03 MNIST - DROPOUT 추가

학습 내용

- dropout 이해와 실습해 보기

01 Dropout(드롭아웃) 설명

- 과적합의 이해 - 학습한 결과가 학습 데이터에는 매우 잘 맞지만, 학습 데이터에만 너무 꼭 맞춰져 있어, 그 외의 데이터에는 잘 맞지 않음.
- 학습시 전체 신경망 중 **일부**만을 사용하도록 하는 것.
- 즉, 학습 단계마다 **일부 뉴런을 제거(사용하지 않도록)**함으로써, 일부 특징이 특정 뉴런에 고정되는 것을 막아 가중치의 균형을 잡도록 한다.
- 학습 시 **일부 뉴런을 학습시키지 않기 때문에** 신경망이 **충분히 학습되기까지**의 시간은 전체를 사용하는 것보다 조금 **더 오래 걸리는 편**이다.

In [1]:

```
import os, warnings
# 경고 메시지 무시하거나 숨길때(ignore), 다시보이게(default)
# warnings.filterwarnings(action='default')
warnings.filterwarnings(action='ignore')
```

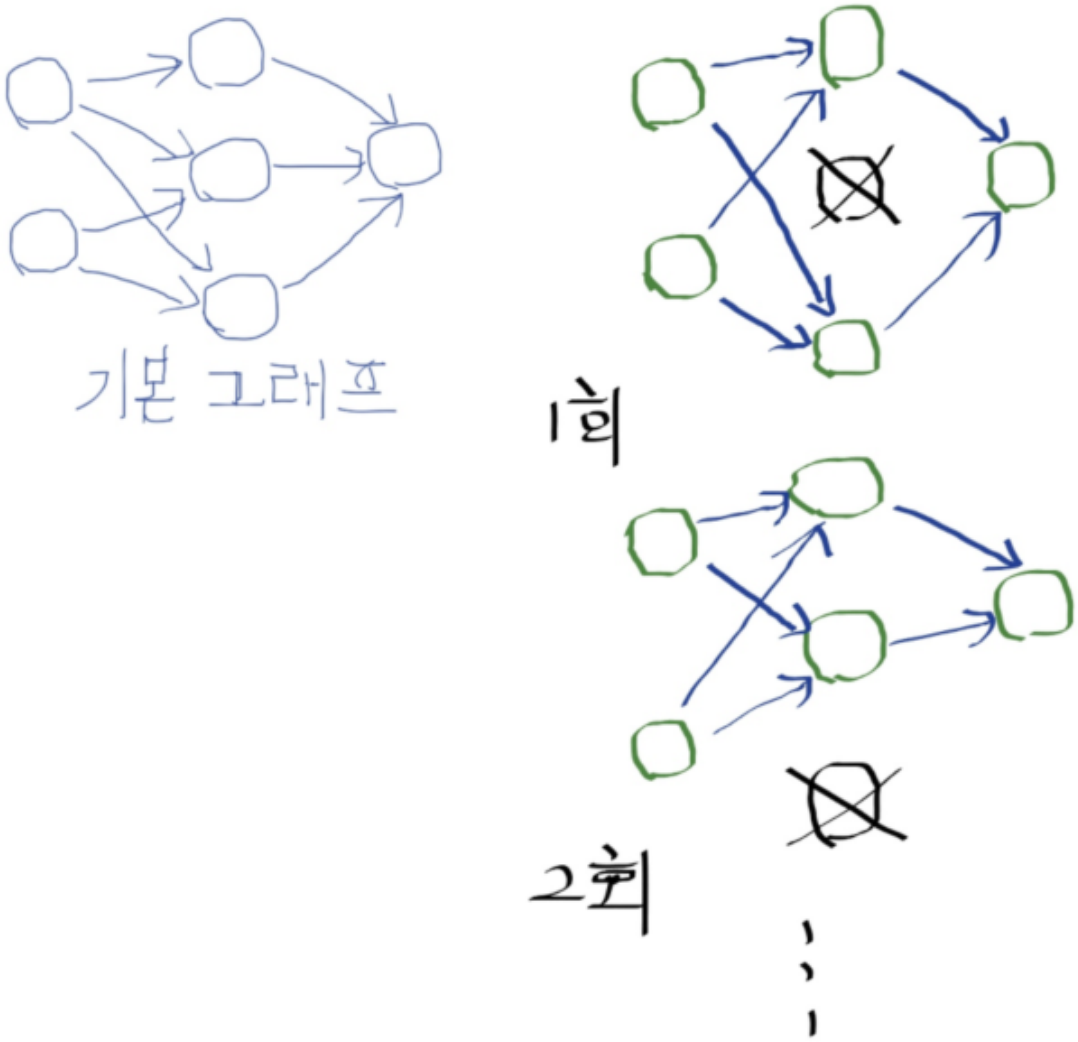
In [2]:

```
from IPython.display import display, Image
```

