In [316]:

```
import math
import csv

import matplotlib.pyplot as plt
import pandas
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split


iris = load_iris()

# 데이터셋 분리
data = iris['data']
target = iris['target']

# train_test_split 모듈을 통해 test set과 train set 분류
# test set의 비율은 default 값이 0.25이다. 이 말은 즉, 전체 데이터 셋의 25%를 테스트 set으로 지정하
# stratify의 default 값은 None인데, startify 값을 target으로 해주면 각각의 class 비율을 train/
# 즉, 한 쪽으로 쏠려서 분배되는 것을 방지해준다. 이 옵션을 설정하지 않고 classification 문제를 다뤘을 때
x_train, x_test, y_train, y_test = train_test_split(data, target, stratify=target, r

# iris_train.csv 파일로 만들기
train_df = pandas.DataFrame(x_train, y_train)
train_df.to_csv('iris_train.csv')

# iris_test.csv 파일로 만들기
test_df = pandas.DataFrame(x_test, y_test)
test_df.to_csv('iris_test.csv')
```

In [329]:

```
mport numpy
mport scipy.special

lass neuralNetwork:
    def __init__(self, inputnodes, hiddennodes, outputnodes, learningrate):
        # 입력, 은닉, 출력 계층의 노드 개수 설정
        self.innodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes

        # 가중치 행렬 wih와 who
        # 배열 내 가중치는 w_i_j로 표기, 노드 i에서 다음 계층의 노드 j로 연결됨을 의미
        # w11 w21
        # w12 w22 등
        self.wih = numpy.random.normal(0.0, pow(self.hnodes, -0.5), (self.hnodes, sel
        self.who = numpy.random.normal(0.0, pow(self.onodes, -0.5), (self.onodes, sel

        # 학습률
        self.lr = learningrate

        # 활성화 함수로는 시그모이드 함수를 이용
        self.activation_function = lambda x: scipy.special.expit(x)

        pass


    # 신경망 학습시키기
    def train(self, input_list, targets_list):
        # 입력 리스트를 2차원의 행렬로 변환
        inputs = numpy.array(input_list, ndmin=2).T
        targets = numpy.array(targets_list, ndmin=2).T

        # 은닉 계층으로 들어오는 신호를 계산
        hidden_inputs = numpy.dot(self.wih, inputs)
        # 은닉 계층에서 나가는 신호를 계산
        hidden_outputs = self.activation_function(hidden_inputs)

        # 최종 출력 계층으로 들어오는 신호를 계산
        final_inputs = numpy.dot(self.who, hidden_outputs)
        # 최종 출력 계층에서 나가는 신호를 계산
        final_outputs = self.activation_function(final_inputs)

        # 출력 계층의 오차는 (실제 값 - 계산 값)
        output_errors = targets - final_outputs
        # 은닉 계층의 오차는 가중치에 의해 나뉜 출력 계층의 오차들을 재조합해 계산
        hidden_errors = numpy.dot(self.who.T, output_errors)

        # 은닉 계층과 출력 계층 간의 가중치 업데이트
        self.who += self.lr * numpy.dot((output_errors*final_outputs*(1.0-final_output

        # 입력 계층과 은닉 계층 간의 가중치 업데이트
        self.wih += self.lr * numpy.dot((hidden_errors*hidden_outputs*(1.0-hidden_outp

        pass

    # 신경망에 질의하기
    def query(self, inputs_list):
        # 입력 리스트를 2차원 행렬로 변환
        inputs = numpy.array(inputs_list, ndmin=2).T
```

```
        # 은닉 계층으로 들어오는 신호를 계산
        hidden_inputs = numpy.dot(self.wih, inputs)
        # 은닉 계층에서 나가는 신호를 계산
        hidden_outputs = self.activation_function(hidden_inputs)

        # 최종 출력 계층으로 들어오는 신호를 계산
        final_inputs = numpy.dot(self.who, hidden_outputs)
        # 최종 출력 계층에서 들어오는 신호를 계산
        final_outputs = self.activation_function(final_inputs)

        return final_outputs
```

In [330]:

```
# 입력, 은닉, 출력 노드의 수
input_nodes = 4
hidden_nodes = 100
output_nodes = 3

# 학습률
learning_rate = 0.1

# 신경망의 인스턴스를 생성
n = neuralNetwork(input_nodes, hidden_nodes, output_nodes, learning_rate)
```

