In [1]:
```
%run ../chap08/cnn_reg_model.ipynb
%run ../chap05/dataset_flowers.ipynb
%run ../chap06/dataset_office31.ipynb
```

In [2]:
```
fd = FlowersDataset([96, 96], [96, 96, 3])
od = Office31Dataset([96, 96], [96, 96, 3])
```

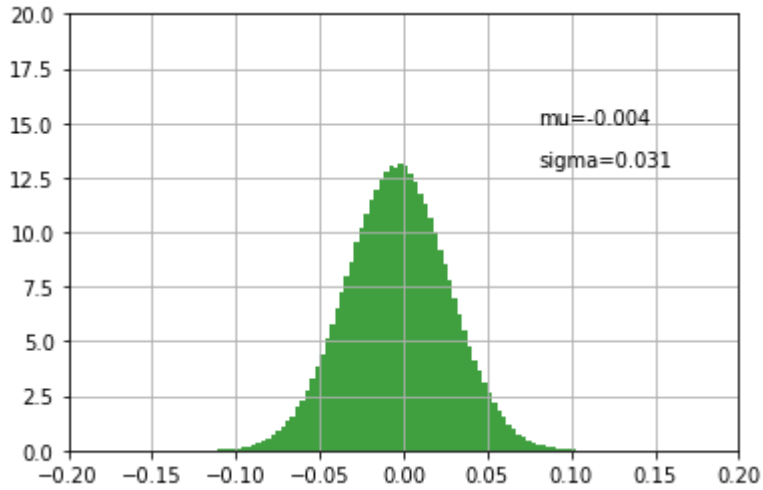
In [3]:
```
fm1 = CnnRegModel('flowers_model_1', fd, [30,10])
fm1.exec_all(epoch_count=10, report=2, show_params=True)
```

```
Model flowers_model_1 train started:
    Epoch 2: cost=1.563, accuracy=0.289/0.290 (14/14 secs)
    Epoch 4: cost=1.599, accuracy=0.246/0.210 (15/29 secs)
    Epoch 6: cost=1.587, accuracy=0.259/0.300 (18/47 secs)
    Epoch 8: cost=1.598, accuracy=0.246/0.260 (17/64 secs)
    Epoch 10: cost=1.598, accuracy=0.246/0.210 (17/81 secs)
Model flowers_model_1 train ended in 81 secs:
Model flowers_model_1 test report: accuracy = 0.233, (0 secs)

Model flowers_model_1 Visualization
```



```
추정확률분포 [18,25,18,17,23] => 추정 dandelion : 정답 dandelion => 0
추정확률분포 [18,25,18,17,23] => 추정 dandelion : 정답 rose => X
추정확률분포 [18,25,18,17,23] => 추정 dandelion : 정답 tulip => X
```

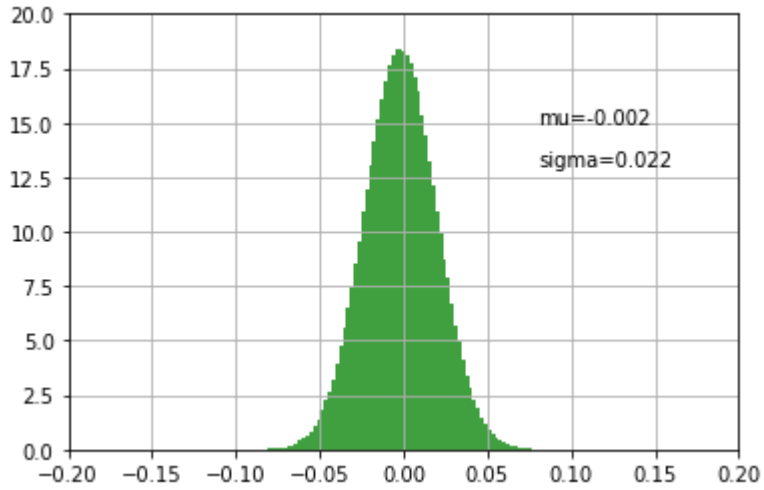



```
Near 0 parameters =  0.0%(209/829790)
```