각 층별 일부 노드를 제거

In [4]:

```
1 display(Image(filename="../img/dropout1.png"))
```



In [6]:

```
import tensorflow as tf
print(tf.__version__)
```

1.15.0

In [7]:

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)
type(mnist)
```

```
WARNING:tensorflow:From <ipython-input-7-5353b57003af>:2: read_data_sets (from tenso
rflow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be removed i
n a future version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
WARNING:tensorflow:From C:₩Users₩peop₩Anaconda3₩lib₩site-packages₩tensorflow_core₩co
ntrib₩learn₩python₩learn₩datasets₩mnist.py:260: maybe_download (from tensorflow.cont
rib.learn.python.learn.datasets.base) is deprecated and will be removed in a future
version.
Instructions for updating:
Please write your own downloading logic.
WARNING:tensorflow:From C:₩Users₩peop₩Anaconda3₩lib₩site-packages₩tensorflow_core₩co
ntrib₩learn₩python₩learn₩datasets₩mnist.py:262: extract_images (from tensorflow.cont
rib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a future
version.
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting ./mnist/data/train-images-idx3-ubyte.gz
WARNING:tensorflow:From C:₩Users₩peop₩Anaconda3₩lib₩site-packages₩tensorflow_core₩co
ntrib₩learn₩python₩learn₩datasets₩mnist.py:267: extract_labels (from tensorflow.cont
rib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a future
version.
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting ./mnist/data/train-labels-idx1-ubyte.gz
WARNING:tensorflow:From C:₩Users₩peop₩Anaconda3₩lib₩site-packages₩tensorflow_core₩co
ntrib₩learn₩python₩learn₩datasets₩mnist.py:110: dense_to_one_hot (from tensorflow.co
ntrib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a futu
re version.
Instructions for updating:
Please use tf.one_hot on tensors.
Extracting ./mnist/data/t10k-images-idx3-ubyte.gz
Extracting ./mnist/data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From C:₩Users₩peop₩Anaconda3₩lib₩site-packages₩tensorflow_core₩co
ntrib₩learn₩python₩learn₩datasets₩mnist.py:290: DataSet.__init__ (from tensorflow.co
ntrib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a futu
re version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
```

Out[7]:

```
tensorflow.contrib.learn.python.learn.datasets.base.Datasets
```

In [8]:

```
# 데이터의 개수
print(mnist.test.num_examples, mnist.train.num_examples, mnist.validation.num_examples)

# 데이터의 행열 사이즈
print(mnist.train.labels.shape, mnist.train.images.shape)
print(mnist.test.labels.shape, mnist.test.images.shape)
print(mnist.validation.labels.shape, mnist.validation.images.shape)
```

```
10000 55000 5000
(55000, 10) (55000, 784)
(10000, 10) (10000, 784)
(5000, 10) (5000, 784)
```

01-02 신경망 모델 구성하기

- MNIST의 손글씨는 28 X 28로 구성되어 있다.
- 784개의 특징(픽셀)로 구성되어 있음.
- 레이블은 0~9까지의 10개의 분류

Placeholder

- X : (한번 학습 시 데이터 개수, 입력층 노드수)
- Y : (한번 학습 시 데이터 개수, 출력층 노드수)

In [11]:

```
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])
```

미니배치의 이해

- 이미지를 하나씩 학습시키는 것보다 여러 개를 한꺼번에 학습시키는 쪽이 효과가 좋다.
- 많은 메모리와 높은 컴퓨터 성능이 필요하므로 일반적으로 데이터를 적당한 크기로 잘라서 학습시킨다.
  - **미니배치**라고 한다.
- tf.float32, [None, 784] => None의 자리에는 한번에 학습시킬 이미지의 개수를 지정하는 값이 들어감., 즉 배치 크기를 지정하는 자리이다.

신경망의 구성

