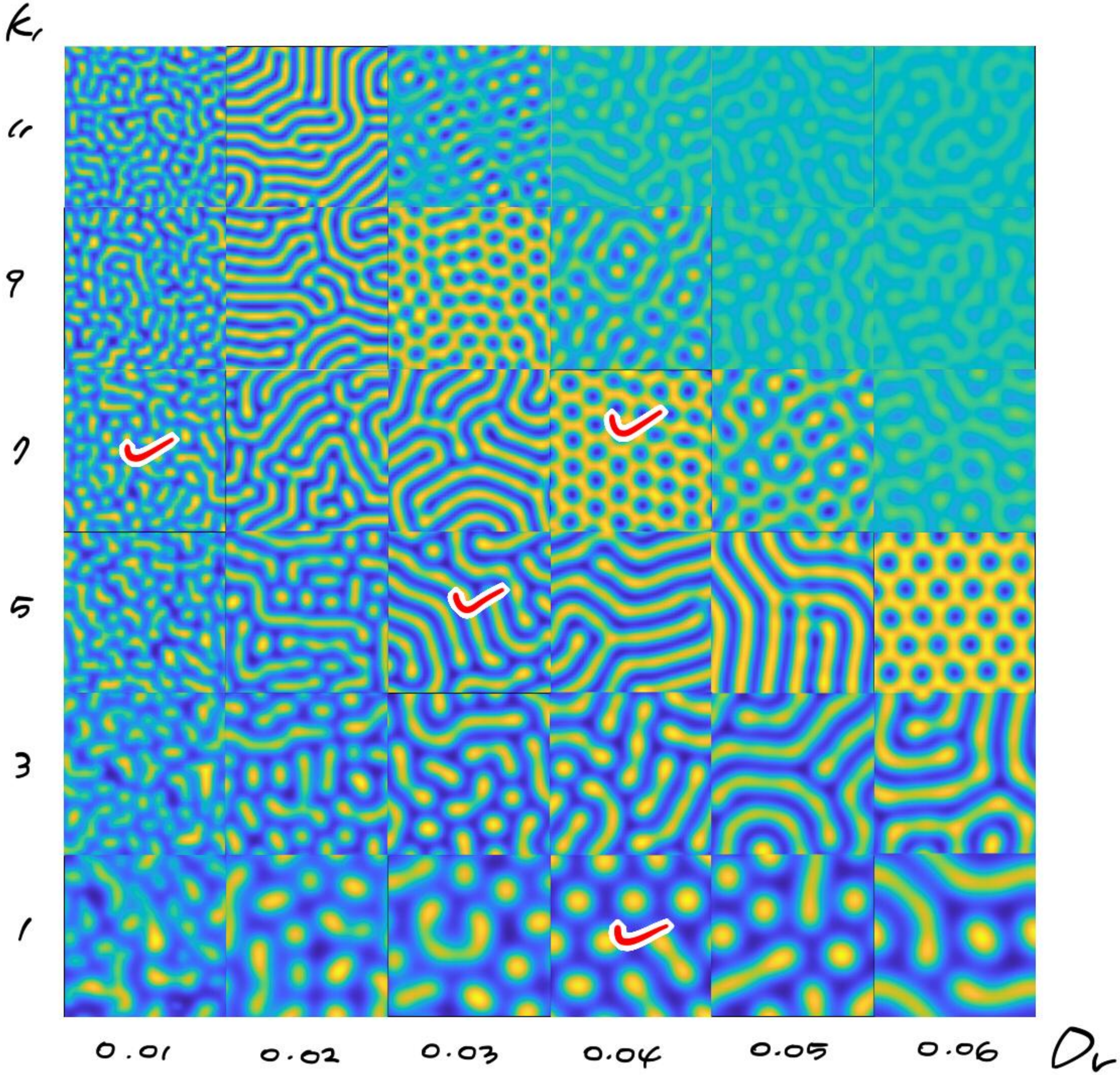