L2

In [4]:
```
fm2 = CnnRegModel('flowers_model_2', fd, [30,10], l2_decay=0.1)
fm2.exec_all(epoch_count=10, show_cnt=0, show_params=True)
```

```
Model flowers_model_2 train ended in 567 secs:
Model flowers_model_2 test report: accuracy = 0.233, (0 secs)
```

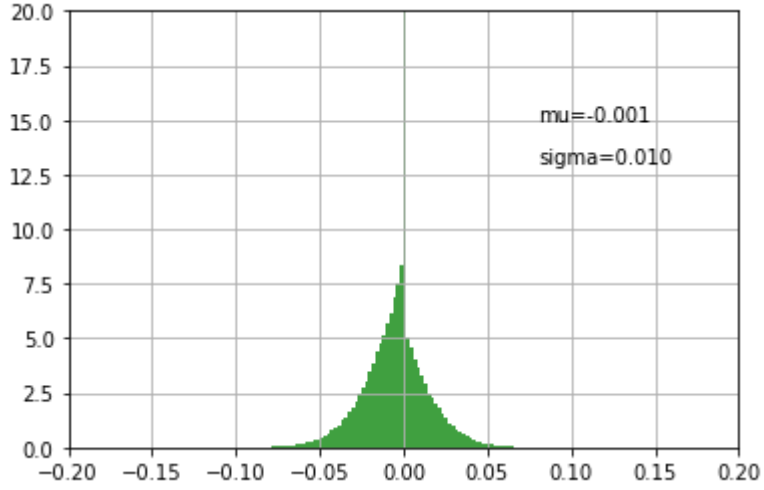



```
Near 0 parameters =  0.0%(342/829790)
```

L1

In [5]:
```
fm3 = CnnRegModel('flowers_model_3', fd, [30,10], l1_decay=0.01)
fm3.exec_all(epoch_count=10, show_cnt=0, show_params=True)
```

```
Model flowers_model_3 train ended in 585 secs:
Model flowers_model_3 test report: accuracy = 0.233, (0 secs)
```




```
Near 0 parameters = 73.7%(611917/829790)
```

In [6]:
```
cnn1 = [['conv',{'ksize':3, 'chn':6}],
        ['max',{'stride':2}],
        ['conv',{'ksize':3, 'chn':12}],
        ['max',{'stride':2}],
        ['conv',{'ksize':3, 'chn':24}],
        ['avg',{'stride':3}]]
fcnn1 = CnnRegModel('flowers_cnn_1', fd, cnn1)
fcnn1.exec_all(epoch_count=10, report=2)
```

```
Model flowers_cnn_1 train started:
    Epoch 2: cost=1.098, accuracy=0.557/0.630 (153/153 secs)
    Epoch 4: cost=0.918, accuracy=0.653/0.540 (155/308 secs)
    Epoch 6: cost=0.785, accuracy=0.703/0.620 (157/465 secs)
    Epoch 8: cost=0.693, accuracy=0.741/0.680 (158/623 secs)
    Epoch 10: cost=0.581, accuracy=0.783/0.670 (154/777 secs)
Model flowers_cnn_1 train ended in 777 secs:
Model flowers_cnn_1 test report: accuracy = 0.633, (5 secs)

Model flowers_cnn_1 Visualization
```



```
추정확률분포 [ 1, 1, 4,91, 3] => 추정 sunflower : 정답 sunflower => 0
추정확률분포 [ 1, 0, 1,98, 0] => 추정 sunflower : 정답 tulip => X
추정확률분포 [ 0, 1, 0,99, 0] => 추정 sunflower : 정답 sunflower => 0
```

Dropout

In [7]:
```
cnn2 = [['conv',{'ksize':3, 'chn':6}],
        ['max',{'stride':2}],
        ['dropout', {'keep_prob':0.6}],
        ['conv',{'ksize':3, 'chn':12}],
        ['max',{'stride':2}],
        ['dropout', {'keep_prob':0.6}],
        ['conv',{'ksize':3, 'chn':24}],
        ['avg',{'stride':3}],
        ['dropout', {'keep_prob':0.6}]]
fcnn2 = CnnRegModel('flowers_cnn_2', fd, cnn2)
fcnn2.exec_all(epoch_count=10, report=2, show_cnt=0)
```

```
Model flowers_cnn_2 train started:
    Epoch 2: cost=1.207, accuracy=0.490/0.500 (162/162 secs)
    Epoch 4: cost=1.064, accuracy=0.572/0.570 (163/325 secs)
    Epoch 6: cost=0.986, accuracy=0.621/0.520 (161/486 secs)
    Epoch 8: cost=0.918, accuracy=0.641/0.570 (172/658 secs)
    Epoch 10: cost=0.898, accuracy=0.658/0.550 (165/823 secs)
Model flowers_cnn_2 train ended in 823 secs:
Model flowers_cnn_2 test report: accuracy = 0.557, (5 secs)
```

