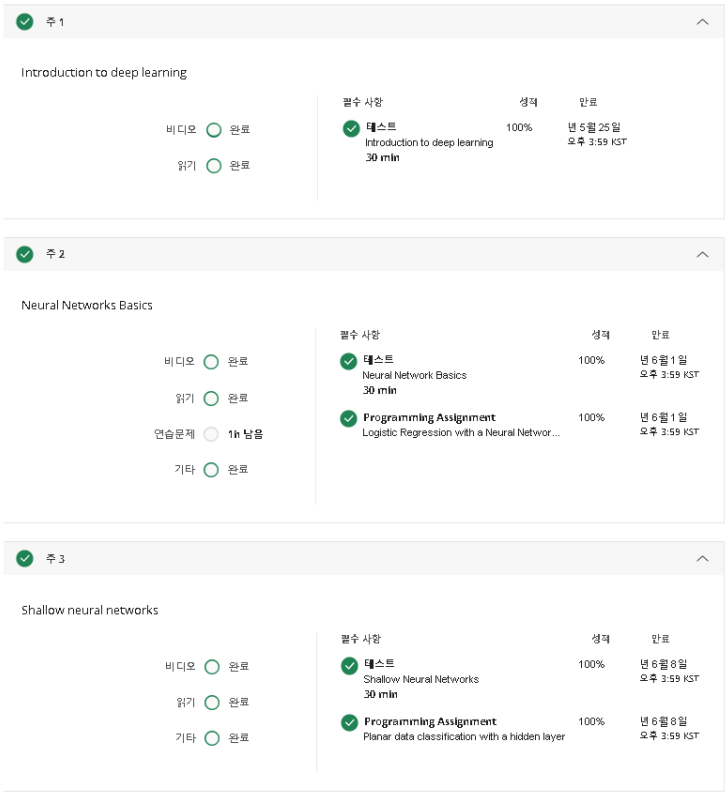
Multi-Perceptron 리포트

이름 : 박건호

학과 : 컴퓨터공학과

학번 : 201635964

- **Cosera 강의 (수강 완료)**



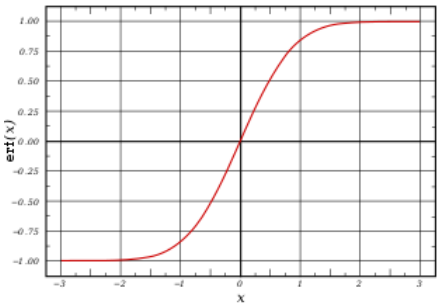
항목	현재 상태	만료	가중치	성적
Introduction to deep learning 테스트	통과	년 5월 25일 오후 3:59 KST	7%	**100%**
Neural Network Basics 테스트	통과	년 6월 1일 오후 3:59 KST	7%	**100%**
Logistic Regression with a Neural Network mindset Programming Assignment	통과	년 6월 1일 오후 3:59 KST	21%	**100%**
Shallow Neural Networks 테스트	통과	년 6월 8일 오후 3:59 KST	7%	**100%**
Planar data classification with a hidden layer Programming Assignment	통과	년 6월 8일 오후 3:59 KST	21%	**100%**

-Multi-Perceptron 구현

*사용할 함수 정의

-시그모이드 함수(활성화 함수)

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



-초기 사이즈 설정

```
def layer_sizes(X, Y):
    n_x = X.shape[0]
    n_h = 4
    n_y = Y.shape[0]
    return (n_x, n_h, n_y)
```

-Parameter 초기화

```
def initialize_parameters(n_x, n_h, n_y):
    W1 = np.random.randn(n_h, n_x) * 0.01
    b1 = np.zeros((n_h, 1))
    W2 = np.random.randn(n_y, n_h) * 0.01
    b2 = np.zeros((n_y, 1))

    assert (W1.shape == (n_h, n_x))
    assert (b1.shape == (n_h, 1))
    assert (W2.shape == (n_y, n_h))
    assert (b2.shape == (n_y, 1))

    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2}

    return parameters
```

-전방향 전파 함수(Forward Propagation)

```
def forward_propagation(X, parameters):
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]

    Z1 = np.dot(W1, X) + b1
    A1 = np.tanh(Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = sigmoid(Z2)

    cache = {"Z1": Z1,
             "A1": A1,
             "Z2": Z2,
             "A2": A2}

    return A2, cache
```