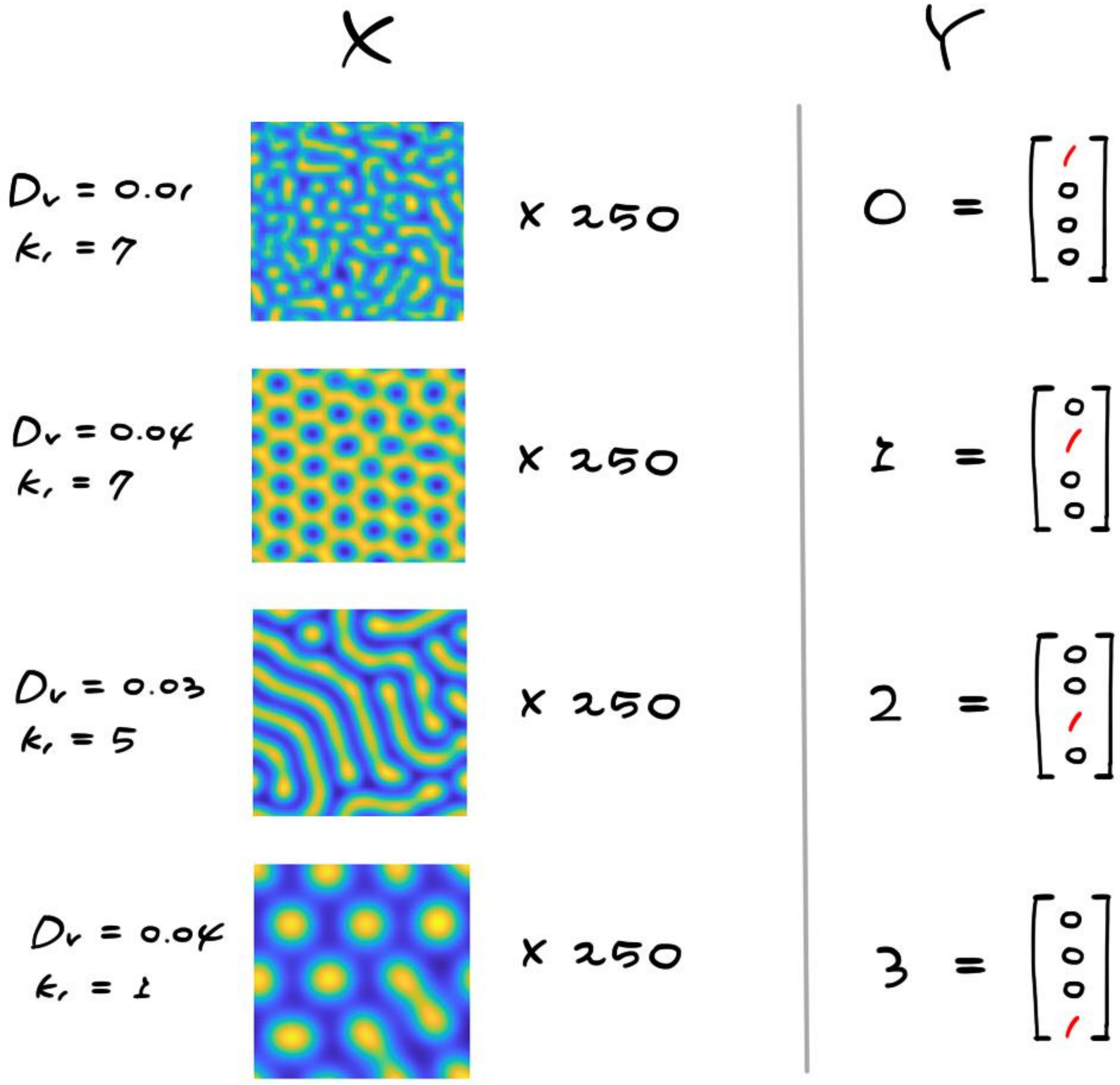
# Softmax Algorithm