Noise

In [8]:
```
noise_std = 0.01
cnn3 = [['noise', {'type':'normal','mean':0,'std':noise_std}],
        ['conv',{'ksize':3, 'chn':6}],
        ['max',{'stride':2}],
        ['noise', {'type':'normal','mean':0,'std':noise_std}],
        ['conv',{'ksize':3, 'chn':12}],
        ['max',{'stride':2}],
        ['noise', {'type':'normal','mean':0,'std':noise_std}],
        ['conv',{'ksize':3, 'chn':24}],
        ['avg',{'stride':3}]]
fcnn3 = CnnRegModel('flowers_cnn_3', fd, cnn3)
fcnn3.exec_all(epoch_count=10, report=2, show_cnt=0)
```

```
Model flowers_cnn_3 train started:
    Epoch 2: cost=1.099, accuracy=0.580/0.610 (180/180 secs)
    Epoch 4: cost=0.932, accuracy=0.646/0.620 (177/357 secs)
    Epoch 6: cost=0.805, accuracy=0.697/0.660 (176/533 secs)
    Epoch 8: cost=0.707, accuracy=0.740/0.680 (176/709 secs)
    Epoch 10: cost=0.590, accuracy=0.780/0.630 (181/890 secs)
Model flowers_cnn_3 train ended in 890 secs:
Model flowers_cnn_3 test report: accuracy = 0.613, (4 secs)
```

Batch Normalization

In [9]:
```
cnn4 = [['batch_normal'],
        ['conv',{'ksize':3, 'chn':6}],
        ['max',{'stride':2}],
        ['batch_normal'],
        ['conv',{'ksize':3, 'chn':12}],
        ['max',{'stride':2}],
        ['batch_normal'],
        ['conv',{'ksize':3, 'chn':24}],
        ['avg',{'stride':3}]]
fcnn4 = CnnRegModel('flowers_cnn_4', fd, cnn4)
fcnn4.exec_all(epoch_count=10, report=2, show_cnt=0)
```

```
Model flowers_cnn_4 train started:
    Epoch 2: cost=1.054, accuracy=0.584/0.410 (168/168 secs)
    Epoch 4: cost=0.878, accuracy=0.661/0.400 (128/296 secs)
    Epoch 6: cost=0.785, accuracy=0.699/0.480 (132/428 secs)
    Epoch 8: cost=0.689, accuracy=0.733/0.540 (134/562 secs)
    Epoch 10: cost=0.608, accuracy=0.774/0.540 (126/688 secs)
Model flowers_cnn_4 train ended in 688 secs:
Model flowers_cnn_4 test report: accuracy = 0.530, (4 secs)
```

In [10]:
```
ocnn1 = CnnRegModel('office31_cnn_1', od, cnn1)
ocnn2 = CnnRegModel('office31_cnn_2', od, cnn2)
ocnn3 = CnnRegModel('office31_cnn_3', od, cnn3)
ocnn4 = CnnRegModel('office31_cnn_4', od, cnn4)

ocnn1.exec_all(epoch_count=10, show_cnt=0)
ocnn2.exec_all(epoch_count=10, show_cnt=0)
ocnn3.exec_all(epoch_count=10, show_cnt=0)
ocnn4.exec_all(epoch_count=10, show_cnt=0)
```

```
Model office31_cnn_1 train ended in 525 secs:
Model office31_cnn_1 test report: accuracy = 0.874+0.558, (3 secs)

Model office31_cnn_2 train ended in 550 secs:
Model office31_cnn_2 test report: accuracy = 0.797+0.530, (3 secs)

Model office31_cnn_3 train ended in 607 secs:
Model office31_cnn_3 test report: accuracy = 0.906+0.541, (4 secs)

Model office31_cnn_4 train ended in 595 secs:
Model office31_cnn_4 test report: accuracy = 0.894+0.390, (4 secs)
```

1) Overfitting이란? 어떤 경우 Overfitting 이라 정의내릴 수 있는가?