-역방향 전파 함수(Back Propagation)

```
def backward_propagation(parameters, cache, X, Y):
    m = X.shape[1]

    W1 = parameters["W1"]
    W2 = parameters["W2"]

    A1 = cache["A1"]
    A2 = cache["A2"]

    dZ2 = A2 - Y
    dW2 = np.dot(dZ2,A1.T) * (1/m)
    db2 = np.sum(dZ2, axis = 1, keepdims = True) * (1/m)
    dZ1 = np.multiply(np.dot(W2.T, dZ2), 1 - np.power(A1, 2))
    dW1 = np.dot(dZ1, X.T) * (1/m)
    db1 = np.sum(dZ1, axis = 1, keepdims = True) * (1/m)

    grads = {"dW1": dW1,
             "db1": db1,
             "dW2": dW2,
             "db2": db2}

    return grads
```

-비용함수 정의

```
def compute_cost(A2, Y, parameters):

    m = Y.shape[1]

    logprobs = np.multiply(Y, np.log(A2)) + np.multiply(1 - Y, np.log(1 - A2))
    cost = -np.sum(logprobs) * (1 / m)

    cost = float(np.squeeze(cost))

    assert(isinstance(cost, float))

    return cost
```

-Prameter 업데이트(학습률 설정: learning_rate)

```
def update_parameters(parameters, grads, learning_rate = 0.7):

    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]

    dW1 = grads["dW1"]
    db1 = grads["db1"]
    dW2 = grads["dW2"]
    db2 = grads["db2"]

    W1 = W1 - learning_rate * dW1
    b1 = b1 - learning_rate * db1
    W2 = W2 - learning_rate * dW2
    b2 = b2 - learning_rate * db2

    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2}

    return parameters
```

learning_rate : 0.6	learning_rate : 0.7
Cost after iteration 0: 2.079355 Cost after iteration 1000: 0.004660 Cost after iteration 2000: 0.002274 Cost after iteration 3000: 0.001502 Cost after iteration 4000: 0.001120 Cost after iteration 5000: 0.000893 Cost after iteration 6000: 0.000742 Cost after iteration 7000: 0.000635 Cost after iteration 8000: 0.000555 Cost after iteration 9000: 0.000493 Cost after iteration 10000: 0.000443 Cost after iteration 11000: 0.000402 Cost after iteration 12000: 0.000368 Cost after iteration 13000: 0.000340 Cost after iteration 14000: 0.000315 Cost after iteration 15000: 0.000294 Cost after iteration 16000: 0.000275 Cost after iteration 17000: 0.000259 Cost after iteration 18000: 0.000245 Cost after iteration 19000: 0.000232	Cost after iteration 0: 2.079355 Cost after iteration 1000: 0.004001 Cost after iteration 2000: 0.001952 Cost after iteration 3000: 0.001290 Cost after iteration 4000: 0.000963 Cost after iteration 5000: 0.000768 Cost after iteration 6000: 0.000638 Cost after iteration 7000: 0.000546 Cost after iteration 8000: 0.000477 Cost after iteration 9000: 0.000424 Cost after iteration 10000: 0.000381 Cost after iteration 11000: 0.000346 Cost after iteration 12000: 0.000317 Cost after iteration 13000: 0.000293 Cost after iteration 14000: 0.000272 Cost after iteration 15000: 0.000253 Cost after iteration 16000: 0.000238 Cost after iteration 17000: 0.000224 Cost after iteration 18000: 0.000211 Cost after iteration 19000: 0.000200

learning_rate : 0.8	learning_rate : 0.9
Cost after iteration 0: 2.079355 Cost after iteration 1000: 0.003533 Cost after iteration 2000: 0.001726 Cost after iteration 3000: 0.001142 Cost after iteration 4000: 0.000852 Cost after iteration 5000: 0.000680 Cost after iteration 6000: 0.000566 Cost after iteration 7000: 0.000484 Cost after iteration 8000: 0.000423 Cost after iteration 9000: 0.000376 Cost after iteration 10000: 0.000338 Cost after iteration 11000: 0.000307 Cost after iteration 12000: 0.000281 Cost after iteration 13000: 0.000260 Cost after iteration 14000: 0.000241 Cost after iteration 15000: 0.000225 Cost after iteration 16000: 0.000211 Cost after iteration 17000: 0.000198 Cost after iteration 18000: 0.000187 Cost after iteration 19000: 0.000177	Cost after iteration 0: 2.079355 Cost after iteration 1000: 0.485376 Cost after iteration 2000: 0.482797 Cost after iteration 3000: 0.480265 Cost after iteration 4000: 0.479206 Cost after iteration 5000: 0.478764 Cost after iteration 6000: 0.478496 Cost after iteration 7000: 0.478311 Cost after iteration 8000: 0.478136 Cost after iteration 9000: 0.478470 Cost after iteration 10000: 0.478146 Cost after iteration 11000: 0.478032 Cost after iteration 12000: 0.477959 Cost after iteration 13000: 0.477903 Cost after iteration 14000: 0.477859 Cost after iteration 15000: 0.477822 Cost after iteration 16000: 0.477790 Cost after iteration 17000: 0.477763 Cost after iteration 18000: 0.477740 Cost after iteration 19000: 0.477719