2017010698
수학과 오서영



$k_r$
11
9
7
5
3
1
0.01
0.02
0.03
0.04
0.05
0.06
$D_r$


X
Y
Dv = 0.01
k1 = 7
x 250
0 = [1 0 0 0]
Dv = 0.04
k1 = 7
x 250
1 = [0 1 0 0]
Dv = 0.03
k1 = 5
x 250
2 = [0 0 1 0]
Dv = 0.04
k1 = 1
x 250
3 = [0 0 0 1]

$$x \to z = \underbrace{wx+b}_{\text{training}} \to g(z) \to \bar{y}$$

# Softmax

$$: S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

$$\underset{\text{score}}{z = \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}} \to \underset{\text{probability}}{\bar{y} = \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}}$$

$$\bar{Y} = \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix} \xleftrightarrow[\text{argmax}]{} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

# Cost function

: cross entropy

$$L(\underset{\begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}}{\bar{Y}}, \underset{\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}}{Y}) = -\frac{1}{m} \cdot \sum_i y_i \log(\bar{y}_i)$$

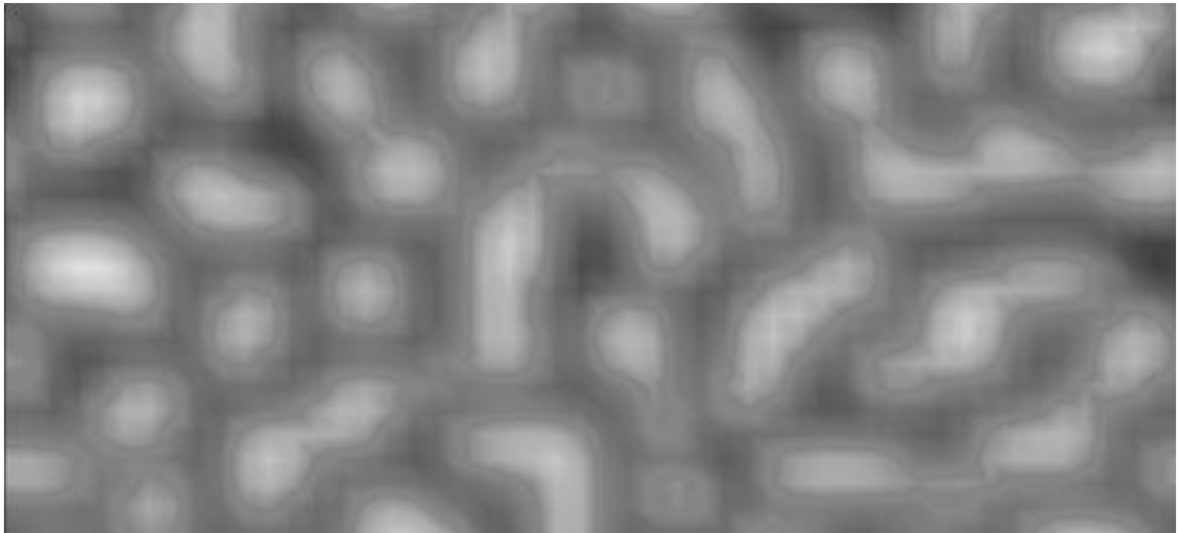
In [115]:

```python
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from tensorflow.python.framework import ops
# import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import math
```

In [32]:

```python
# Show an image
img = Image.open('0/pattern_1.jpg')
img
```

Out[32]:



```
ke dataset
g = []
g = np.zeros((1,250))
in range(1,1001):
 <= 250 :
folder = 0
f i <=500 :
folder = 1
f i <= 750 :
folder = 2
se : folder = 3
g = Image.open('{0}/pattern_{1}.jpg'.format(folder,i))
ta = np.array(img)
orig.append(data)

in range(1,4):
orig = np.append(y_orig, np.full((1, 250),i), axis = 1)
```

```
g = np.array(x_orig)
x_orig.shape)
y_orig.shape)
```