- 전체 문제 특성을 파악하는 대신에 지엽적인 특성을 암기하여 Train 단계에서는 성능이 매우 좋으나, Test 단계에서 정확도가 매우 감소하는 현상. Train Set에 지나치게 맞추어져서, 일반화 단계(실제로 학습 모델을 적용)에서 문제가 발생한다.
- Validation Set을 구성하여 성능이 최고점에 도달한 뒤 점차적으로 감소하기 시작하는 것을 확인함으로써 과적합 여부를 판별할 수 있다.

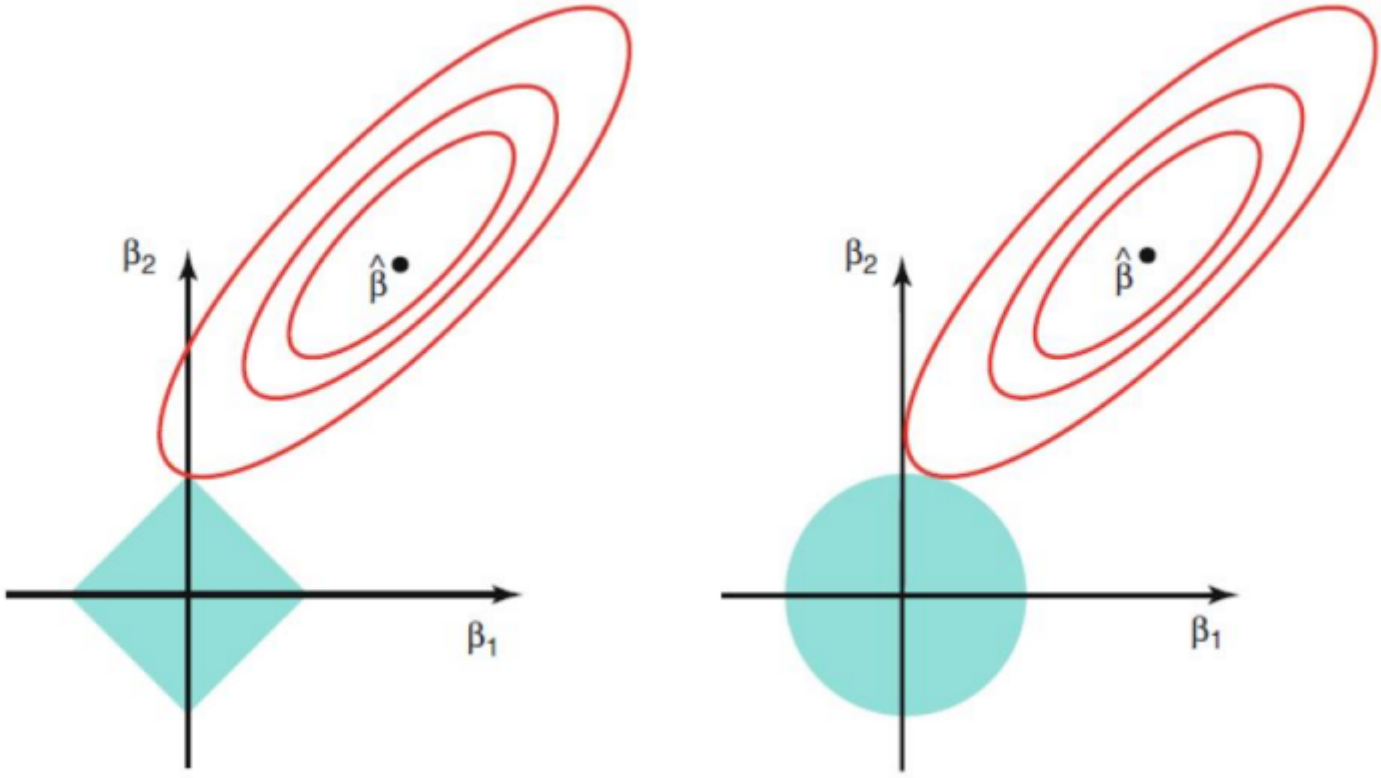
2) 정규화를 왜 사용하는가? 정규화를 사용해서 얻는 효과는?

