ITE4053 Deep Learning course
Practice 2

2016026080 한다진

- Experiment environments

OS	Ubuntu 18.04 LTS
GPU	TITAN Xp
Language	python3
Tools	Jupyter notebook

- Time comparison (element-wise version vs. vectorized version, (m, K) = (1000,100))

element-wise version

```
In [10]: start=time.time()
         result_elementwise = train_elementwise(train_samples, test_samples, learning_rate, epochs)
         print(time.time()-start)
         print(result_elementwise)

         0.5849499702453613
         {'w': array([[0.41454156, 0.40819955]]), 'b': -0.04905599544328502, 'train_loss': 0.1466250264640928, 'test_loss': 0.
         15289553856197846, 'train_acc': 98.0, 'test_acc': 100.0}
```

vector version

```
In [11]: start=time.time()
         result_vector = train(train_samples, test_samples, learning_rate, epochs)
         print(time.time()-start)
         print(result_vector)

         0.012872934341430664
         {'w': array([[0.42484946, 0.41395356]]), 'b': -0.08788218729060701, 'train_loss': 0.13828093278404657, 'test_loss':
         0.15607500607482863, 'train_acc': 100.0, 'test_acc': 100.0}
```

Vector 연산을 하는 것이 속도면에서 50 배 가량 빠른것을 확인했습니다.
연산시간 외의 정확도 부분에서 거의 동일한 성능을 보였기에 정확하게 테스트했음을 확인했습니다.

- Empirically determined (best) hyper parameter, α

Test Parameter lr (=learning_rate)

In [16]:
```
for lr in [1e-8, 1e-4, 1e-2, 1e-1, 1e0, 1e1]:
    print("\nLearning_rate : ", lr)
    print(test_parameter(train_samples, test_samples, lr, epochs, runs))
```

```
Learning_rate :  1e-08
train_loss     0.693142
test_loss      0.693142
train_acc     96.280000
test_acc      95.070000
dtype: float64

Learning_rate :  0.0001
train_loss     0.641518
test_loss      0.641716
train_acc     96.390000
test_acc      95.800000
dtype: float64

Learning_rate :  0.01
train_loss     0.167929
test_loss      0.175276
train_acc     98.290000
test_acc      97.390000
dtype: float64

Learning_rate :  0.1
train_loss     0.075399
test_loss      0.080807
train_acc     99.470000
test_acc      98.800000
dtype: float64

Learning_rate :  1.0
train_loss      0.024045
test_loss       0.030065
train_acc     100.000000
test_acc       99.610000
dtype: float64

Learning_rate :  10.0
train_loss     0.004852
test_loss      0.031214
train_acc     99.930000
test_acc      99.500000
dtype: float64
```

Type *Markdown* and LaTeX: α^2

Runs=100 으로 설정하여 training 을 100 번 돌렸고, 그 결과값의 평균을 출력했습니다. Outlier 의 성질을 지니고 있는 케이스의 영향을 덜 받기 위해, 여러번 돌렸을때의 평균값을 사용하여 파라미터의 변화에 따른 모델의 성능변화를 확인하고자 했습니다.

구간별로 더 잘 탐색하기 위해, learning rate 값을 linear 하게 나누지 않고, log 를 씌운 값이 linear 하도록 설정하여 테스트했습니다.
1e-4 근처의 값에서 가장 높은 성능을 보일것으로 예측하였으나, 가장 높은 성능을 내는 Learning rate 는 1.0 근처의 값임을 확인했습니다. 1e-4 라는 값은 이번 practice#2 를 100 번의 iteration 만에 학습시키기엔 작은 값이라는 것을 확인했습니다. K 값을 늘린다면, 더 낮은 lr 값에 대해서도 정확도가 크게 향상될 것으로 보입니다.

- Accuracy (fill in the blanks in the tables below and add them to the report)

Test Parameter m (=train_samples)

In [14]:
```
for m in [10,100,1000]:
    print("\nTrain_samples : ", m)
    print(test_parameter(m, test_samples, learning_rate, epochs, runs))
```

```
Train_samples :  10
train_loss     0.14185
test_loss      0.22862
train_acc     98.40000
test_acc      92.31000
dtype: float64

Train_samples :  100
train_loss     0.171549
test_loss      0.174781
train_acc     98.270000
test_acc      97.610000
dtype: float64

Train_samples :  1000
train_loss     0.172270
test_loss      0.168548
train_acc     98.918000
test_acc      98.970000
dtype: float64
```

Test Parameter K (=epochs)

In [15]:
```
for K in [10,100,1000]:
    print("\nEpochs : ", K)
    print(test_parameter(train_samples, test_samples, learning_rate, K, runs))
```

```
Epochs :  10
train_loss     0.397752
test_loss      0.405442
train_acc     96.630000
test_acc      95.760000
dtype: float64

Epochs :  100
train_loss     0.172736
test_loss      0.173047
train_acc     98.280000
test_acc      97.800000
dtype: float64

Epochs :  1000
train_loss     0.074933
test_loss      0.078105
train_acc     99.560000
test_acc      98.960000
dtype: float64
```

Runs=100 으로 설정하여 training 을 100 번 돌렸고, 그 결과값의 평균을 출력했습니다. Outlier 의 성질을 지니고 있는 케이스의 영향을 덜 받기 위해, 여러번 돌렸을때의 평균값을 사용하여 파라미터의 변화에 따른 모델의 성능변화를 확인하고자 했습니다.

전반적으로 M, K 값을 늘릴수록 정확도가 향상했습니다. M 값이 늘어나면 데이터의 보편적인 특성을 파악하기 쉬워지고 K 값이 늘어날수록 데이터를 학습할 수 있는 시간이 늘어나기 때문에 정확도가 느는 것으로 판단했습니다.

n=100

lr=1e-2

K=100	m = 10	m = 100	m = 1000
train_loss	0.14185	0.171549	0.172270
test_loss	0.22862	0.174781	0.168548
train_acc	98.40000	98.270000	98.918000
test_acc	92.31000	97.610000	98.970000

m=100	K = 10	K = 100	K = 1000
train_loss	0.397752	0.172736	0.074933
test_loss	0.405442	0.173047	0.078105
train_acc	96.630000	98.280000	99.560000
test_acc	95.760000	97.800000	98.960000

- Estimated unknown function parameters W & b

Best parameters

```
In [17]: train(1000, test_samples, 1.0, epochs, runs)

Out[17]: {'w': array([[2.41652767, 2.3998954 ]]),
          'b': -1.0520468981772695,
          'train_loss': 0.02426320864479219,
          'test_loss': 0.03131281814486276,
          'train_acc': 100.0,
          'test_acc': 100.0}
```

가장 높은 성능을 보인 lr, m, K 값을 사용하여 트레이닝을 시키고 그 결과값을 받아왔습니다. w[0]값과 w[1]값이 비슷하게 뭉치는 것을 확인할 수 있었습니다.