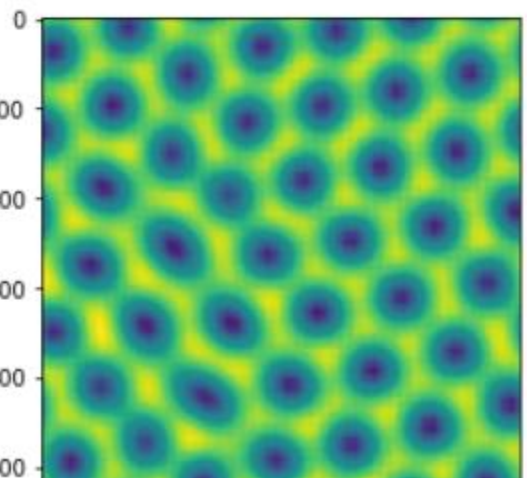
```
 533, 533)
00)
```

```
= np.arange(x_orig.shape[0])
p.random.shuffle(s)

_shuffle = x_orig[s,:]
_shuffle = y_orig[:,s]

rint(x_shuffle.shape)
rint(y_shuffle.shape)
 y_shuffle
```

```
000, 533, 533)
, 1000)
```

```
 Example of a picture
ndex = 990
lt.imshow(x_shuffle[index,:])
rint ("y = " + str(np.squeeze(y_shuffle[:, index])))
```

```
= 1.0
```



```
Split train and test datasets
rain_orig, x_test_orig, y_train_orig, y_test_orig = train_test_split(x_shuffle,y_shuffle.T,
                                                     test_size=0.2,  shuffle=True, random_state=1004)
```

```
rain_orig.shape
```

```
, 1)
```

```
latten the training and test images
rain_flatten = x_train_orig.reshape(x_train_orig.shape[0], -1).T
est_flatten = x_test_orig.reshape(x_test_orig.shape[0], -1).T

lormalize image vectors
rain = x_train_flatten/255.
est = x_test_flatten/255.

Convert training and test labels to one hot matrices
c = OneHotEncoder()
= y_train_orig.reshape(-1,1)
c.fit(y1)
rain = enc.transform(y1).toarray()
rain = y_train.T

= y_test_orig.reshape(-1,1)
c.fit(y2)
est = enc.transform(y2).toarray()
est = y_test.T
```

```python
# Explore your dataset
print ("number of training examples = " + str(x_train.shape[1]))
print ("number of test examples = " + str(x_test.shape[1]))
print ("X_train shape: " + str(x_train.shape))
print ("Y_train shape: " + str(y_train.shape))
print ("X_test shape: " + str(x_test.shape))
print ("Y_test shape: " + str(y_test.shape))
```

```
number of training examples = 800
number of test examples = 200
X_train shape: (284089, 800)
Y_train shape: (4, 800)
X_test shape: (284089, 200)
Y_test shape: (4, 200)
```

```python
def create_placeholders(nx, ny):

    X = tf.placeholder(tf.float32,[nx,None],name = 'X')
    Y = tf.placeholder(tf.float32,[ny,None],name = 'Y')

    return X, Y
```

```python
def initialize_parameters():

    '''
    The shapes are
    W : [4, 284089] , b : [4, 1]
    Z = WX + b
    '''

    tf.set_random_seed(1)  # so that your "random" numbers match ours

    W = tf.Variable(tf.glorot_uniform_initializer()((4,284089)))
    b = tf.get_variable("b", [4,1], initializer = tf.zeros_initializer())

    parameters = {"W": W,
                  "b": b}

    return parameters
```

```python
def forward_propagation(X, parameters):

    # Z -- the output of linear

    W = parameters['W']
    b = parameters['b']

    Z = tf.add(tf.matmul(W,X),b)
    # A = tf.nn.softmax(Z)

    return Z
```

```python
def compute_cost(Z, Y):

    z = tf.transpose(Z)
    y = tf.transpose(Y)
    # softmax_cross_entropy_with_logits()가 softmax()를 포함하기 때문에 A대신 Z 입력
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits = z, labels = y))

    return cost
```