학습률이 0.9부터 정상 값이 나오지 않기 시작하였다.

*위의 함수들을 사용하여 모델을 구현

```
def nn_model(X, Y, n_h, num_iterations, print_cost=False):

    np.random.seed(3)
    n_x = layer_sizes(X, Y)[0]
    n_y = layer_sizes(X, Y)[2]

    parameters =initialize_parameters(n_x, n_h, n_y)

    for i in range(0, num_iterations):

        A2, cache = forward_propagation(X, parameters)

        cost = compute_cost(A2, Y, parameters)

        grads = backward_propagation(parameters, cache, X, Y)

        parameters = update_parameters(parameters, grads)

        if print_cost and i % 1000 == 0:
            print ("Cost after iteration %i: %f" %(i, cost))

    return parameters
```

```
# Forward propagation. Inputs: "X, parameters". Outputs: "A2, cache".
# Cost function. Inputs: "A2, Y, parameters". Outputs: "cost".
# Backpropagation. Inputs: "parameters, cache, X, Y". Outputs: "grads".
# Gradient descent parameter update. Inputs: "parameters, grads". Outputs: "parameters".
```

*결과 계산 함수

```
def predict(parameters, X, Y):
    A2, cache = forward_propagation(X, parameters)
    predictions = (A2 > 0.5)

    answer = []
    for i in range(8):
        for j in range(8):
            if(predictions[0][i] == Y[0][j] and predictions[1][i] == Y[1][j] and predictions[2][i] == Y[2][j]):
                answer.append(j)
    return answer
```

