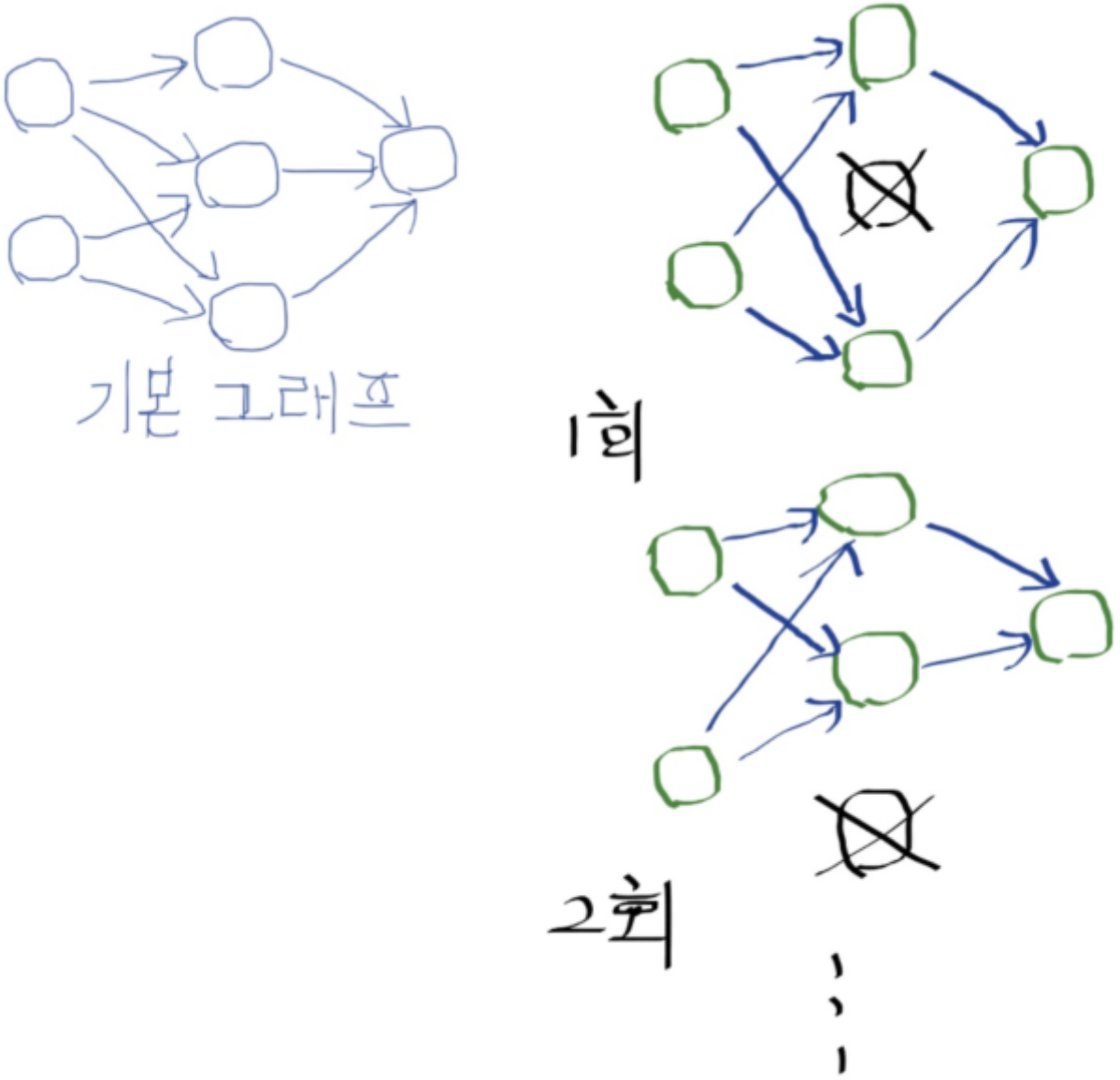
MNIST 데이터를 이용한 신경망 문제 풀이- 드롭아웃(Dropout)

01. MNIST 데이터 셋 설명

- 0 ~ 9까지의 숫자를 28 X 28 픽셀 크기의 이미지로 구성.
- 머신러닝계의 Hello World!
- 기본 내장된 mnist 모듈을 이용하여 데이터를 로드

02. 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 일부만을 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 학습되기까지의 시간은 조금 더 오래 걸리는 편이다.


