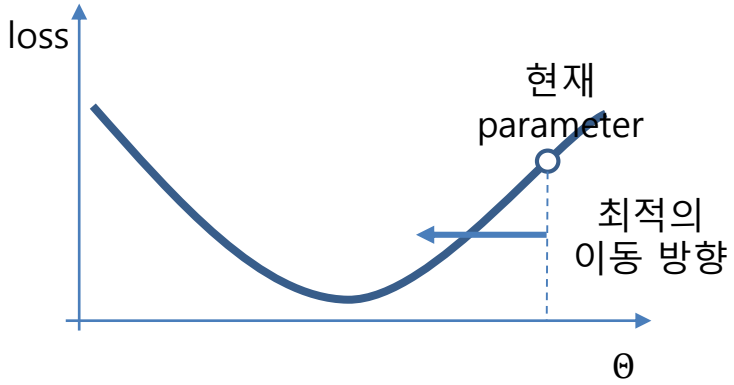
- DNN 학습 과정에서 발생할 수 있는 Covariate shift
  - DNN의 학습 과정에서 변화된 parameter 들이 다음 layer의 입력 값에 영향



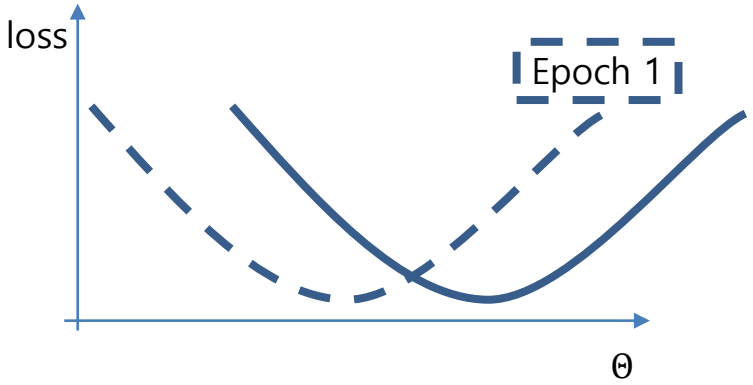
http://neuralnetworksanddeeplearning.com/chap6.html

Internal Covariate shift

- Parameter 변화에 의해 SGD 학습의 정상적인 동작이 보장 안됨
  - Ex) DNN의 두 번째 hidden layer의 학습 과정
    주어진 input data에 대해 loss function을 최소화시키는 방향으로 학습
    → 주어진 input data의 변화 → loss function 값 변화



두 번째 hidden layer의 parameter 학습
(Epoch 1)



두 번째 hidden layer의 parameter에
따른 loss 값
(Epoch 2)

Batch normalization

- Batch 단위로 각 layer의 출력 값을 normalization
- Scaling, shift parameter를 추가 (gamma, beta) (To recover the identity mapping)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

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

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

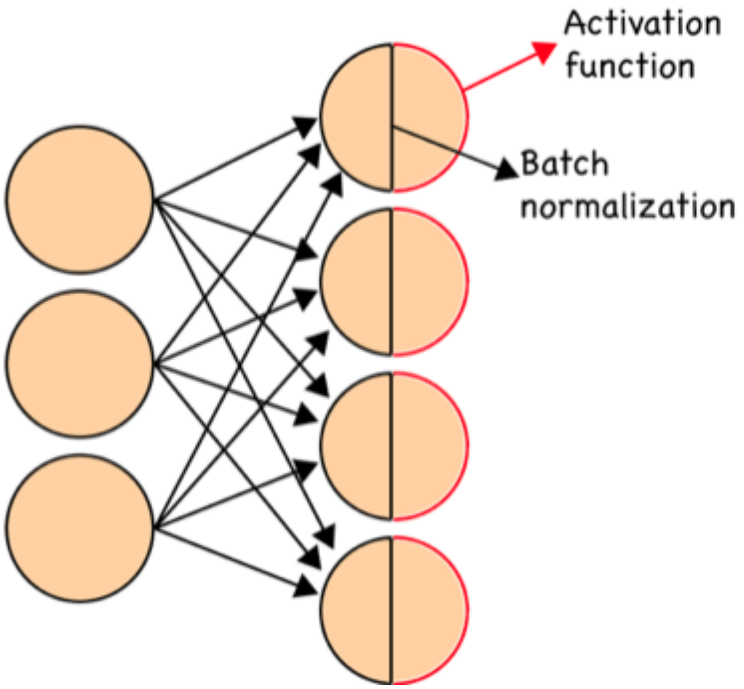
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

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

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

Batch normalization 구현

- Weight 연산 → batch normalization → activation function



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

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