- Overfitting은 문제 난도에 비해 데이터가 부족할 때 주로 발생한다. 데이터를 얻어 사용하는 것이 좋으나, 비용, 시간, 현실적 문제 등으로 충분한 데이터의 양을 보장하기 어렵기 때문에 overfitting을 방지하기 위해서 정규화 기법을 사용한다.
- 일부러 학습이 명확하게 이루어지지 않도록 하는 요소를 추가함으로써 과적합을 방지할 수 있다.

3) 5가지 정규화 기법(L2 Loss, L1 Loss, Dropout, Noise, Batch Normalization) ?

- L2 Loss : 기존의 손실함수 + L2 penalty (가중치 parameter의 제곱을 곱함)
- L1 Loss : 기존의 손실함수 + L1 penalty (가중치 parameter의 절댓값을 곱함)
- Dropout :
    - 계층 입력 중 일부만 이용하여 신경망을 학습시키는 규제 기법
    - 순전파 처리 : mask 처리를 통해서 배제 성분 값을 0으로 변경
    - 역전파 처리 : 같은 mask로 배제 성분 손실 기울기를 0으로 변경
    - 계산량을 줄이는 것이 아니라 과적합을 방지하는 것. 오히려 계산량은 늘어난다.
- Noise Injection : 두 계층 사이에 삽입되어 은닉 벡터에 잡음을 추가하여 전달.
    - 순전파 처리 : 학습 단계에 한해 랜덤한 잡음을 생성해 주입.
    - 입력 변이에 대한 강건한 학습 통해 평가 단계에서의 품질 제고 매번 다른 잡음 주입으로 인해 overfitting 방지.

- Batch Normalization
    - 동일한 선형 변환으로 대상값들을 평균 0,표준편차 1 분포로 만드는 처리
    - gradient vanishing, exploding 방지하기 위해서 사용(입력의 성분별 분포가 심하게 다를때 쉽게 발생
    - 모집단 전체 데이터를 대상으로 정규화하는 것이 아니라 mini-batch 데이터 각각을 정규화 대상으로 삼음.
    - 모든 데이터(mini-batch size로 쪼갠)를 학습에 충분히 이용하여 일부 성분만을 결정에 참여시키는 Dropout보다 유리.



[그림 1] 각각 L1, L2

4) 5가지 정규화 기법을 각각 적용했을 때의 실험 결과 어떻게 달라지는가?

- Baseline Model
  * 평균 0.004, 표준편차 0.031, 0 근사값 209개(0.0%)

- L2 Loss : 기존의 손실함수 + L2 penalty (가중치 parameter의 제곱을 곱함)
  * 평균 0.002, 표준편차 0.022, 0 근사값 : 342개(0.0%)
  * 전반적인 절댓값 감소로 표준편차 감소 확인

- L1 Loss : 기존의 손실함수 + L1 penalty (가중치 parameter의 절댓값을 곱함)
  * 평균 0.002, 표준편차 0.010, 0 근사값 : 약 61만개(73.7%)
  * 전반적인 절댓값 감소와 위의 그래프와 같이 0이 되는 파라미터 급증

L1, L2 규제를 통해 표준편차의 감소를 확인할 수 있었으며, 특히, L1 규제의 경우 위의 그림과 같이 0이 될 수 있음에 따라 parameter가 0이 되는 것들을 확인할 수 있었다.

- Baseline Model : 63.3%

- Dropout : 내부에 3개의 dropout layer 추가. 63.3% -> 55.7%

- Noise Injection : 내부에 3개의 Noise Injection layer 추가. 63.3 -> 61.3%

- Batch Normalization : 63.3% -> 53.0% : 내부에 3개의 batch normalization layer 추가.

일부러 학습이 명확하게 이루어지지 않도록 하는 Normalization을 통해 Dropout, Noise Injection, Batch Normalization 모두 성능이 하락하는 것을 확인할 수 있었다.

In [ ]: