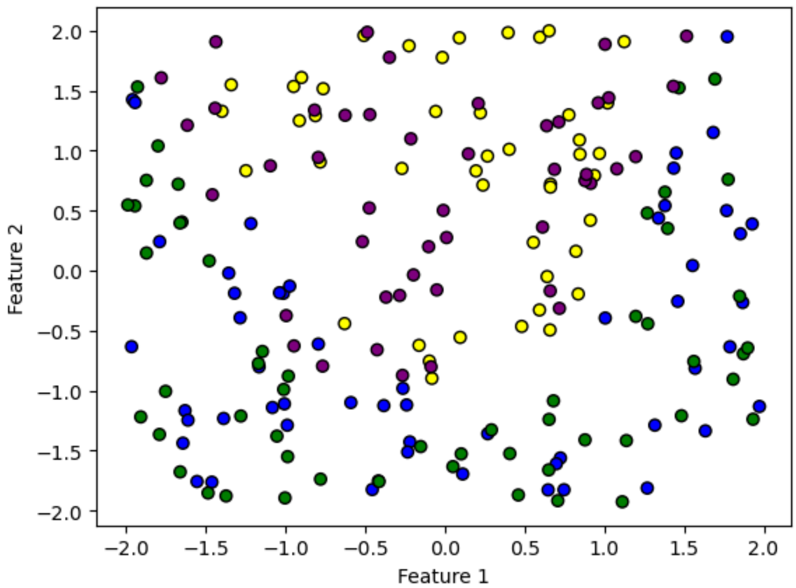
In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

데이터셋 생성

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt

# 데이터셋 생성
n_obs = 100

np.random.seed(317)
X = np.random.rand(n_obs, 2) * 4 - 2  # -2에서 2 사이의 난수 생성
y = ((X[:, 1]) + 1 > (X[:, 0] )**2).astype(int)  # 2차 함수 모양을 따라 y값 생성
# error_indices = np.random.choice(len(y), int(len(y)*0.005), replace=False)
# y[error_indices] = 1 - y[error_indices]  # 오차를 더함

# 적당한 오차를 섞어줌
X_test = np.random.rand(n_obs, 2) * 4 - 2  # -2에서 2 사이의 난수 생성
y_test = ((X_test[:, 1] + 0.2) + 1 > (X_test[:, 0] + 0.2)**2).astype(int)  # 2차 함수 모양을 따라 y값 생성
# error_indices = np.random.choice(len(y_test), int(len(y_test)*0.005), replace=False)
# y_test[error_indices] = 1 - y_test[error_indices]  # 오차를 더함

# 시각화
display_obs = 500
plt.scatter(X[:display_obs, 0], X[:display_obs, 1], c=['blue' if val == 0 else 'yellow' for val in y[:display_obs
plt.scatter(X_test[:display_obs, 0], X_test[:display_obs, 1], c=['green' if val == 0 else 'purple' for val in y_t

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

```



활성화 함수 정의

In [3]:
```
def relu(x):
    return np.maximum(x, 0)

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def tanh(x):
    return np.tanh(x)
```

NN 구성을 위한 모듈 구성

In [4]:

```python
def forward(x1, x2, y, params, **hyper_params):
    b11, b12, b13 = params['b11'], params['b12'], params['b13']
    w11, w12, w13 = params['w11'], params['w12'], params['w13']
    w21, w22, w23 = params['w21'], params['w22'], params['w23']

    b21 = params['b21']
    w31, w32, w33 = params['w31'], params['w32'], params['w33']

    # ---
    a1 = b11 + w11 * x1 + w21 * x2
    a2 = b12 + w12 * x1 + w22 * x2
    a3 = b13 + w13 * x1 + w23 * x2


    if hyper_params.get('batch_normalize', False):
        a1 = (a1 - a1.mean()) / a1.std()
        a2 = (a2 - a2.mean()) / a2.std()
        a3 = (a3 - a3.mean()) / a3.std()

    activation = hyper_params.get('activation', 'relu')
    if activation == 'relu':
        z1, z2, z3 = relu(a1), relu(a2), relu(a3)
    elif activation == 'tanh':
        z1, z2, z3 = tanh(a1), tanh(a2), tanh(a3)

#     if hyper_params.get('batch_normalize', False):
#         z1 = (z1 - z1.mean()) / z1.std()
#         z2 = (z2 - z2.mean()) / z2.std()
#         z3 = (z3 - z3.mean()) / z3.std()

    ay = b21 + w31 * z1 + w32 * z2 + w33 * z3
    yhat = sigmoid(ay)

    loss =  - (y * np.log(yhat) + (1-y) * np.log(1-yhat)).mean()

    ypred = (yhat > 0.5) * 1

#     print(params)

    return {'a1': a1, 'a2': a2, 'a3': a3, 'z1': z1, 'z2': z2, 'z3': z3, 'ay': ay, 'yhat': yhat, 'ypred': ypred, 'l
           'x1': x1, 'x2': x2}
```

In [5]:

```python
def get_gradient(X, y, params, **hyper_params):
    h = hyper_params.get('h', 1e-7)
    batch_normalize = hyper_params.get('batch_normalize', False)
    activation = hyper_params.get('activation', 'relu')

    grad = {}
    for key in params:
        params_h = params.copy()

        params_h[key] = params[key] - h
        yhat = forward(X[:, 0], X[:, 1], y, params_h, batch_normalize=batch_normalize, activation=activation)['yha
        loss1 =  - (y * np.log(yhat) + (1-y) * np.log(1-yhat)).mean()

        params_h[key] = params[key] + h
        yhat = forward(X[:, 0], X[:, 1], y, params_h, batch_normalize=batch_normalize, activation=activation)['yha
        loss2 =  - (y * np.log(yhat) + (1-y) * np.log(1-yhat)).mean()

        grad[key] = (loss2 - loss1) / (2 * h)

    return grad
```

In [6]:

```
from graphviz import Digraph

def graph(params, result, index=0):

    b11, b12, b13 = params['b11'], params['b12'], params['b13']
    w11, w12, w13 = params['w11'], params['w12'], params['w13']
    w21, w22, w23 = params['w21'], params['w22'], params['w23']

    b21 = params['b21']
    w31, w32, w33 = params['w31'], params['w32'], params['w33']

    x1, x2 = result['x1'], result['x2']
    a1, a2, a3 = result['a1'], result['a2'], result['a3']
    z1, z2, z3 = result['z1'], result['z2'], result['z3']

    ay, yhat, loss = result['ay'], result['yhat'], result['loss']

    dot = Digraph()

    fontsize = '11'
    width = '0.3'
    height = '0.3'

    dot.node('x1', 'x1={:.2f}'.format(x1[index]), shape='circle', fontsize=fontsize, width=width, height=height)
    dot.node('x2', 'x2={:.2f}'.format(x2[index]), shape='circle', fontsize=fontsize, width=width, height=height)

    dot.node('a1', 'a1={:.2f}'.format(a1[index]), shape='circle', fontsize=fontsize, width=width, height=height)
    dot.node('a2', 'a2={:.2f}'.format(a2[index]), shape='circle', fontsize=fontsize, width=width, height=height)
    dot.node('a3', 'a3={:.2f}'.format(a3[index]), shape='circle', fontsize=fontsize, width=width, height=height)

    dot.node('z1', 'z1={:.2f}'.format(z1[index]), shape='circle', fontsize=fontsize, width=width, height=height)
    dot.node('z2', 'z2={:.2f}'.format(z2[index]), shape='circle', fontsize=fontsize, width=width, height=height)
    dot.node('z3', 'z3={:.2f}'.format(z3[index]), shape='circle', fontsize=fontsize, width=width, height=height)

    dot.node('ay', 'ay={:.2f}'.format(ay[index]), shape='circle', fontsize=fontsize, width=width, height=height)
    dot.node('y', 'y={:.2f}'.format(yhat[index]), shape='circle', fontsize=fontsize, width=width, height=height)

    dot.node('loss', 'loss={:.2f}'.format(loss), shape='circle', fontsize=fontsize, width=width, height=height)

    dot.edge('x1', 'a1', label='{:.2f}'.format(w11), arrowsize='0.5')
    dot.edge('x1', 'a2', label='{:.2f}'.format(w12), arrowsize='0.5')
    dot.edge('x1', 'a3', label='{:.2f}'.format(w13), arrowsize='0.5')
    dot.edge('x2', 'a1', label='{:.2f}'.format(w21), arrowsize='0.5')
    dot.edge('x2', 'a2', label='{:.2f}'.format(w22), arrowsize='0.5')
    dot.edge('x2', 'a3', label='{:.2f}'.format(w23), arrowsize='0.5')

    dot.edge('a1', 'z1', label=f'{activation}', arrowsize='0.5')
    dot.edge('a2', 'z2', label=f'{activation}', arrowsize='0.5')
    dot.edge('a3', 'z3', label=f'{activation}', arrowsize='0.5')

    dot.edge('z1', 'ay', label='{:.2f}'.format(w31), arrowsize='0.5')
    dot.edge('z2', 'ay', label='{:.2f}'.format(w32), arrowsize='0.5')
    dot.edge('z3', 'ay', label='{:.2f}'.format(w33), arrowsize='0.5')

    dot.edge('ay', 'y', arrowsize='0.5')

    dot.edge('y', 'loss', arrowsize='0.5')

    dot.attr(rankdir='LR')

    dot

    print('b11:', round(b11, 3), '  / b12:', round(b12, 3), '  / b13:', round(b13, 3), '  / b21:', round(b21, 3))
    return dot

```

In [7]:

```
def draw_loss(loss_bucket, loss_bucket_test):
    _ = plt.plot(loss_bucket, label='train')
    _ = plt.plot(loss_bucket_test, label='test')
    _ = plt.legend()
```

In [8]:

```
import numpy as np
import matplotlib.pyplot as plt

def draw_hist(result):

    plt.figure(figsize=(10, 5))  # 그림의 크기 조절

    plt.subplot(1, 3, 1)  # 1행 3열 중 첫 번째 subplot
    plt.hist(result['z1'], bins=20, color='skyblue', edgecolor='black')
    plt.title('Histogram of Data 1')
    plt.xlabel('Value')
    plt.ylabel('Frequency')

    plt.subplot(1, 3, 2)  # 1행 3열 중 두 번째 subplot
    plt.hist(result['z2'], bins=20, color='lightgreen', edgecolor='black')
    plt.title('Histogram of Data 2')
    plt.xlabel('Value')
    plt.ylabel('Frequency')

    plt.subplot(1, 3, 3)  # 1행 3열 중 세 번째 subplot
    plt.hist(result['z3'], bins=20, color='salmon', edgecolor='black')
    plt.title('Histogram of Data 3')
    plt.xlabel('Value')
    plt.ylabel('Frequency')

    plt.tight_layout()  # subplot 간 간격 조절
    plt.show()

```

In [9]:

```

def decision_boundary(params, batch_normalize=False, index=None, title='Decision Boundary'):

    display_obs = 500
    plt.scatter(X[:display_obs, 0], X[:display_obs, 1], c=['blue' if val == 0 else 'yellow' for val in y[:display
#     plt.scatter(X_test[:display_obs, 0], X_test[:display_obs, 1], c=['green' if val == 0 else 'purple' for val in

    x1_min = min(X[:, 0].min(), X_test[:, 0].min()) - 0.1
    x1_max = max(X[:, 0].max(), X_test[:, 0].max()) + 0.1

    x2_min = min(X[:, 1].min(), X_test[:, 1].min()) - 0.1
    x2_max = max(X[:, 1].max(), X_test[:, 1].max()) + 0.1

    x1, x2 = np.meshgrid(np.arange(x1_min, x1_max, 0.02), np.arange(x2_min, x2_max, 0.02))

    Z = forward(x1, x2, np.zeros(x1.shape), params, batch_normalize=batch_normalize)['ypred']
    Z = Z.reshape(x1.shape)

    plt.contourf(x1, x2, Z, alpha=0.3)
#     plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k')

    # 관심 point
    if index is not None:
#         for idx in index:
        plt.scatter(X[index, 0], X[index, 1], c=['blue' if val == 0 else 'yellow' for val in y[index]], edgecolor

    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title(title)

```

NN 구성

초기 파라미터 지정

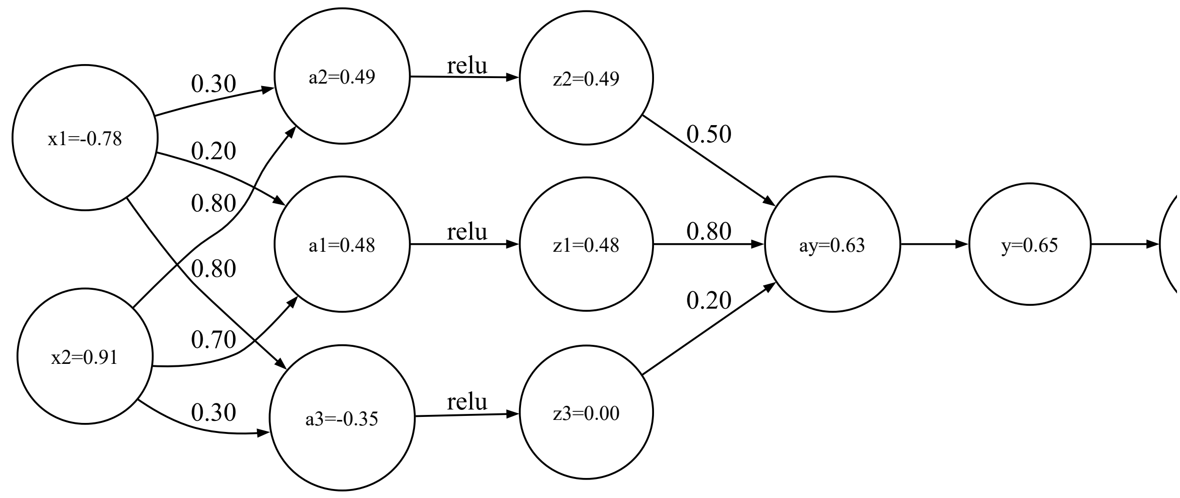
In [10]:

```
params = {'b11': 0, 'b12': 0, 'b13': 0, 'b21': 0,
          'w11': 0.2, 'w12': 0.3, 'w13': 0.8,
          'w21': 0.7, 'w22': 0.8, 'w23': 0.3,
          'w31': 0.8, 'w32': 0.5, 'w33': 0.2}

activation = 'relu'
learning_rate = 0.2

result = forward(X[:, 0], X[:, 1], y, params)
```

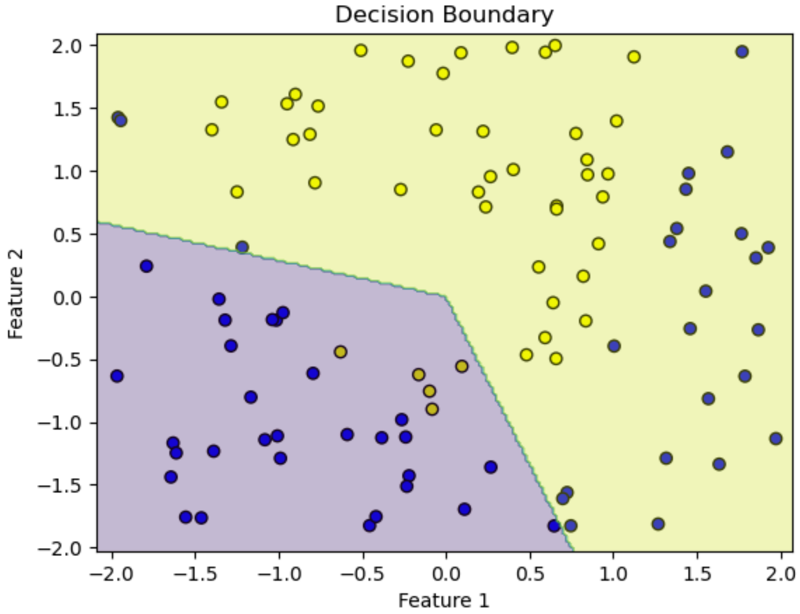
In [11]:
```
graph(params, result)
```

```
b11: 0   / b12: 0   / b13: 0   / b21: 0
```

Out[11]:



In [12]:
```
decision_boundary(params)
```



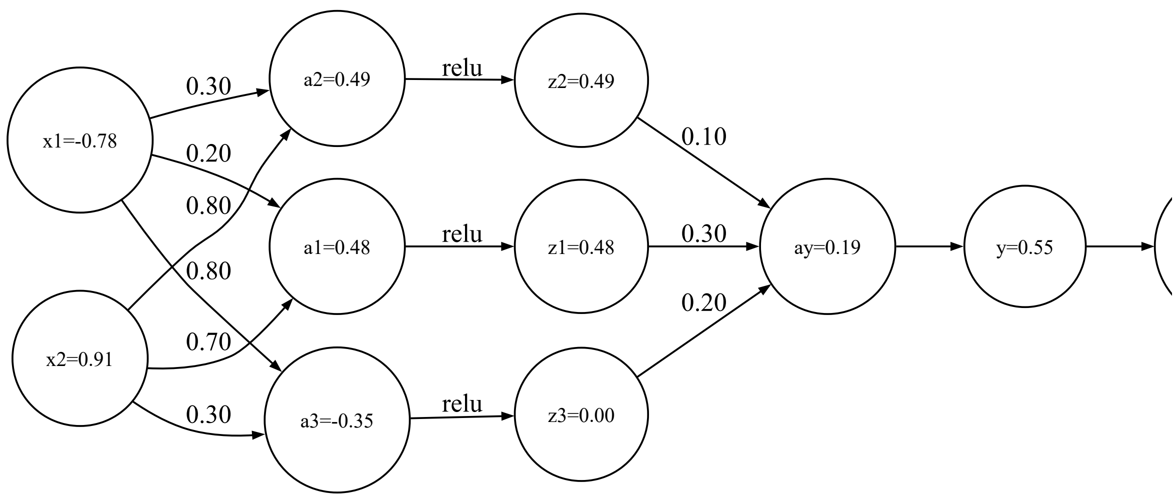
단계별 학습

In [13]:
```

params = {'b11': 0, 'b12': 0, 'b13': 0, 'b21': 0,
          'w11': 0.2, 'w12': 0.3, 'w13': 0.8,
          'w21': 0.7, 'w22': 0.8, 'w23': 0.3,
          'w31': 0.3, 'w32': 0.1, 'w33': 0.2}

hyper_params = {'activation': 'relu', 'learning_rate': 0.1, 'batch_normalize': False}

loss_bucket, loss_bucket_test = [], []
result = forward(X[:, 0], X[:, 1], y, params, activation=activation)

np.random.seed(317)
```

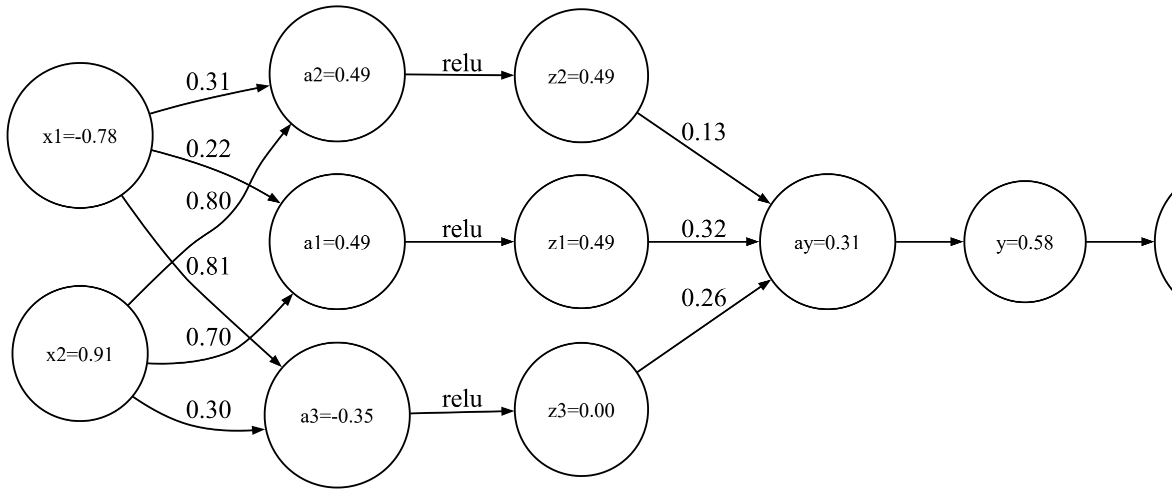
In [14]:

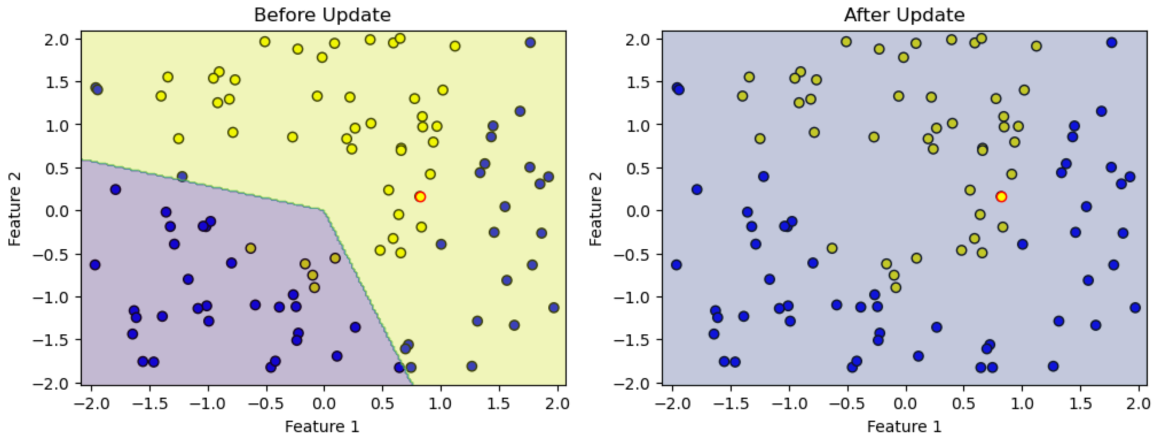
```python

# 학습을 위한 1개의 point 추출
batch_size = 1
batch_index = np.random.choice(range(len(y)),batch_size)
print('index=', batch_index, 'X =', X[batch_index, :], 'y =', y[batch_index])
print('-'*100)

print('Before Update')
display(graph(params, result))

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)

decision_boundary(params, index=batch_index, title='Before Update')

# update params
grad = get_gradient(X[batch_index, :], y[batch_index], params, **hyper_params)

for key, value in params.items():
    params[key] -= grad[key] * learning_rate

# result after backward
result = forward(X[:, 0], X[:, 1], y, params, **hyper_params)

loss_bucket.append(result['loss'])

result_test = forward(X_test[:, 0], X_test[:, 1], y_test, params, **hyper_params)
loss_bucket_test.append(result_test['loss'])

plt.subplot(1, 2, 2)
decision_boundary(params, index=batch_index, title='After Update')

print('After Update')
display(graph(params, result))


```

```
index= [74] X = [[0.8216662  0.16103194]] y = [1]
----------------------------------------------------------------------------------------------------
Before Update
b11: 0   / b12: 0   / b13: 0   / b21: 0
```




```
After Update
b11: 0.026   / b12: 0.009   / b13: 0.017   / b21: 0.087
```





시뮬레이션

In [15]:

```
def draw_result(summary):
    print(summary['desc'])
    print('loss train:', round(summary['loss_train'][-1], 4),
          ' / loss test:', round(summary['loss_test'][-1], 4))
    print('-'*100)

    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    draw_loss(summary['loss_train'], summary['loss_test'])

    plt.subplot(1, 2, 2)
    decision_boundary(summary['params'])

def draw_graph(summary):

    display(graph(summary['params'], summary['result_train']))
```

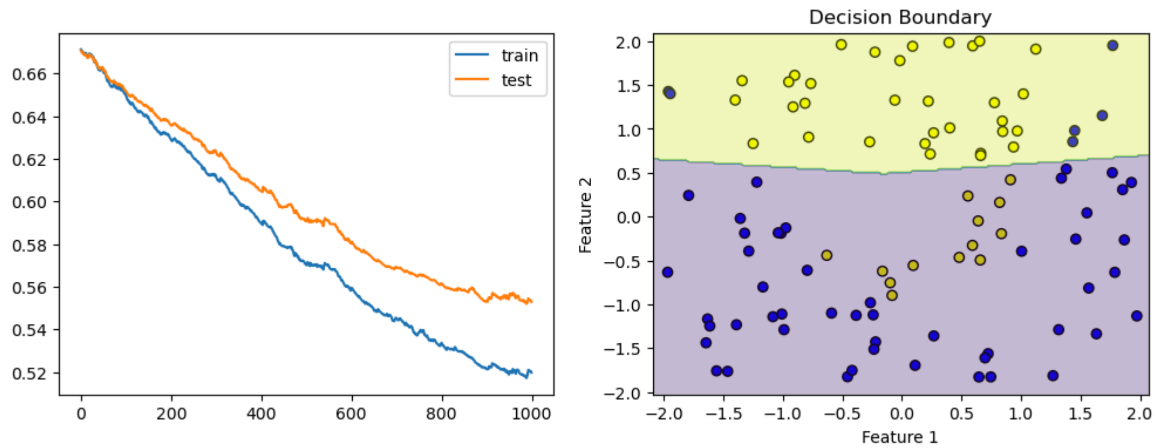
In [16]:

```
def execute_ann(activation='relu', batch_size=1, batch_normalize=False, epoch=1, learning_rate=0.01, init_params=
    if init_params is None:
        params = {'b11': 0, 'b12': 0, 'b13': 0, 'b21': 0,
                'w11': 0.2, 'w12': 0.3, 'w13': 0.8,
                'w21': 0.7, 'w22': 0.8, 'w23': 0.3,
                'w31': 0.3, 'w32': 0.1, 'w33': 0.2}
    else:
        params = init_params.copy()

    if batch_size >= len(y):
        batch_size = len(y)

    loss_bucket, loss_bucket_test = [], []

    max_iter = int(len(y) / batch_size) * epoch

    np.random.seed(317)
    for i in range(max_iter):

        if batch_size >= len(y):
            batch_index = list(range(len(y)))
        else:
            batch_index = np.random.choice(range(len(y)), batch_size)

        grad = get_gradient(X[batch_index, :], y[batch_index], params, batch_normalize=batch_normalize)

        for key, value in params.items():
            params[key] -= grad[key] * learning_rate

        result = forward(X[:, 0], X[:, 1], y, params, batch_normalize=batch_normalize, activation=activation)
        loss_bucket.append(result['loss'])

        result_test = forward(X_test[:, 0], X_test[:, 1], y_test, params, batch_normalize=batch_normalize)
        loss_bucket_test.append(result_test['loss'])

    summary = {'desc': f'iter: {max_iter} / act: {activation} / lr: {learning_rate} / btch_size: {batch_size} / epc
                'loss_train': loss_bucket.copy(),
                 'loss_test': loss_bucket_test.copy(),
                 'params': params.copy(),
                 'result_train': result.copy(),
                 'result_test': result_test.copy(),
              'batch_normalize': batch_normalize
                }
    return summary
```

In [17]:

```
ann_case_1 = execute_ann(epoch=10)
draw_result(ann_case_1)
```

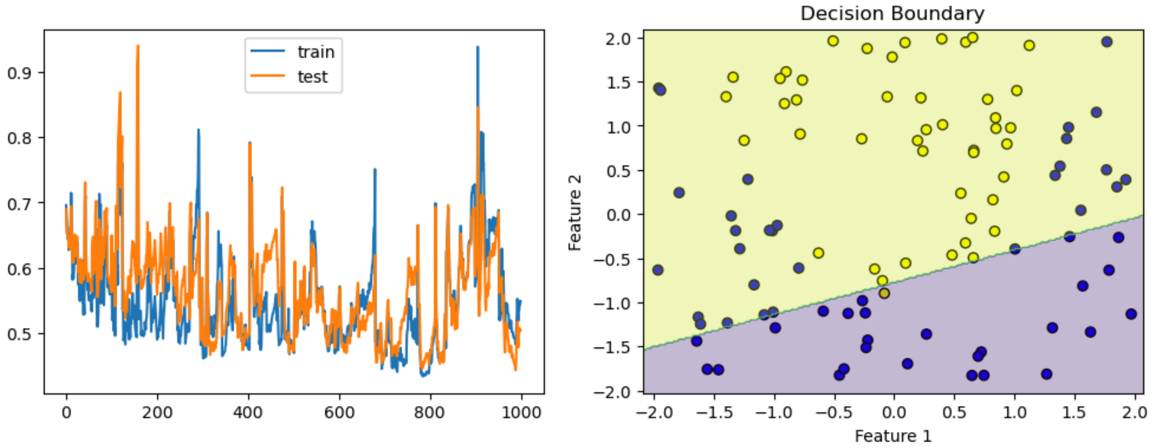
```
iter: 1000 / act: relu / lr: 0.01 / btch_size: 1 / epoch: 10 / batch_norm: False / custom_param: False
loss train: 0.5197  / loss test: 0.5529
-----------------------------------------------------------------------------------------------
```



In [18]:

```
ann_case_2 = execute_ann(epoch=10, learning_rate=0.3)
draw_result(ann_case_2)
```

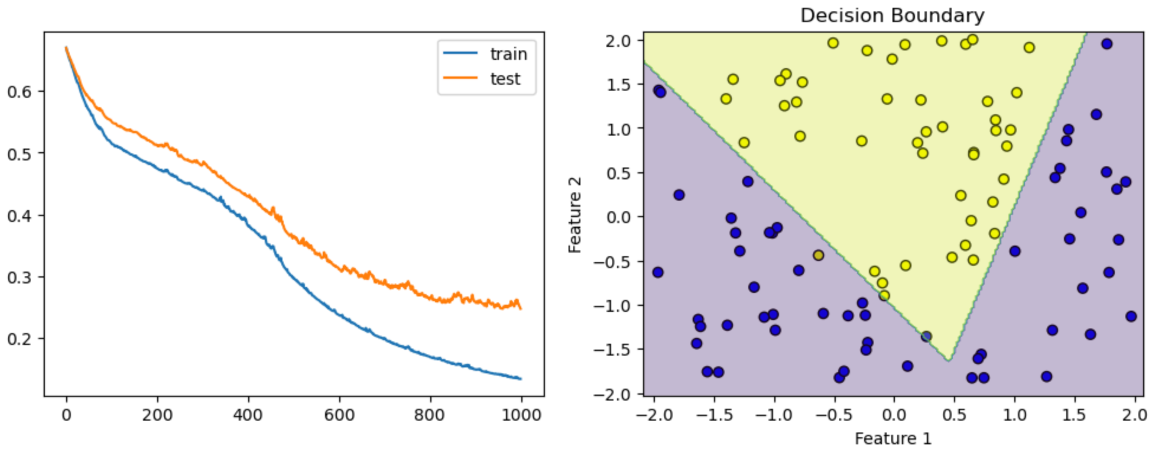
```
iter: 1000 / act: relu / lr: 0.3 / btch_size: 1 / epoch: 10 / batch_norm: False / custom_param: False
loss train: 0.5488  / loss test: 0.5039
----------------------------------------------------------------------------------------------------
```



In [19]:

```
ann_case_3 = execute_ann(epoch=200, learning_rate=0.1, batch_size=20)
draw_result(ann_case_3)
```

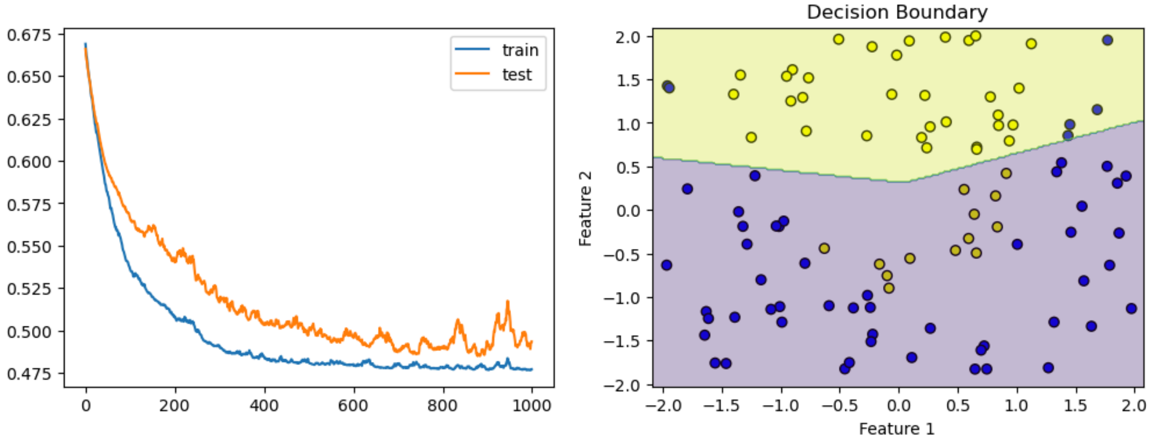
```
iter: 1000 / act: relu / lr: 0.1 / btch_size: 20 / epoch: 200 / batch_norm: False / custom_param: False
loss train: 0.1336  / loss test: 0.2472
----------------------------------------------------------------------------------------------------
```



In [20]:

```
ann_case_4 = execute_ann(epoch=200, learning_rate=0.1, batch_size=20, batch_normalize=True)
draw_result(ann_case_4)
```

```
iter: 1000 / act: relu / lr: 0.1 / btch_size: 20 / epoch: 200 / batch_norm: True / custom_param: False
loss train: 0.4771  / loss test: 0.4934
----------------------------------------------------------------------------------------------------
```

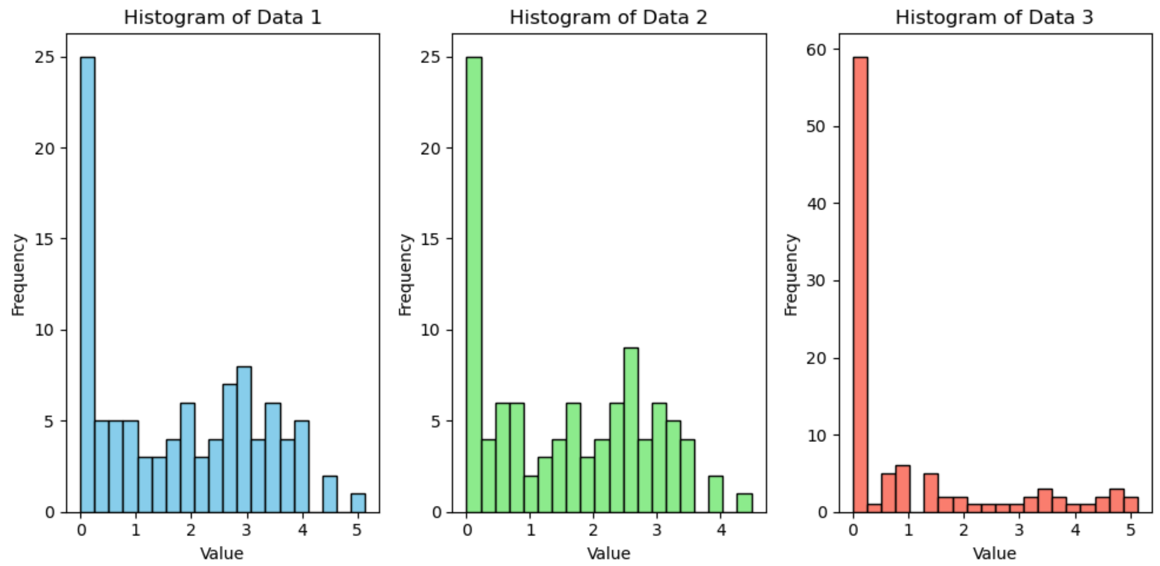


Batch Normalize 비교

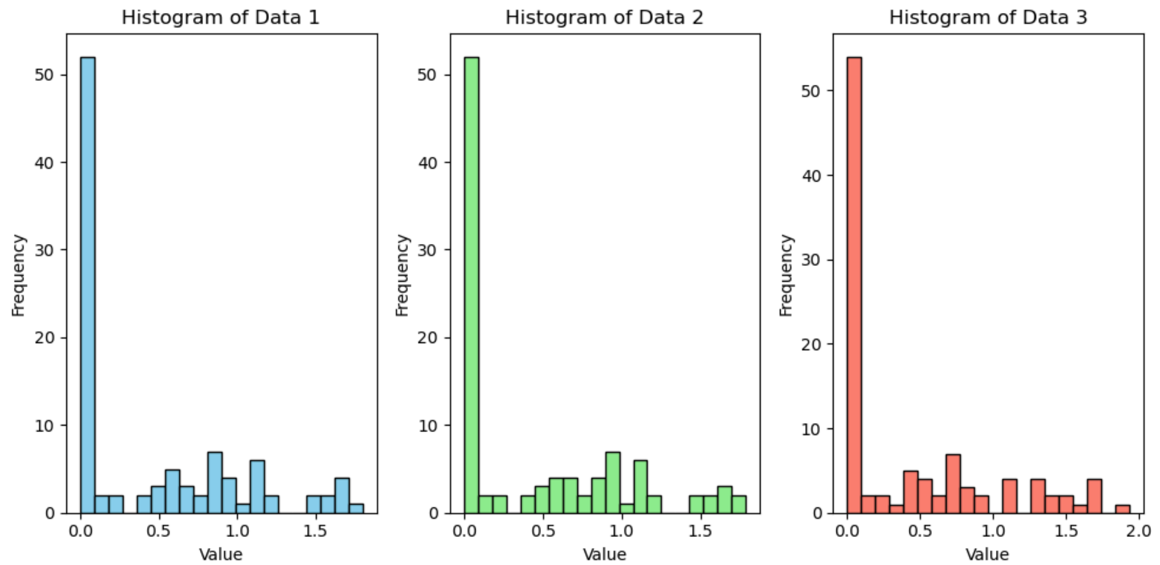
- 일반적으로는 Batch Normalize 에 의한 결과의 성능이 높음
- Batch Normalize 는 은닉층 내에서 활성화함수 이전의 결과에 대해서 Batch 단위로 Normalize를 수행하며,
- 은닉층 결과의 분포를 넓게 만들어서 최종 결과에 긍정적인 영향을 미침
- 본 예시처럼 ① 데이터가 작고 ② 배치사이즈가 작고 ③ 2개 변수간의 분포가 동일하고 ④ 모형의 구조가 간단한 경우에는 Batch Normalize 에 의해서 오히려 활성화 정도가 평균적으로 낮은 현상이 발생할 수 있음

- 아래 히스토그램은 z1, z2, z3에 대한 히스토그램으로, 일반적으로는 z3과 같은 효과가 나타남
- Dropout 등 모형이 복잡한 경우 z1,z2, z3 각각에 대한 Scale 보정효과도 있음

In [21]:
```
# Batch Normalize 미수행
draw_hist(ann_case_3['result_train'])
```



In [22]:
```
# Batch Normalize 수행
draw_hist(ann_case_4['result_train'])
```



In [ ]:
```
```

In [ ]:
```
```