기본 그래프
1회
2회

In [2]:
```
import tensorflow as tf
import numpy as np

from tensorflow.examples.tutorials.mnist import input_data
```

```
C:₩Users₩WITHJS₩Anaconda3₩lib₩site-packages₩h5py₩__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype fr
om `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

In [3]:
```
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)
```

```
WARNING:tensorflow:From <ipython-input-3-6fa84048fdd1>:1: read_data_sets (from tensorflow.contrib.learn.python.learn.datasets.mnis
t) is deprecated and will be removed in a future version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
WARNING:tensorflow:From C:₩Users₩WITHJS₩Anaconda3₩lib₩site-packages₩tensorflow₩contrib₩learn₩python₩learn₩datasets₩mnist.py:260: m
aybe_download (from tensorflow.contrib.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₩WITHJS₩Anaconda3₩lib₩site-packages₩tensorflow₩contrib₩learn₩python₩learn₩datasets₩base.py:252: _i
nternal_retry.<locals>.wrap.<locals>.wrapped_fn (from tensorflow.contrib.learn.python.learn.datasets.base) is deprecated and will
be removed in a future version.
Instructions for updating:
Please use urllib or similar directly.
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
WARNING:tensorflow:From C:₩Users₩WITHJS₩Anaconda3₩lib₩site-packages₩tensorflow₩contrib₩learn₩python₩learn₩datasets₩mnist.py:262: e
xtract_images (from tensorflow.contrib.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
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
WARNING:tensorflow:From C:₩Users₩WITHJS₩Anaconda3₩lib₩site-packages₩tensorflow₩contrib₩learn₩python₩learn₩datasets₩mnist.py:267: e
xtract_labels (from tensorflow.contrib.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₩WITHJS₩Anaconda3₩lib₩site-packages₩tensorflow₩contrib₩learn₩python₩learn₩datasets₩mnist.py:110: d
ense_to_one_hot (from tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a future version.
Instructions for updating:
Please use tf.one_hot on tensors.
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting ./mnist/data/t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting ./mnist/data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From C:₩Users₩WITHJS₩Anaconda3₩lib₩site-packages₩tensorflow₩contrib₩learn₩python₩learn₩datasets₩mnist.py:290: D
ataSet.__init__ (from tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a future version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
```

02. 신경망 모델 구성

- 28 X 28 픽셀 -> 784개의 특징
- Label은 0~9까지 10개의 분류
- 입력 X, 출력 Y
- None에는 이미지의 개수를 지정하는 값이 들어간다.

In [4]:
```
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])
```

In [5]:
```
keep_prob = tf.placeholder(tf.float32)   # 학습시에는 0.8을 넣어 드롭 아웃을 사용, 예측 시에는 1을 넣어 신경망 전체를 사용.
```

우리가 만들 신경망

- 784개의 특징(입력)
- 256 (첫번째 은닉층의 뉴런 개수)
- 256 (두번째 은닉층의 뉴런 개수)
- 10 (결과값 0-9 분류 개수)

In [6]:
```
# 784개 입력, 256개의 뉴런
# 표준편차가 0.01인 정규 분포를 가지는 임의의 뉴런을 초기화 시킨다.
W1 = tf.Variable(tf.random_normal([784, 256], stddev=0.01))

# X(입력값)에 가중치를 곱하고, 이후 ReLU 함수를 이용하여 레이어를 만든다.
L1 = tf.nn.relu(tf.matmul(X, W1))  # 데이터수 X 784 * 784 X 256 => 데이터수 X 256
L1 = tf.nn.dropout(L1, keep_prob)  # tf.nn.dropout()를 이용하여 DROPOUT 기법 적용이 가능하다.