A2의 값이 0.5 이상이면 TRUE, 작으면 FALSE 저장후 Y(교사신호)와 비교하여 인덱스 저장

*입력값

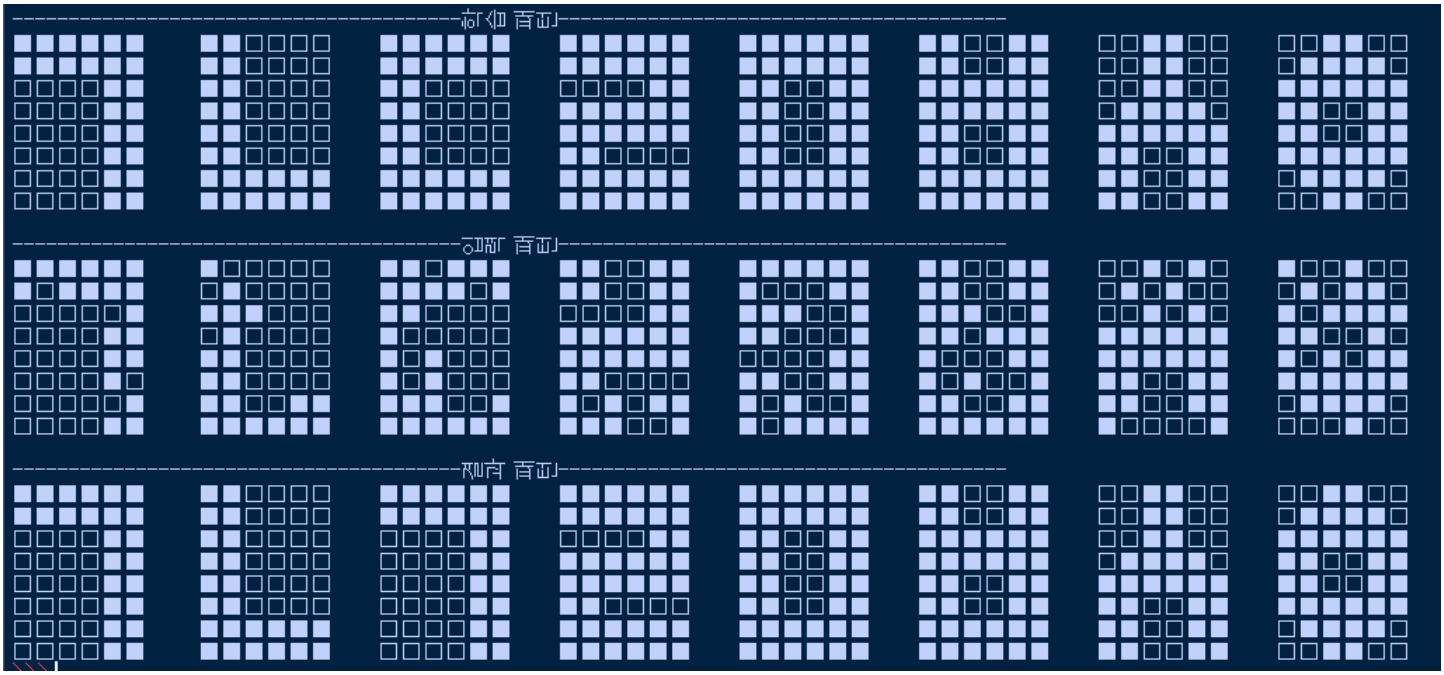
```
import numpy as np

X = np.array([[1,1,1,1,1,1,1, 1,1,1,1,1,1,1, 0,0,0,0,0,1,1, 0,0,0,0,0,1,1, 0,0,0,0,0,1,1, 0,0,0,0,0,1,1, 0,0,0,0,0,1,1, 0,0,0,0,0,1,1],
              [1,1,0,0,0,0,0, 1,1,0,0,0,0,0, 1,1,0,0,0,0,0, 1,1,0,0,0,0,0, 1,1,0,0,0,0,0, 1,1,0,0,0,0,0, 1,1,1,1,1,1,1, 1,1,1,1,1,1,1],
              [1,1,1,1,1,1,1, 1,1,1,1,1,1,1, 1,1,0,0,0,0,0, 1,1,0,0,0,0,0, 1,1,0,0,0,0,0, 1,1,0,0,0,0,0, 1,1,1,1,1,1,1, 1,1,1,1,1,1,1],
              [1,1,1,1,1,1,1, 1,1,1,1,1,1,1, 0,0,0,0,0,1,1, 1,1,1,1,1,1,1, 1,1,1,1,1,1,1, 1,1,0,0,0,0,0, 1,1,1,1,1,1,1, 1,1,1,1,1,1,1],
              [1,1,1,1,1,1,1, 1,1,1,1,1,1,1, 1,1,0,0,0,1,1, 1,1,0,0,0,1,1, 1,1,0,0,0,1,1, 1,1,0,0,0,1,1, 1,1,1,1,1,1,1, 1,1,1,1,1,1,1],
              [1,1,0,0,0,1,1, 1,1,0,0,0,1,1, 1,1,1,1,1,1,1, 1,1,1,1,1,1,1, 1,1,0,0,0,1,1, 1,1,0,0,0,1,1, 1,1,1,1,1,1,1, 1,1,1,1,1,1,1],
              [0,0,1,1,0,0,0, 0,0,1,1,0,0,0, 0,0,1,1,0,0,0, 0,1,1,1,1,1,0, 1,1,1,1,1,1,1, 1,1,0,0,0,1,1, 1,1,0,0,0,1,1, 1,1,0,0,0,1,1],
              [0,0,1,1,0,0,0, 0,1,1,1,1,1,0, 1,1,1,1,1,1,1, 1,1,0,0,1,1,1, 1,1,0,0,1,1,1, 1,1,1,1,1,1,1, 0,1,1,1,1,1,0, 0,0,1,1,0,0,0]])
X = X.T

Y = np.array([[0,0,0],
              [1,0,0],
              [0,1,0],
              [0,0,1],
              [1,1,0],
              [1,0,1],
              [0,1,1],
              [1,1,1]])
Y = Y.T

X_test = np.array([[1,1,1,1,1,1,1, 1,0,1,1,1,1,1, 0,0,0,0,0,0,1, 0,0,0,0,0,1,1, 0,0,0,0,0,1,1, 0,0,0,0,0,1,0, 0,0,0,0,0,0,1, 0,0,0,0,1,1,1],
                   [1,0,0,0,0,0,0, 0,1,0,0,0,0,0, 1,1,1,0,0,0,0, 0,1,0,0,0,0,0, 1,1,0,0,0,0,0, 1,1,0,0,0,0,0, 1,1,0,0,1,1, 1,1,1,1,1,1,1],
                   [1,1,0,1,1,1, 1,1,1,1,0,1, 1,1,0,0,0,0,0, 1,0,0,0,0,0,0, 1,0,1,0,0,0, 1,0,1,0,0,0, 1,1,1,0,0,1, 1,1,1,1,1,1,1],
                   [1,1,0,0,1,1, 1,1,0,0,1,1, 0,0,0,0,1,1, 1,1,1,1,1,1, 1,1,1,1,1,1, 1,1,0,0,0,0, 1,0,1,0,1,1, 1,1,1,0,0,1],
                   [1,1,1,1,1,1, 1,0,0,0,1,1, 1,1,1,0,0,1, 1,1,0,0,0,1, 0,0,0,0,1,1, 1,1,0,0,1,1, 1,0,1,0,0,1, 1,0,1,1,1,1],
                   [1,1,0,0,1,1, 1,1,0,0,1,1, 1,1,1,0,0,1, 1,1,0,1,1,1, 1,0,0,0,1,1, 1,0,1,0,0,1, 1,1,0,0,1,1, 1,1,1,1,1,1],
                   [0,0,1,0,1,0, 0,1,0,1,0,0, 0,0,1,0,1,0, 1,1,1,1,1,1, 1,1,1,1,1,1, 1,1,0,0,1,1, 1,1,0,0,1,1, 1,0,0,0,0,1],
                   [1,0,0,1,0,0, 0,1,0,1,1,0, 1,0,1,1,1,1, 1,1,0,0,1,0, 1,0,1,0,1,1, 1,1,1,1,1,1, 0,1,1,1,1,0, 0,0,0,1,0,0]])

X_test = X_test.T
```