In [331]:

```
# mnist 학습 데이터인 csv 파일 리스트로 불러오기
training_data_file = open('iris_train.csv', 'r')
next(training_data_file)
training_data_list = training_data_file.readlines()
training_data_file.close()
```

In [332]:

```
# 신경망 학습시키기

# 주기(epoch)란 학습 데이터가 학습을 위해 사용되는 횟수를 의미
epoch = 1000

for e in range(epoch):
    for record in training_data_list:
        all_values = record.split(',')
        inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
        targets = numpy.zeros(output_nodes) + 0.01
        targets[int(all_values[:1][0])] = 0.99
        print(targets)
        n.train(inputs,targets)
        pass
    pass
```

```
[0.99 0.01 0.01]
[0.01 0.99 0.01]
[0.01 0.99 0.01]
[0.01 0.01 0.99]
[0.99 0.01 0.01]
[0.01 0.01 0.99]
[0.01 0.99 0.01]
[0.01 0.99 0.01]
[0.99 0.01 0.01]
[0.01 0.99 0.01]
[0.99 0.01 0.01]
[0.01 0.01 0.99]
[0.01 0.99 0.01]
[0.01 0.01 0.99]
[0.01 0.01 0.99]
[0.01 0.99 0.01]
[0.01 0.99 0.01]
[0.99 0.01 0.01]
[0.01 0.99 0.01]
```

In [333]:

```
# mnist 테스트 데이터의 csv 파일을 리스트로 불러오기
test_data_file = open('iris_test.csv', 'r')
next(test_data_file)
test_data_list = test_data_file.readlines()
test_data_file.close()
```

In [334]:

```
# 신경망 테스트하기

# 신경망의 성능의 지표가 되는 성적표를 아무 값도 가지지 않도록 초기화
scorecard = []

# 테스트 데이터 모음 내에서 모든 레코드 탐색
for record in test_data_list:
    # 레코드를 쉼표에 의해 분리
    all_values = record.split(',')
    # 정답은 첫 번째 값
    correct_label = int(all_values[0])
    # 입력 값의 범위와 값 조정
    inputs = (numpy.asfarray(all_values[1:])/ 255.0 * 0.99) + 0.01
    # 신경망에 질의
    outputs = n.query(inputs)
    # 가장 높은 값의 인덱스는 레이블의 인덱스와 일치
    label = numpy.argmax(outputs)
    # 정답 또는 오답을 리스트에 추가
    if(label == correct_label):
        # 정답인 경우 성적표에 1을 더함
        scorecard.append(1)
    else:
        # 정답이 아닌 경우 성적표에 0을 더함
        scorecard.append(0)
        pass

    print("correct_label: ", correct_label)
    print(f"label is {label}")

    pass
```

```
correct_label:  2
label is 1
correct_label:  1
label is 1
correct_label:  2
label is 1
correct_label:  1
label is 1
correct_label:  0
label is 0
correct_label:  2
label is 1
correct_label:  1
label is 1
correct_label:  0
label is 0
correct_label:  0
label is 0
correct_label:  2
label is 2
correct_label:  0
label is 0
correct_label:  2
label is 1
correct_label:  1
label is 1
correct_label:  0
label is 0
```

```
correct_label:  1
label is 1
correct_label:  0
label is 0
correct_label:  1
label is 1
correct_label:  1
label is 1
correct_label:  0
label is 0
correct_label:  0
label is 0
correct_label:  2
label is 1
correct_label:  1
label is 1
correct_label:  2
label is 1
correct_label:  1
label is 1
correct_label:  2
label is 2
correct_label:  0
label is 0
correct_label:  2
label is 1
correct_label:  2
label is 2
correct_label:  1
label is 1
correct_label:  0
label is 0
```

In [335]:

```
# 정답의 비율인 성적을 계산해 출력
print(scorecard)
scorecard_array = numpy.asarray(scorecard)
print(f'epochs = {epoch}')
print(f'hidden_nodes = {hidden_nodes}')
print(f'learning_rate = {learning_rate}')
print(f'performance = {scorecard_array.sum()/scorecard_array.size}')
print('201700949 설재혁')
```

```
[0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
1, 1, 1, 0, 1, 1, 1]
epochs = 1000
hidden_nodes = 100
learning_rate = 0.1
performance = 0.7666666666666667
201700949 설재혁
```

201700949 설재혁

In [ ]:

```

```