W2 = tf.Variable(tf.random_normal([256,256], stddev=0.01))
# L1(입력값)에 가중치를 곱하고, 이후 ReLU 함수를 이용하여 레이어를 만든다.
L2 = tf.nn.relu(tf.matmul(L1, W2)) # 데이터수 X 256 * 256 X 256 => 데이터수 X 256
L2 = tf.nn.dropout(L2, keep_prob)

W3 = tf.Variable(tf.random_normal([256, 10], stddev=0.01))
model = tf.matmul(L2, W3) # 데이터수 X 256 * 256 X 10 => 데이터수 X 10
```

03. 미니배치의 평균 손실값을 구한다.

- 미니배치의 평균 손실값을 구한다.
- tf.train.AdamOptimizer 함수 이용 최적화를 수행

In [7]:
```
cost = tf.reduce_mean(
          tf.nn.softmax_cross_entropy_with_logits_v2(logits=model, labels=Y))  # 최신버전 변경 부분
optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)
```

신경망 모델 학습

In [8]:
```
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
```

04. 테스트용 데이터와 학습 데이터의 분류

- 머신러닝을 위한 데이터는 항상 학습용과 테스트용으로 분리해서 사용
- 학습 데이터는 학습을 시킬 때 사용.
- 테스트 데이터는 학습이 잘되었는지 확인하는데 사용.
- 분류하는 이유 : 별도의 테스트 데이터를 사용하는 이유는 학습 데이터로 예측을 하면 예측 정확도가 매우 높게 나오지만, 학습 데이터에 포함되지 않은 새로운 데이터를 예측할 때는 정확도가 매우 떨어지는 경우가 많기 때문.
- 이런 현상을 우리는 과적합이라 한다.

In [9]:
```
batch_size = 100
total_batch = int(mnist.train.num_examples / batch_size)
```

In [10]:
```
for epoch in range(30):
  total_cost = 0
  for i in range(total_batch):
    batch_xs, batch_ys = mnist.train.next_batch(batch_size) # 학습할 데이터를 가져온다.


    # 입력 X, 출력 Y에 각각의 데이터 넣고 실행
    _, cost_val = sess.run([optimizer,cost],
                           feed_dict={X:batch_xs, Y:batch_ys, keep_prob:0.8 })
    total_cost += cost_val
  print(batch_xs.shape, batch_ys.shape)
  print('Epoch {}, Avg. cost = {}'.format(epoch+1, total_cost/total_batch))
```

```
(100, 784) (100, 10)
Epoch 1, Avg. cost = 0.422382107661529
(100, 784) (100, 10)
Epoch 2, Avg. cost = 0.16178126987069846
(100, 784) (100, 10)
Epoch 3, Avg. cost = 0.11106827651912515
(100, 784) (100, 10)
Epoch 4, Avg. cost = 0.08679323284463449
(100, 784) (100, 10)
Epoch 5, Avg. cost = 0.07213178769939325
(100, 784) (100, 10)
Epoch 6, Avg. cost = 0.060250531117516486
(100, 784) (100, 10)
Epoch 7, Avg. cost = 0.05222327719865875
(100, 784) (100, 10)
Epoch 8, Avg. cost = 0.046215800569943066
(100, 784) (100, 10)
Epoch 9, Avg. cost = 0.04108418627218766
(100, 784) (100, 10)
Epoch 10, Avg. cost = 0.03864069186320359
(100, 784) (100, 10)
Epoch 11, Avg. cost = 0.03285767755407671
(100, 784) (100, 10)
Epoch 12, Avg. cost = 0.031716899113009935
(100, 784) (100, 10)
Epoch 13, Avg. cost = 0.0297970265251669
(100, 784) (100, 10)
```