```
* 입력층 - 784(입력, 특징 개수) ->
    256(첫번째 은닉층 뉴런) ->
    256(두번째 은닉층 뉴런)
            출력층 -> 10(결과값 0~9 분류 개수 )
```

DROPOUT 적용

- tf.nn.dropout(Layer, 비율)

In [12]:

```
keep_prob = tf.placeholder(tf.float32)

W1 = tf.Variable(tf.random_normal([784, 256], stddev=0.01))
L1 = tf.nn.relu(tf.matmul(X, W1))
L1 = tf.nn.dropout(L1, keep_prob)   # dropout

W2 = tf.Variable(tf.random_normal([256, 256], stddev=0.01))
L2 = tf.nn.relu(tf.matmul(L1, W2))
L2 = tf.nn.dropout(L2, keep_prob)   # keep_prob가 0.80이면 80% 사용 (학습시 해당 계층의 80%만 이

W3 = tf.Variable(tf.random_normal([256, 10], stddev=0.01))
model = tf.matmul(L2, W3)

print(W3)
print(model)
```

```
WARNING:tensorflow:From <ipython-input-12-13c78c1c8abd>:5: calling dropout (from ten
sorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a fut
ure version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_pro
b`.
<tf.Variable 'Variable_2:0' shape=(256, 10) dtype=float32_ref>
Tensor("MatMul_2:0", shape=(?, 10), dtype=float32)
```

비용함수, 최적화 함수 지정

- 최적화 함수 : AdamOptimizer (Adaptive Moment Estimation)은 RMSProp와 Momentum방식을 합친 것.

In [13]:

```
# old 버전 : cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=model, labels
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model, labels=Y))
optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)
```

세션 생성 및 초기화

In [14]:

```
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
```

배치 사이즈 지정

- Mini-batch 크기가 전체 트레이닝 셋 데이터 사이즈인 m과 같다면 이것은 Batch gradient descent방법
  - 데이터가 별로 없다면 batch gradient descent를 사용
- Mini-batch 크기가 1이라면 Stochastic gradient descent라고 한다.
  - **적은 메모리**로 동작가능
  - 64,128, 256, 512 사이즈 선택

In [15]:

```
batch_size = 100
total_batch = int(mnist.train.num_examples / batch_size)
```

학습

- 학습 데이터 전체를 한 바퀴를 도는 일을 **에포크(epoch)**라 한다.

In [16]:

```
# MNIST 데이터 전체를 학습하는 일을 15번 반복함.
# 학습 데이터 전체를 한 바퀴를 도는 일을 에포크(epoch)라 한다.

# 15 epochs
for epoch in range(15):
    total_cost = 0

    # total_batch : 1에폭당 학습 횟수
    for i in range(total_batch):

        # 배치 사이즈만큼 데이터 가져오기
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)

        # 입력값 : batch_xs, 출력값 : batch_ys
        # 최적화를 수행 후, 손실을 구한다.
        _, cost_val = sess.run([optimizer, cost],
                    feed_dict={X:batch_xs, Y:batch_ys,
                               keep_prob:0.8})

        # 총 손실 계산
        total_cost = total_cost + cost_val

    print("Epoch : %4d" %(epoch + 1),
                '평균 Cost = ', "{:.3f}".format(total_cost/total_batch))

print("최적화 완료!")
```

```
Epoch :    1 평균 Cost =  0.432
Epoch :    2 평균 Cost =  0.167
Epoch :    3 평균 Cost =  0.115
Epoch :    4 평균 Cost =  0.088
Epoch :    5 평균 Cost =  0.071
Epoch :    6 평균 Cost =  0.059
Epoch :    7 평균 Cost =  0.053
Epoch :    8 평균 Cost =  0.046
Epoch :    9 평균 Cost =  0.041
Epoch :   10 평균 Cost =  0.039
Epoch :   11 평균 Cost =  0.033
Epoch :   12 평균 Cost =  0.030
Epoch :   13 평균 Cost =  0.030
Epoch :   14 평균 Cost =  0.028
Epoch :   15 평균 Cost =  0.024
최적화 완료!
```

정확도 확인

- argmax : 가장 큰 값의 갖는 인덱스를 반환

In [17]:

```
# 모델의 예측값과 실제값의 비교한다.
is_correct = tf.equal(tf.argmax(model, 1), tf.argmax(Y, 1))  # 같은가?
is_correct

# 같은 것으로 판단되는 것의 평균(즉 정확도)
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
print('정확도 :', sess.run(accuracy,
                        feed_dict = {X:mnist.test.images,
                                     Y:mnist.test.labels,
                                     keep_prob: 0.8}))
```

정확도 : 0.9746

과적합을 막아주는 기법으로 가장 유명한 것 (드롭아웃)

다른 과적합 방지 방법

- 배치 정규화(Batch Normalization) : Gradient Vanishing/Gradient Exploding 이 일어나지 않도록 하기 위한 방법
  - 기존의 해결방법 : Activation 함수의 변화(ReLU등), 주의깊은 initialization, 작은 small learning rate
  - 2015년 발표 논문
  - 아래 두 함수를 이용하여 적용이 가능하다.
- tf.nn.batch_normalization
- tf.layers.batch_normalization

In [18]:

```
labels = sess.run(model,
                  feed_dict = {X:mnist.test.images,
                               Y:mnist.test.labels,
                               keep_prob: 0.8})
labels.shape
```

Out[18]:

```
(10000, 10)
```

In [19]:

```
import matplotlib.pyplot as plt
import numpy as np
# %matplotlib inline   # 필요.
```

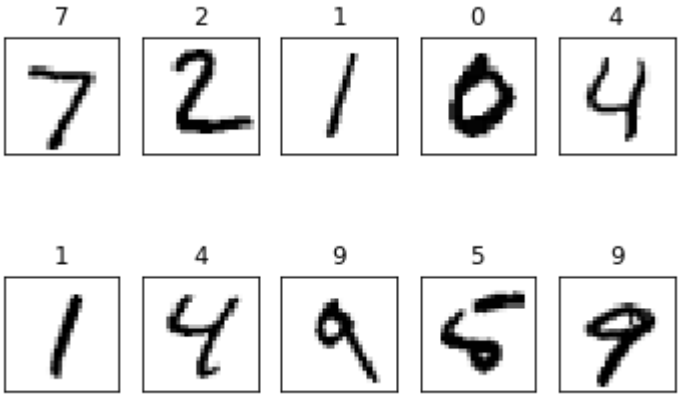
In [20]:

```
fig = plt.figure()

for i in range(10):
    subplot = fig.add_subplot(2,5, i+1)

    # 이미지 깨끗하게 출력, x,y의 눈금 출력 안함.
    subplot.set_xticks([])
    subplot.set_yticks([])

    subplot.set_title('{}'.format(np.argmax(labels[i])))
    subplot.imshow(mnist.test.images[i].reshape((28,28)),
                   cmap=plt.cm.gray_r)
plt.show()
```



더 알아보기

- 과적합을 막아주는 기법으로 가장 유명한 것이 쉽게 이해 가능한 것은 드롭아웃이다.
- **배치 정규화(Batch Normalization)**라는 기법이 많이 이용됨.
  - 과적합을 막고, 학습 속도도 향상시켜주는 장점이 있다.
  - 등장 배경은 학습 시 발산이나 소실 등을 방지하여 학습 속도를 높이기 위한 방법이다.
  - tf.nn.batch_normalization과 tf.layers.batch_normalization 함수로 쉽게 적용 가능.

실습해 보기

- dropout 비율을 조절해 본다. 0.3, 0.5, 0.7
- 각각의 값을 확인해 보자.

더 해보기

- dropout 비율을 자신이 지정해 볼 수 있도록 해 본다.

REF - 참고 링크

- Batch Normalization : http://bitly.kr/IA8kuYY (http://bitly.kr/IA8kuYY)
- Batch Normalization 논문 :https://arxiv.org/abs/1502.03167 (https://arxiv.org/abs/1502.03167)
- tf.argmax : https://www.tensorflow.org/api_docs/python/tf/argmax (https://www.tensorflow.org/api_docs/python/tf/argmax)