8*6 사이즈의 배열 (한글을 입력하여 학습 시켰을 때에 8*6사이즈가 오류가 적다고 하여 5*5에서 8*6사이즈로 변경)

학습시키는 데이터 : X

구하고자 하는 데이터 : Y

*출력 결과



(패턴 가독성을 위해 consolas 폰트에서 돋움 폰트로 변경하였습니다. 이 때문에 한글이 뉘어져서 보이는 것 같습니다….)

가장 낮은 오차율을 보이는 learnig_rate : 0.8 로 하여 패턴 학습 후 출력시켰습니다.

-소감문

홉필드를 구현했을 때에는 그렇게 오랜 시간이 걸리지 않았습니다. 평소 자주 사용하는 C++을 사용하여 구현했고 교수님이 가르쳐 주신대로 PPT에 나와있는 순서에 맞게 구현만 하면 되었습니다. 하지만 다층 퍼셉트론을 구현했을 때에는 조금 달랐습니다. 전체적인 흐름은 이해가 되었지만 세부적인 내용을 이해하며 구현하기 개인적으로 조금 힘들었습니다. 전방향 전파 알고리즘, 역방향 전파 알고리즘, 시그모이드 함수 등 잘 이해하고 넘어가야하는 부분은이 있었습니다. 하지만 교수님이 보라고 하신 코세라 강의를 듣고 직접 코드를 보고 문제를 풀어가니 아예 못할 수준은 아니였습니다. 물론 C++로 구현을 하다가 완벽하게 구현은 하지 못했지만 코세라 강의에서 연습문제를 토대로 구현해 보니 처음 생각했던 것 만큼 어렵지 않게 구현할 수 있었습니다. 자주 써보지 않았던 파이썬으로 구현했지만 다층 퍼셉트론을 구현하기 위해서 파이썬을 다시 공부하였는데 파이썬에 대한 좋은 복습과정이 된 것 같았습니다. 학습률과 가중치를 변경하면서 값들이 매우 다양하게 바뀌며 너무 작거나 크게 설정하였을 경우 원하는대로 구현이 되지 않는 것을 보고 주먹구구식으로 값을 바꾸지 못하는 상황이라면 큰 프로젝트에 참여하였을 경우에는 정말 힘들것 같다는 생각도 들었습니다. 난이도가 있어 거의 2주에 가까운 시간에 걸쳐 구현을 하였지만 그만큼 성취감도 크고 많은 도움이 되었던 것 같습니다.

평소에 알고리즘 문제들을 풀어보며 이제 어느정도 문제에 대해서는 어렵지 않게 해낼 수 있다는 생각을 하고 있었지만 이번 기회로 아직 내가 모르는 것이 훨씬 많다는 것을 알게 되었고 자극이 많이 되었습니다. 앞으로도 수업에서 많은 것을 얻어갔으면 좋겠습니다.

C++ 로 구현한 다층 퍼셉트론 결과물(5*5배열)