```
Epoch 14, Avg. cost = 0.026619224710262974
(100, 784) (100, 10)
Epoch 15, Avg. cost = 0.025721379598570904
(100, 784) (100, 10)
Epoch 16, Avg. cost = 0.02225306106429674
(100, 784) (100, 10)
Epoch 17, Avg. cost = 0.02445726358691569
(100, 784) (100, 10)
Epoch 18, Avg. cost = 0.02272003052761482
(100, 784) (100, 10)
Epoch 19, Avg. cost = 0.020639550468873825
(100, 784) (100, 10)
Epoch 20, Avg. cost = 0.021150246850791685
(100, 784) (100, 10)
Epoch 21, Avg. cost = 0.021148089979859917
(100, 784) (100, 10)
Epoch 22, Avg. cost = 0.018578996791118036
(100, 784) (100, 10)
Epoch 23, Avg. cost = 0.018214320528336842
(100, 784) (100, 10)
Epoch 24, Avg. cost = 0.018434117139470553
(100, 784) (100, 10)
Epoch 25, Avg. cost = 0.017848555111612024
(100, 784) (100, 10)
Epoch 26, Avg. cost = 0.015746865395128474
(100, 784) (100, 10)
Epoch 27, Avg. cost = 0.017815726379558326
(100, 784) (100, 10)
Epoch 28, Avg. cost = 0.017824648096522486
(100, 784) (100, 10)
Epoch 29, Avg. cost = 0.01689858251320402
(100, 784) (100, 10)
Epoch 30, Avg. cost = 0.01619280996633196
```

05. 학습 후, 결과 출력

In [11]:
```
# tf.argmax(model, 1)는 1번인덱스(두번째)값 중에서 최대값을 뽑기
# tf.argmax(Y,1)는 1번 인덱스(두번째)값 중에서 최대값 뽑기
# 결과는 10개 레이블중에 확률이 가장 높은 값이 된다.
is_correct = tf.equal(tf.argmax(model, 1), tf.argmax(Y,1))
is_correct
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
accuracy
```

Out[11]:
```
<tf.Tensor 'Mean_1:0' shape=() dtype=float32>
```

In [12]:
```
print(mnist.test.images.shape)
print(mnist.test.labels.shape)
```

```
(10000, 784)
(10000, 10)
```

In [14]:
```
print('정확도', sess.run(accuracy, feed_dict={X:mnist.test.images,
                                              Y:mnist.test.labels,
                                        keep_prob:1}))
```

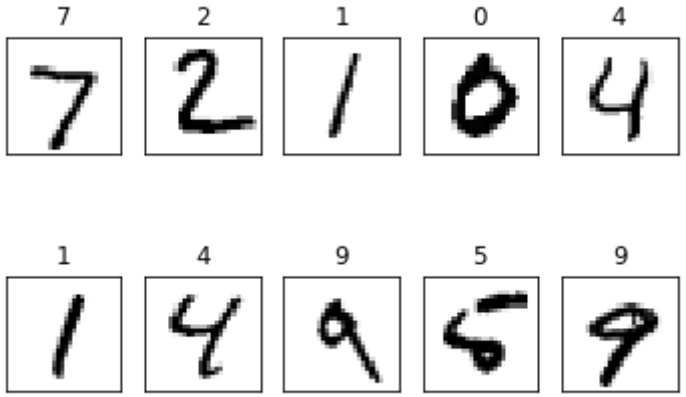
```
정확도 0.9819
```

In [16]:
```
import matplotlib.pyplot as plt
```

In [17]:
```
#########
# 결과 확인 (matplot)
######
labels = sess.run(model,
                  feed_dict={X: mnist.test.images,
                             Y: mnist.test.labels,
                             keep_prob: 1})

fig = plt.figure()
for i in range(10):
    subplot = fig.add_subplot(2, 5, i + 1)
    subplot.set_xticks([])
    subplot.set_yticks([])
    subplot.set_title('%d' % np.argmax(labels[i]))
    subplot.imshow(mnist.test.images[i].reshape((28, 28)),
                   cmap=plt.cm.gray_r)

plt.show()
```



In [ ]:
```
```