인공지능 레포트

-숫자인식 프로젝트 (MLP 사용)-

지도 교수님: 공용해 교수님
제출일: 2019.06.18.

소속: 순천향대학교 의료IT공학과
성명: 20165215 위예진

-목차-

1. 서론

(a) MLP에 대한 고찰과 계획

퍼셉트론은 인간의 뉴런을 본따 만든 것으로 선형분리가 가능한 인공신경망이다. 그러나 2개 이상의 종류로 분류를 할 때에는 선형분리로는 불가능하다. 그러한 퍼셉트론의 한계를 극복하기 위해 나온 것이 MLP이다. MLP는 여러 개의 퍼셉트론을 다층으로 사용함으로써 선형 분류 함수를 여러 개 쓰는 효과를 낸다. 그러므로 비선형 분리가 가능하게 된다. MLP는 스스로 학습을 한다. 전방향으로 연산을 한 후에 출력에서의 오차를 계산하여 오차에 비례하여 가중치를 역전파 시킨다. 이렇게 계속해서 학습을 하게 되면 원하는 출력값에 가까워지게 되는 것이다.

교수님께서 지시하신대로 7*7행렬의 값을 그대로 49개의 입력으로 MLP 학습을 시켜보고, 내가 정하는 특징 추출 후의 데이터를 입력으로 MLP 학습을 시켜볼 것이다. 특징은 총 4가지로 특징추출을 하고 나니 34개의 데이터가 생겼다. 원본 행렬, 특징 추출 후 데이터에 대해 모두 학습시킨 후에 비교를 할 것이다. 49개의 입력을 그대로 넣어 학습을 진행하더라도 49차원의 데이터가 모두 입력되므로 인식률이 많이 낮지 않을 것이라고 예상한다. 또한, 특징들을 뽑아 학습시킨다면 인식률 또한 높아지고 입력의 수가 줄어들기 때문에 효율 또한 높아질 것이다. 그러나 의미있는 특징이 아니라면 오히려 학습의 방향이 이상한 곳으로 진행되어 제대로 이루어지지 않기 때문에 인식률이 낮아질 수도 있다.

(b) 변이설정과 그에 따른 결과값들의 기대

MLP는 학습을 통해 이루어지기 때문에 학습하는 방법에 따라서 성능이 결정된다. 따라서 가중치에 영향을 주는 요인들에 대해서 변이를 주었고, 그에 따라서 살펴볼 결과값들을 정하였다. 변이는 학습률, 학습횟수, 학습과 테스트 데이터의 수(비율), 중간층 2개의 각각의 노드의 개수로 두었고, 그에 대한 결과로는 모든 샘플에 대한 오차율의 평균, 학습시간, 전체 인식결과의 정확도로 설정하였다.

학습률은 목적에 얼마나 빠르게 도달하고자 하는지를 나타낸다. 이 값이 클수록 목표에 빨리 도달하지만 정확한 값에 수렴하지 못하는 단점이 있고, 값이 작을수록 목표에 느리게 도달하지만, 정확한 값에 가까이 수렴하게 된다. 따라서 학습률이 낮을수록 학습 시간이 오래 걸리기는 하겠지만, 샘플들의 오차율이 낮아지기 때문에 인식 정확도가 높아질 것으로 기대된다.

학습횟수가 많을수록 오차 수정 횟수가 많아지므로 인식 정확도가 높아질 것이며, 학습 시간 또한 오래 걸릴 것이다. 그러나 일정 학습횟수 이상이 되면 오차율이 일정한 값에 수렴하기 때문에 더 이상의 학습이 무의미할 것이라고 예상한다.

학습과 테스트 데이터의 수(비율)은 학습데이터의 수가 많을수록, 테스트 데이터의 수가 적을수록 인식률이 좋을 것으로 예측한다. 또한, 테스트 데이터에서 학습한 데이터가 차지하는 비율이 높아질수록 인식률은 높아질 것이다.

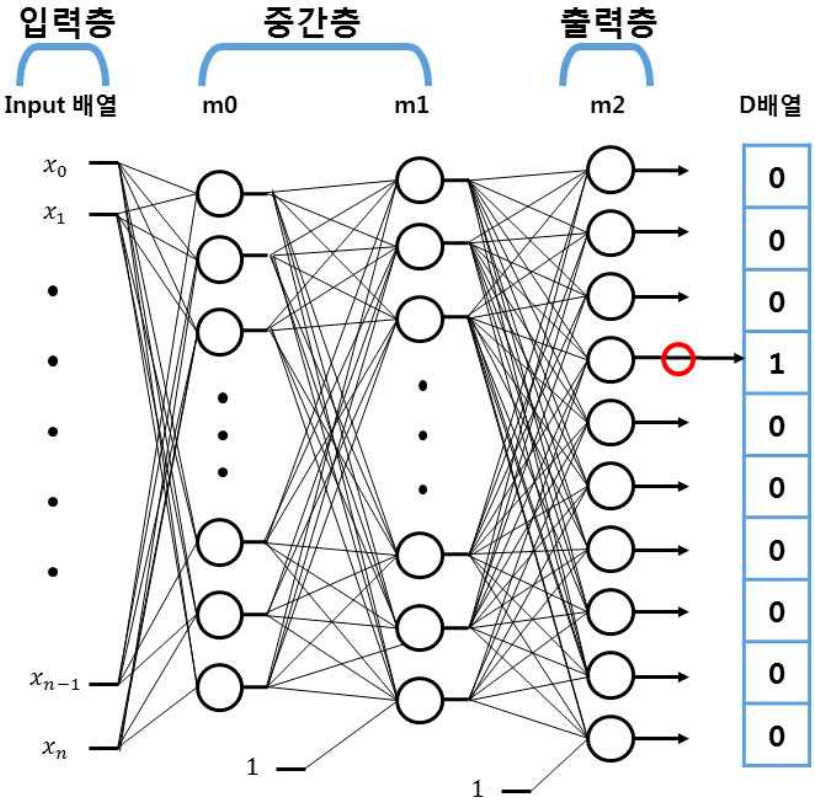
사용한 코드는 중간층의 개수가 2개로, 각각의 중간층들의 노드들의 개수 변경이 가능했다. 비율과 상관없이 중간 노드들의 수가 많을수록 인식율이 좋을 것이라고 예상한다. 또한, 중간 노드들의 개수가 많아질수록 계산량이 많아지므로 학습시간이 오래 걸릴 것이다.

전체 인식결과의 정확도는 이 모델이 얼마나 인식을 잘 하는지를 확인하기 위함이고, 전체 샘플의 오차율 평균이 작을수록 전체 정확도도 높아질 것으로 보인다. 학습시간은 정확도와 오차율이 비슷한 정도일 때 학습을 하는 데에 걸린 시간을 확인하여 어느 것이 더 효율적인지 알아보기 위함이다. 테스트 시간은 확인해 본 결과, 매번 1초를 넘지 않았기 때문에 일일이 확인하지 않았다.

2. 본론

(a) 참조한 소스코드

- 참고한 코딩 링크 : https://github.com/park-ju1008/Neural-Network
- 설명: https://juyoung-1008.tistory.com/5
- 입력층은 입력개수에 따라 개수를 계속 조절해주었고, 출력층을 10개로 바꾸었다. 입력을 0~1 사이의 실수로 바꿔주기 위해서 최대 입력값으로 나누어 주었다.
- 입력을 파일로 받아들이기 때문에 각 행렬들을 .txt파일로 변환시켜 주었다.



[그림 1. 수정한 소스코드의 MLP 학습과정 예시]

0
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 1 1 1 1 1 1
0
0 1 1 1 1 1 0
0 1 0 0 0 1 0
0 1 0 0 0 1 0
0 1 0 0 0 1 0
0 1 0 0 0 1 0
0 1 0 0 0 1 0
0 1 1 1 1 1 0

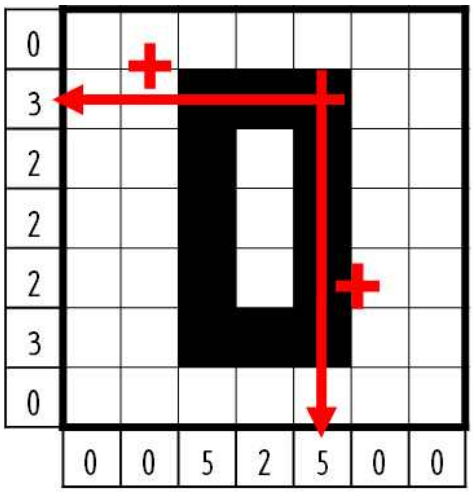
[그림 2. 변환된 데이터 예시]

(b) 선정한 특징 4가지

1차 프로젝트를 했을 때의 정했던 특징들을 모두 사용하지 않고, 프로젝션 방식과 3가지 특징을 더하여 총 특징 4가지를 선정하였다.

1) 프로젝션

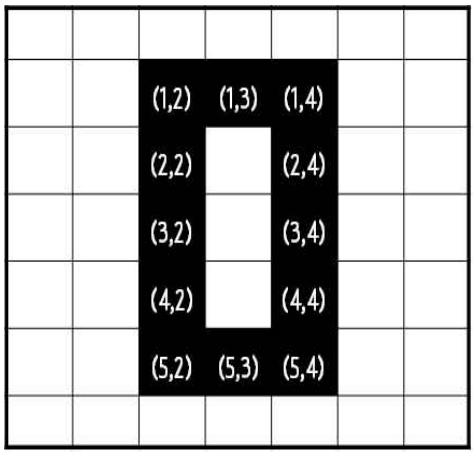
- 각각의 행과 열에 대해 값이 1인 인덱스의 개수를 더하였다.



[그림 3. 프로젝션 방식의 특징추출 예시]

2) 행, 열의 분산 값

- 값이 1인 위치의 행과 열의 분산값을 구한다.
- 예시

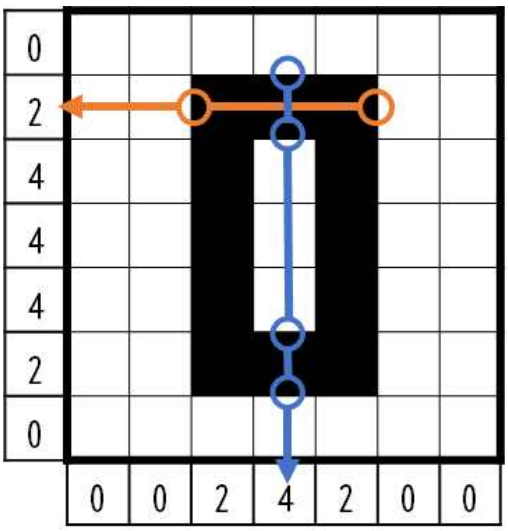


[그림 4. 행, 열의 분산값 특징추출 예시]

- 행의 분산 값 : $\sigma^2 = \frac{1}{12-1}\{(1-3)^2+(2-3)^2+(3-3)^2+(3-3)^2+(4-3)^2+(5-3)^2+\cdots\}$
- 열의 분산 값 : $\sigma^2 = \frac{1}{12-1}\{(2-3)^2+(3-3)^2+(4-3)^2+\cdots\}$

3) 교차 특징

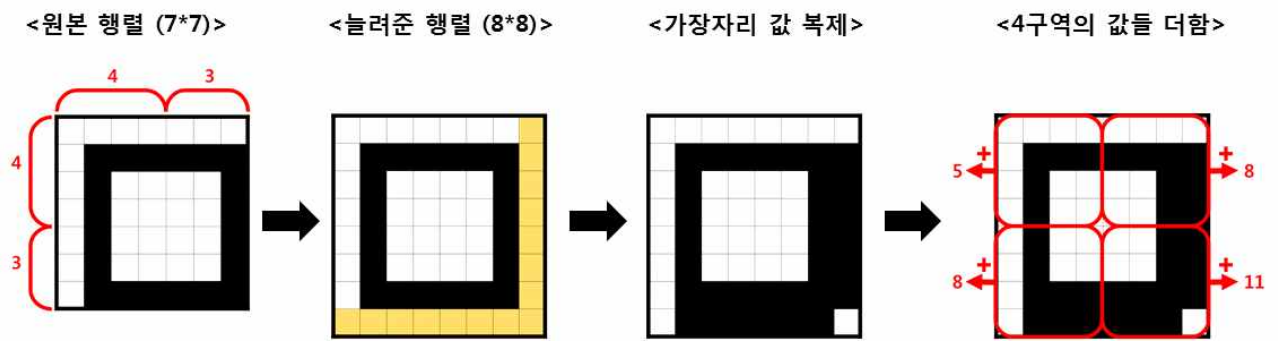
- 각각의 행과 열에서 값이 토글(0에서 1로 변화/1에서 0으로 변화)되는 횟수를 구한다
- 예시



[그림 5. 교차특징 추출 예시]

4) 망 특징

- 전체를 4칸으로 나눠서 각 칸의 1인 인덱스의 개수를 더하였다.
- 7*7 행렬이므로 똑같이 4칸으로 나뉘지 않는다. 따라서 8*8 행렬로 늘려주었다. 늘어난 칸에 대한 값들은 바깥의 값을 그대로 복제하였다.
- 예시



[그림 6. 망특징 추출 예시]

(c) 변이 변화값 설정, 비교 방식

본격적인 학습에 앞서 의미 있는 변이들의 변화값을 설정하기 위해 간단한 학습을 먼저 실행하였다. 그 결과 [표 1]과 같은 변화값들을 설정하기로 하였다.

변이들의 변화로 인해 결과값들이 어떠한 영향을 받는지 알고 싶었기 때문에, 영향을 주는 변이를 제외하고는 모두 일정하게 유지시켜야 했다. 그러므로 간이 실험을 하였을 때, 중간값으로 나온 것들로 전체 변이들의 표준으로 하여 일정하게 유지시키고, 영향을 알아보고 싶은 변이에만 변화를 주었다. 모든 변이들의 영향을 확인하고 난 후에 가장 성능이 좋았던 조건들로 학습을 시켜 최대 인식률이 몇인지 알아보았다.

학습의 종료조건은 정해둔 학습 횟수에 도달하면 멈추게 하였다. 모든 샘플에 대한 오차율의 평균, 웨이트들의 변화량이 일정값보다 작으면 멈추게 하는 방법들도 있었지만. 이러한 것들은 다른 변이들에 대한 결과값으로 설정해 두어서 종료 조건으로 선정하지 않았다.

	학습률	학습횟수	테스트 데이터의 수	학습 데이터의 수	중간 노드의 수 m0	중간노드의 수 m1
표준	0.05	200	1600 (sample 1)	1600 (sample 2)	50	100
변화	0.5, 0.1, 0.01, 0.05, 0.005	10, 20, 50, 100, 200,500, 1000	1600:1600, 1600:3200, 2600:600, 3200:3200		50:50, 50:100, 100:50, 50:150, 100:100, 50:200	

[표 1. 변이 변화와 표준값]

3. 소스코드

(a) 특징추출, 행렬을 .txt파일로 추출 (source 1)

1) 필요한 함수와 변수 등을 선언하는 헤더파일 (head.h)

```
#define SORT 10                 //숫자 0~9
#define ORDER 160               //숫자가 갖고 있는 프로토타입의 개수(10개씩*16명)
#define ROWS 7                  //행
#define COLS 7                  //열
#define PEOPLE 16               //사람 수
#define FEAT 16                 //특징벡터 행렬 수(프로젝션14 + 분산2)
#define DIV 2                   //망 특징 추출 시 열,행을 DIV로 나눔(즉 전체를 DIV*DIV개의 망으로
나눔)


int count_r[ROWS];              // 1의 개수를 행마다 더한 행렬
int count_c[COLS];              // 1의 개수를 열마다 더한 행렬
int dist[2];                    //x,y의 분산값   인덱스 0:x의 분산 1:y의 분산
int cross_r[ROWS];              //행의 0,1토글 횟수
int cross_c[COLS];              //열의 0,1토글 횟수
int mat[DIV*DIV];                           //각 망에서 1의 개수


//특징1. 프로젝션
void projection(int N[SORT][ORDER][ROWS][COLS], int sort, int order);
//특징2. x,y의 분산값
void distribute(int N[SORT][ORDER][ROWS][COLS], int sort, int order);
//특징3. 교차 특징
void cross(int N[SORT][ORDER][ROWS][COLS], int sort, int order);
//특징4. 망 특징
void matrix(int N[SORT][ORDER][ROWS][COLS], int sort, int order);

int protoData[SORT][ORDER][ROWS][COLS];                 //16_1
int num[SORT][ORDER][ROWS][COLS];                       //16_2
```

2) 학생 32명의 데이터들을 저장해놓은 소스파일 (numbers.c)

```
#include "head.h"
// 총 32명의 학생 데이터 중
// 숫자 하나당 160개(16명*10개씩)의 데이터 가짐 (행렬 하나마다)

// 16명(정리 1)
int protoData[SORT][ORDER][ROWS][COLS]={(생략)};

// 16명(정리 2)
int num[SORT][ORDER][ROWS][COLS]=={(생략)};
```

3) 특징들을 추출하는 함수들을 정의해놓은 소스파일 (feature.c)

```
#include "head.h"

//특징1. 프로젝션
void projection(int N[SORT][ORDER][ROWS][COLS], int sort, int order) {

	// 개수 초기화
	for (int i = 0; i < ROWS; i++) {
		count_r[i] = 0;
		count_c[i] = 0;
	}

	// 행마다 1의 개수를 셈
	for (int i = 0; i < ROWS; i++) {
		for (int j = 0; j < COLS; j++) {
			if (N[sort][order][i][j] == 1)
				count_r[i] += 1;
			else ;
		}
	}

	// 열마다 1의 개수를 셈
	for (int j = 0; j < COLS; j++) {
		for (int i = 0; i < ROWS; i++) {
			if (N[sort][order][i][j] == 1)
				count_c[j] += 1;
			else ;
		}
	}

}

//특징2. x,y의 분산값
void distribute(int N[SORT][ORDER][ROWS][COLS], int sort, int order) {

	double sum_x = 0, mean_x, var_x = 0;
	double sum_y = 0, mean_y, var_y = 0;
	int count = 0;

	for (int i = 0; i < ROWS; i++) {
		for (int j = 0; j < ROWS; j++) {
			if (N[sort][order][i][j] > 0) {
				count++;
				sum_x += i;
				sum_y += j;
			}
		}
	}

	mean_x = sum_x / count;
	mean_y = sum_y / count;

	for (int i = 0; i < ROWS; i++) {
		for (int j = 0; j < ROWS; j++) {
			if (N[sort][order][i][j] > 0) {
				var_x += (i - mean_x)*(i - mean_x);
				var_y += (j - mean_y)*(j - mean_y);
			}
		}
	}

	dist[0] = var_x / (count - 1);
	dist[1] = var_y / (count - 1);

}
```

```
//특징3. 교차 특징
void cross(int N[SORT][ORDER][ROWS][COLS], int sort, int order) {

	// 개수 초기화
	for (int i = 0; i < ROWS; i++) {
		cross_r[i] = 0;
		cross_c[i] = 0;
	}

	//행마다 교차특징 구함
	for (int i = 0; i < ROWS; i++) {
		for (int j = 0; j < COLS - 1; j++) {
			if (N[sort][order][i][j] != N[sort][order][i][j + 1])
				cross_r[i]++;
			else ;
		}
	}

	//열마다 교차특징 구함
	for (int j = 0; j < COLS; j++) {
		for (int i = 0; i < ROWS - 1; i++) {
			if (N[sort][order][i][j] != N[sort][order][i + 1][j])
				cross_c[j]++;
			else;
		}
	}
}

//특징4. 망 특징
void matrix(int N[SORT][ORDER][ROWS][COLS], int sort, int order) {

		int t[8][8];

		for(int i=0;i<DIV*DIV;i++)			//망특징저장 행렬 초기화
			mat[i]=0;

		for (int i = 0; i < ROWS; i++) {			//행렬t에 원래 행렬 복사
			for (int j = 0; j < COLS; j++) {
				t[i][j] = N[sort][order][i][j];
			}
		}
		for (int i = 0; i < 8; i++) {				//가장자리 값으로 복제
			t[ROWS][i] = N[sort][order][ROWS][i];
			t[i][COLS] = N[sort][order][i][COLS];
		}

		for (int i = 0; i < 8; i++) {
			for (int j = 0; j < 8; j++) {
				if (t[i][j] > 0) {
					if (i < 4 && j < 4)		//제2사분면
						mat[0]++;
					else if (i < 4 && j>3)		//제1사분면
						mat[1]++;
					else if (i > 3 && j < 4)	//제3사분면
						mat[2]++;
					else
						mat[3]++;		//제4사분면
				}
				else;
			}
		}
}
```

4) 원본 데이터의 행렬, 특징추출 후의 데이터 행렬을 .txt파일로 추출하는 메인소스 (main.c)

```
#include <stdio.h>
#include <math.h>
#include "head.h"

int main(void) {
	//코딩의 행렬을 파일로 뽑아내기-데이터 그대로
	FILE *fp = NULL;
	fp = fopen("sample.txt", "w");
	if (fp == NULL) {
	printf("파일 열기 실패\n");
	return 1;
	}
	for (int p = 0; p < PEOPLE; p++) {			//인원수만큼 돌림
		for (int n = 0; n < SORT; n++) {			//숫자
			for (int m = p * 10; m < p * 10 + SORT; m++) {		//숫자당 데이터 10개
				fprintf(fp, "%d\n", n);
				for (int i = 0; i < ROWS; i++) {
					for (int j = 0; j < COLS; j++) {
						fprintf(fp, "%d ", protoData[n][m][i][j]);		//원하는
데이터배열 지정
					}
					fputc('\n', fp);
				}
			}
		}
	}

	fclose(fp);


	//코딩의 행렬을 파일로 뽑아내기-특징추출 후
	fp = fopen("feature.txt", "w");
	if (fp == NULL) {
		printf("파일 열기 실패\n");
		return 1;
	}
	for (int p = 0; p < PEOPLE; p++) {			//인원수만큼 돌림
		for (int n = 0; n < SORT; n++) {			//숫자
			for (int m = p * 10; m < p * 10 + SORT; m++) {		//숫자당 데이터 10개
				fprintf(fp, "%d\n", n);

				//원하는 데이터 선택함
				projection(num, n, m);
				distribute(num, n, m);
				cross(num, n, m);
				matrix(num, n, m);

				for(int i=0;i<ROWS;i++)				//프로젝션(행)
					fprintf(fp, "%d ", count_r[i]);
				fputc('\n', fp);
				for (int i = 0; i<COLS; i++)			//프로젝션(열)
					fprintf(fp, "%d ", count_c[i]);
				fputc('\n', fp);
				for (int i = 0; i<2; i++)			//분산 x,y
					fprintf(fp, "%d ", dist[i]);
				fputc('\n', fp);
				for (int i = 0; i<ROWS; i++)			//교차특징(행)
					fprintf(fp, "%d ", cross_r[i]);
				fputc('\n', fp);
				for (int i = 0; i<COLS; i++)			//교차특징(열)
					fprintf(fp, "%d ", cross_c[i]);
				fputc('\n', fp);
				for (int i = 0; i<DIV*DIV; i++)			//망특징
					fprintf(fp, "%d ", mat[i]);
				fputc('\n', fp);
			}
		}
	}

	fclose(fp);
	return 0;
}
```

(b) MLP 인식 (source 2)

1) 필요한 함수와 변수 등을 선언하는 헤더파일 (AI.h)

```
/*
 * 참고한 기존코드 *
 * 설명 : "https://juyoung-1008.tistory.com/5"
 *                                                    소스코드                                                    :
"https://github.com/park-ju1008/Neural-Network/tree/91343ec215ba3ec1e9c13f594bdfaf3f66eb9092"
 * 사용하려는 목적에 맞게 적절하게 코드와 주석을 수정, 첨가하였음
 */

#ifndef _AI_H_
#define _AI_H_

#define NLayer 3                    //전체 레이어수
#define MLayerSize 1000                     //은닉층1/2 의 개수보다 큰 임의의 수
#define m0 50               //은닉층1 개수
#define m1 50               //은닉층2 개수
#define m2 10               //출력층 개수
#define N 34+1              //총 입력 시그날의 수(데이터 그대로:49  특징추출 후:34)
#define tr_N 1000           //설정한 학습 횟수
#define train_threshold 0.005               //목표 오차율
#define N_tr_examples 3200                  //학습 데이터의 수
#define N_te_examples 3200                  //테스트 데이터의 수
int M[NLayer];
double trainTime;                   //학습 시간
double testTime;            //테스트 시간
int correct[m2];            //숫자 m2의 맞은 개수
int input_max;                      //입력값의 최대값
double avg_sq_error;        //오차율


void data_Onmemory(); //트레이닝 데이터 및 테스트 데이터 메모리에 저장하는 함수
void weight_init(); // 초기 가중치 설정 함수
void forward_compute();             //포워드 계산 함수
void backward_compute();            //BP연산함수(역전파)
void weight_update();               //가중치 수정하는 함수
int training(); //트레이닝 데이터로 훈련하는 함수
double test(); //테스트 데이터로 정답률 출력 함수
int correctNum(); // 출력 값중에 가장 큰 값을 가지는 배열의 index 출력(index=인식된 숫자)
#endif
```

2) 가중치 초기화, 역전파, 트레이닝, 테스트 등을 구현하는 함수들을 정의해놓은 소스파일 (AI.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <memory.h>
#include "AI.h"

int M[NLayer] = { m0, m1, m2 };                                 //층마다의 노드 수 저장
int trainData[N_tr_examples][N - 1] = { 0, };                   //학습데이터들 받아와서 저장
int d_tr[N_tr_examples][m2] = { 0, };                                   //N_tr_exples번째의   학습데이터가
인식되어야할 숫자(즉, 정답명시-정답인 인덱스에 1들어감)
int testData[N_te_examples][N - 1] = { 0, };                    //테스트 데이터들 받아와서 저장
int d_te[N_te_examples][m2] = { 0, };                                   ////N_te_exples번째의   테스트데이
터가 인식되어야할 숫자(즉, 정답명시)

int input[N];                           //입력 N개 저장함
int D[m2];                                      //정답 넣어줌


double s[NLayer][MLayerSize];                   //NLayer층의 MLayerSize번째 노드의 값(들어온 값에 웨이트
값을 곱한 결과)
double f[NLayer][MLayerSize];                   //s값을 시그노이드 함수에 통과시킨 후의 값
double delta[NLayer][MLayerSize];
double W[NLayer][MLayerSize][MLayerSize];                       //웨이트 값
double c = 0.5;             //학습률

void data_Onmemory() {
        FILE *trainfp;
        FILE *testfp;
        int i = 0, j, ans;                      //ans:기대되는 출력값(숫자)
        input_max = 0;

        trainfp = fopen("train.txt", "rt");
        testfp = fopen("test.txt", "rt");
        if (trainfp != NULL) {

                while (!feof(trainfp)) {                        //파일에 있는 것들을 모두 불러와서 저장시킴
                        fscanf(trainfp, "%d ", &ans);
                        d_tr[i][ans] = 1;                               //d_tr의 i번째 데이터의 숫자는 ans
임(맞는 숫자 인덱스 값에 1넣어줌)
                        for (j = 0; j < N - 1; j++) {
                                fscanf(trainfp, "%d ", &trainData[i][j]);
                                if (input_max < trainData[i][j])
                                        input_max = trainData[i][j];                    //가장 큰 입력값
찾음
                                else ;
                        }
                        i++;
                }
                fclose(trainfp);

        }
        if (testfp != NULL) {
                while (!feof(testfp)) {
                        for (i = 0; i < N_te_examples; i++) {
                                fscanf(testfp, "%d", &ans);
                                d_te[i][ans] = 1;
                                for (j = 0; j < N - 1; j++) {
                                        fscanf(testfp, "%d ", &testData[i][j]);
                                }

                        }
```

```
                }
                fclose(testfp);
        }
}


void weight_init() {                                    //모든 노드의 weight parameter들 초기화
        int i, j, k, r, pre_layer;
        srand(time(NULL));
        for (i = 0; i < NLayer; i++) {
                if (i == 0) {
                        pre_layer = N;
                        for (j = 0; j < M[i]; j++) {
                                for (k = 0; k < pre_layer; k++) {
                                        r = (double)(rand());
                //초기에 rand()함수 돌려서 랜덤으로 나온 값으로 가중치 고정시킴(rand()함수는 프로그램
한 번 실행하면 바뀌지X)
                                        W[i][j][k] = (r / (double)RAND_MAX) - 0.5; //입력 값이 커
w값 작게 설정 -->
                                }
                        }
                }
                else {
                        pre_layer = M[i - 1] + 1;
                        for (j = 0; j < M[i]; j++) {
                                for (k = 0; k < pre_layer; k++) {
                                        r = (double)(rand());
                                        W[i][j][k] = (r / (double)RAND_MAX) - 0.5;
                                }
                        }
                }
        }
}

void forward_compute() {
        int i, j, layer;
        //0층에 대한 s계산 및 f계산
        for (i = 0; i < M[0]; i++) {
                s[0][i] = 0.0;
                for (j = 0; j < N; j++) {
                        //s[0][i] += (input[j] / 255.0) * W[0][i][j];            //gray   scale   이미지를
0~1사이의 실수로 바꿈(필기체 인식 MNIST 데이터를 사용할 때)
                        s[0][i] += (input[j] / (double)input_max) * W[0][i][j];                  //0~1사
이의 실수로 바꿔주기 위해 입력의 최대값으로 나눠줌
                }
                f[0][i] = 1.0 / (1.0 + exp(-s[0][i]));                  //시그모이드 함수 적용
        //exp(n):지수함수(e의 n제곱)
        }
        f[0][m0] = 1.0; //더미 입력

        //층1 부터 s계산 및 f계산
        for (layer = 1; layer < NLayer; layer++) {
                for (i = 0; i < M[layer]; i++) {                //layer층의 노드 개수 만큼 돌림
                        s[layer][i] = 0.0; //초기화
                        for (j = 0; j < M[layer - 1] + 1; j++) {      //layer-1층의 노드 개수 만큼 돌림
                                s[layer][i] += f[layer - 1][j] * W[layer][i][j];          //layer층의 i번째
값
                        }
                        f[layer][i] = 1.0 / (1.0 + exp(-s[layer][i])); //layer층의 i번째 뉴런의 출력값
                }
                f[layer][M[layer]] = 1.0; //더미 입력
        }
}
void backward_compute() {
        int i, j;
        double tsum;
```

```
	int k = NLayer - 1; //마지막층
	for (i = 0; i < M[k]; i++) {                    //마지막층의 델타값들 구하기
		delta[k][i] = (D[i] - f[k][i])*f[k][i] * (1 - f[k][i]);                    //D-f:발생   에러
		f*(1-f):시그모이드함수 미분값
	}
	for (k = NLayer - 2; k >= 0; k--) {//중간층의 델타값 구하기
		for (i = 0; i < M[k]; i++) { //k의 층의 델타 값
			tsum = 0.0;
			for (j = 0; j < M[k + 1]; j++) {//k+1층 에서의 델타*가중치
				tsum += delta[k + 1][j] * W[k + 1][j][i];
			}
			delta[k][i] = f[k][i] * (1 - f[k][i])*tsum;
		}
	}
}
void weight_update() {
	int i, j, layer;

	for (i = 0; i < M[0]; i++) {// 0층의 가중치 업데이트
		for (j = 0; j < N; j++) {
			W[0][i][j] += c*delta[0][i] * input[j];                    //학습률,  입력값,
delta값 반영하여 수정
		}
	}
	for (layer = 1; layer < NLayer; layer++) {          // layer 층의 가중치 업데이트
		for (i = 0; i < M[layer]; i++) {
			for (j = 0; j < M[layer - 1] + 1; j++) {
				W[layer][i][j] += c*delta[layer][i] * f[layer - 1][j];
			}
		}
	}

}

int training() {
	int tr = 0, i, j;
	int num = 0;
	int plus_t=1;
	int epoch = N_tr_examples;
	double sum_sq_error;
	char flag = 'n';
	trainTime = 0.0;
	while (plus_t>=1) { //설정 학습횟수 미만일 경우 선택
		do {
			time_t startTime = time(NULL);                    //시작시간 저장

			for (i = 0; i < epoch; i++) {                    //전체  학습  데이
터 학습 시킴
				memcpy(input, trainData[i], sizeof(trainData[i]));
	//메모리 카피 memcpy(dest,sour,size):sour의 메모리영역을 dest에 복사함, size는 복사할 바이트
개수
				input[N - 1] = 1;
				memcpy(D, d_tr[i], sizeof(d_tr[i]));                    //D배열에   i번째
입력의 숫자값(0~9) 넣어줌(명시된 값)
				forward_compute();
				backward_compute();
				weight_update();
			}
			sum_sq_error = 0.0;                    //오차율의 합 초기화

			for (i = 0; i < epoch; i++) {

				memcpy(input, trainData[i], sizeof(trainData[i]));
				input[N - 1] = 1;
				memcpy(D, d_tr[i], sizeof(d_tr[i]));
				forward_compute();
```

```
                                for (j = 0; j < M[NLayer - 1]; j++)
                                        sum_sq_error += (D[j] - f[NLayer - 1][j])* (D[j] - f[NLayer - 1][j]);          //각 층에서의 에러값 제곱(음수일수도 있으므로)

                        }
                        //c = c*0.98;                   //기존 코딩은 학습률을 계속 줄임 -->왜일까??            but 기존코딩대로하니 정확도 낮아져서 주석처리함
                        tr++;                   //트레이닝 횟수
                        avg_sq_error = sum_sq_error / (epoch*M[NLayer - 1]); //오차의 평균
                        printf("#%depoch 후 평균 오차율: %f \n", tr, avg_sq_error);

                        time_t endTime = time(NULL);                    //끝난 시간
                        trainTime += difftime(endTime, startTime);              //총 학습시간 더함
                } while (tr<tr_N+num);                                  //설정한 트레이닝 횟수보다 작을 동안 반복
                printf("목표 트레이닝 횟수만큼 학습하였습니다.\n추가학습 횟수를 적어주세요(그만하려면 0) : ");
                scanf("%d", &plus_t);
                num += plus_t;
        }
        return tr;
}

double test() {
        int num_correct = 0;
        for (int i = 0; i < m2; i++)
                correct[i] = 0;
        int i, index;
        double test_accurancy;
        time_t startTime = time(NULL);                  //시작시간 저장
        for (i = 0; i < N_te_examples; i++) {
                memcpy(input, testData[i], sizeof(testData[i]));
                input[N - 1] = 1;
                memcpy(D, d_te[i], sizeof(d_te[i]));
                forward_compute();
                index = correctNum();                   //인식결과의 숫자
                if (D[index] == 1) {                    //인식한대로 숫자가 index라면
                        num_correct++;                  //맞은 개수 더함
                        correct[index]++;               //각 숫자마다 맞은 개수 더함
                }
        }
        time_t endTime = time(NULL);                    //끝난 시간
        testTime = difftime(endTime, startTime);        //테스트 시간 구함

        test_accurancy = (double)num_correct / N_te_examples;   //테스트한 결과 전체정확도
        return test_accurancy;
}

int correctNum() {
        double temp = f[NLayer - 1][0];
        int ret = 0;
        for (int i = 1; i < M[NLayer - 1]; i++) {                       //출력노드 수만큼 돌림(10)
                if (temp < f[NLayer - 1][i]) {                          //값이 더 큰지 확인함
                        temp = f[NLayer - 1][i];
                        ret = i;
                }
        }
        return ret; //가장 큰 값의 배열의 index 출력
}
```

3) 학습과 테스트를 실행하고, 진행과정과 결과를 보여주는 메인 소스 (main.c)

```
#include "AI.h"
#include <stdio.h>
#include <time.h>
extern double c;

int main() {
	double sum_sq_error = 0.0; //오차들의 합
	avg_sq_error = 0.0;                    //오차의 평균
	double test_accurancy;
	int tr;

	printf("                                                                시작\n");

printf("##############################################################################\n\n");
	printf("각층의 노드 수 : ");
	for (int i = 0; i < NLayer; i++) {
		printf("%d ", M[i]);
	}
	printf("\n1epoch=%d\n", N_tr_examples);
	printf("train_threshold : %f\n", train_threshold);
	printf("학습률 c값: %f\n", c);
	printf("----------------------------------------\n");
	printf("트레이닝 데이터 및 테스트 데이터 읽어오는 중...\n");
	data_Onmemory();
	printf("랜덤 가중치 설정 중...\n ");
	weight_init();              //한번 설정하면 visual 끄기까지 바뀌지 않음(rand함수 사용했으므로)
	printf("트레이닝 시작\n");
	tr = training();
	printf("\n----------------------------------------\n");
	printf("테스트 시작\n");
	test_accurancy = test();
	printf("------------------설정값----------------------\n");
	if (N - 1 == 49)
		printf("<7*7 데이터 그대로 입력하여 학습>\n");
	else
		printf("<특징 추출 후 학습>\n");

	printf("층마다 노드의 개수: ");
	printf("%d ", N - 1);
	for (int i = 0; i < NLayer; i++) {
		printf("%d ", M[i]);
	}
	printf("\n학습률 : %f\n", c);
	printf("학습 횟수(반복한 에폭 수) : %d\n", tr);
	printf("학습 데이터의 수 (1epoch) : %d\n", N_tr_examples);
	printf("테스트 데이터의 수 : %d\n", N_te_examples);

	printf("------------------결과값----------------------\n");
	printf("마지막 평균오차율 : %lf\n", avg_sq_error);
	printf("총 학습시간 : %.1f sec\n", trainTime);
	//printf("테스트시간 : %lf sec\n", testTime);
	printf("전체 숫자 인식 정확도 : %f\% \n", test_accurancy * 100);
	printf("---------------숫자별 인식정확도----------------\n");
	for (int i = 0; i<m2; i++)
		printf("숫자 %d : %f\% \n", i, (double)correct[i] / N_te_examples * 10 * 100.0);

	return 0;
}
```

4. 결과

(a) 변이변화가 결과들에 끼치는 영향 확인

1) 학습률

학습률의 변화는 아래와 같이 주었다. 예상대로 학습률이 높을수록 학습시간이 오래 걸림을 알 수 있었다.

	오차율	학습시간(sec)	전체 정확도(%)
0.5	0.016013	44	73.1875
0.1	0.031213	44	67.3125
0.05	0.044556	44	62.875
0.01	0.072013	44	43.3125
0.005	0.082759	45	30.4375

[표 2. 학습률 변화와 그에 따른 영향 (원본 데이터)]

	오차율	학습시간(sec)	전체 정확도(%)
0.5	0.000577	42	88.125
0.1	0.0028	41	87.0625
0.05	0.007153	41	84.75
0.01	0.029755	41	76.625
0.005	0.043428	41	71.3125

[표 3. 학습률 변화와 그에 따른 영향 (특징추출 데이터)]

2) 학습횟수

예상대로 학습횟수가 많을수록 인식 정확도가 높아지며, 학습 시간이 오래 걸렸다. 또한 학습횟수가 일정 이상 많아지면 오차율이 크게 변하지 않아 인식률에 큰 영향을 미치지 않음을 알 수 있었다. 학습횟수가 많아질수록 인식률의 변화크기가 줄어들었다.

	오차율	학습시간(sec)	전체 정확도(%)
10	0.089922	2	13.0625
20	0.088194	4	18.3125
50	0.079426	11	35.5625
100	0.06726	22	52.0625
200	0.043541	44	61.875
500	0.026578	110	66.3125
1000	0.023221	220	67.9375

[표 4. 학습횟수 변화와 그에 따른 영향 (원본 데이터)]

	오차율	학습시간(sec)	전체 정확도(%)
10	0.072193	2	41.4375
20	0.055472	4	61
50	0.038603	10	71
100	0.021446	21	79
200	0.006714	42	85.875
500	0.001932	104	87.6875
1000	0.001547	206	89.0625

[표 5. 학습횟수 변화와 그에 따른 영향 (특징추출 데이터)]

3) 학습과 테스트 데이터의 수(비율)

첫 번째(1600:1600)은 서로 다른 학습 데이터이다. 각각의 데이터들을 테스트 데이터에서의 학습 데이터의 비중을 따져본다면 각각 0%, 0%, 50%, 100%로 테스트 데이터에서의 학습 데이터의 비중이 높아질수록 인식률이 높아짐을 알 수 있었다.

train	*test*	오차율	학습시간(sec)	전체 정확도(%)
1600	**1600**	0.045856	44	62.3125
2600	**600**	0.031185	72	62
1600	**3200**	0.043376	44	65.5
3200	**3200**	0.032581	87	74.125

[표 6. 데이터의 수 변화와 그에 따른 영향 (원본 데이터)]

train	*test*	오차율	학습시간(sec)	전체 정확도(%)
1600	1600	0.008107	41	85.3125
2600	600	0.003758	67	86.833333
1600	3200	0.007177	42	91.25
3200	3200	0.00362	52	97.8125

[표 7. 데이터의 수 변화와 그에 따른 영향 (표본추출 데이터)]

4) 중간층의 노드의 개수

중간층의 노드들의 개수가 많을수록 인식률이 높게 나올 것이라고 예상했었다. 그러나 예상과는 달리 전체 인식률은 크게 변하지 않았다. 인식률은 비슷하게 나타났으나 학습시간 면에서 많은 차이를 보였다. 총 노드수가 많아질수록 학습시간이 오래 걸렸다. 또한, 50:150, 100:100 데이터를 보면 총 노드수가 같더라도 한 층이 다른 층보다 노드수가 작을 때 미미하지만 인식률도 높고 학습시간도 덜 걸림을 알 수 있었다. 이는 50:100, 100:50 데이터를 확인하였을 때 은닉층 1이 은닉층 2보다 작을 때 더욱 효율적인 결과를 얻을 수 있음을 알 수 있었다.

은닉층1	*은닉층2*	오차율	학습시간(sec)	전체 정확도(%)
50	50	0.43125	31	62.8125
50	100	0.045543	44	62.0625
50	150	0.045442	58	62.0625
50	200	0.046484	72	62.4375
100	50	0.045198	57	63.25
100	100	0.045091	80	61.5625
100	200	0.048282	129	62.25

[표 8. 중간노드 수 변화와 그에 따른 영향 (원본 데이터)]

은닉층1	*은닉층2*	오차율	학습시간(sec)	전체 정확도(%)
50	50	0.007669	28	85.5625
50	100	0.007764	41	85.4375
50	150	0.006095	55	86.6875
50	200	0.008415	69	84.625
100	50	0.007546	50	84.625
100	100	0.006556	75	85.625
100	200	0.009197	124	86

[표 9. 중간노드 수 변화와 그에 따른 영향 (표본추출 데이터)]

5) 제일 인식률 좋은 조건들

여러 가지 변화들을 주어 학습을 시켜 본 결과 인식률이 제일 높았던 조건들을 정리하였다. 인식률이 비슷하면 효율이 좋게 학습시간이 더 적게 걸리는 것으로 선정하였다.

	학습률	학습횟수	테스트 데이터의 수	학습 데이터의 수	중간 노드의 수 m0	중간노드의 수m1
best	0.5	1000	3200	3200	50	50

[표 10. 결과가 가장 좋았던 변이 (원본 데이터)]

	학습률	학습횟수	테스트 데이터의 수	학습 데이터의 수	중간 노드의 수 m0	중간노드의 수m1
best	0.5	1000	3200	3200	50	50

[표 11. 결과가 가장 좋았던 변이 (특징추출 데이터)]

(b) 원본 데이터, 특징 추출 데이터 간의 정확도와 성능 비교

변이들을 계속해서 변화시키며 봤을 때 원본 데이터보다 특징 추출 데이터의 인식률이 항상 높았다. 이는 의미 있는 특징들을 잘 추출한 것으로 보인다. 또한, 제일 인식률이 높았던 조건들로 학습 후 테스트해 본 결과, 원본 데이터는 82.75%, 표본추출 후 데이터는 99.68%로 매우 높은 결과를 얻을 수 있었다. 이는 차원이 낮더라도 의미 있는 데이터를 넣어주는 것이 더 인식률을 높일 수 있음을 알려준다.

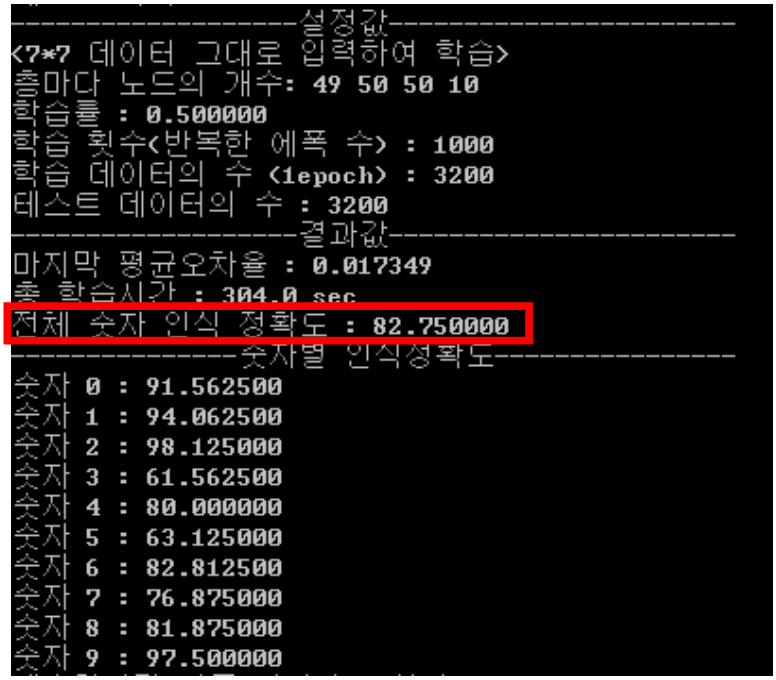
또한, 숫자별 인식정확도를 비교해봤을 때, 특징을 잡기 전에는 인식률이 다른 숫자들에 비해 떨어지는 숫자들이 있었지만, 특징을 추출한 후에는 그 숫자들도 인식률이 높아졌음을 알 수 있었다.

오차율	학습시간	전체정확도
0.01739	304	82.75

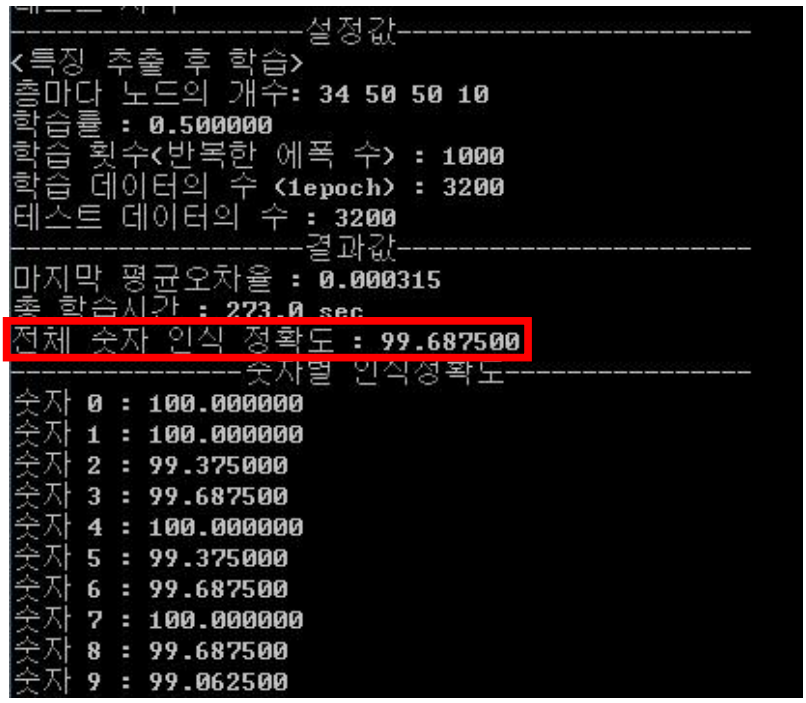
[표 12. 가장 좋은 변이들로 설정 후 결과 (원본 데이터)]

오차율	학습시간	전체정확도
0.000315	273	99.6875

[표 13. 가장 좋은 변이들로 설정 후 결과 (특징추출 데이터)]



[그림 7. 가장 좋은 변이들로 설정 후 결과창 (원본 데이터)]



[그림 8. 가장 좋은 변이들로 설정 후 결과창 (특징추출 데이터)]

5. 결론

여러 가지 변화를 주며 학습을 해보았다. 현재 7*7 행렬의 숫자 인식으로는 학습률이 큰 값인 것이 좋게 나왔지만, 더욱 정밀한 문제에 대해서는 학습률이 작은 것을 사용하는 것이 인식률을 높일 수 있을 것 같다는 생각이 들었다. 예상과 가장 크게 다른 결과를 보여줬던 것은 중간층의 노드의 개수였다. 노드의 개수가 많아지면 더 정밀하게 인식이 가능할 것이라 생각하였지만 노드 수가 너무 많아지자 오히려 인식률이 낮아지는 결과를 보였다. 이는, 너무 많은 노드들을 거치게 되면 오류들에 대한 개입도 커져서 나오는 결과인 것 같다. 전, 후 층 간의 노드 수 비율도 중요했고, 너무 높은 것보다는 적당한 수준을 유지하는 것이 좋을 것 같았다.

본문에 넣지는 않았지만, 학습을 진행하면서 현재 사용하고 있는 소스가 원래의 학습된 데이터뿐만 아니라 새로운 데이터가 들어왔을 때에도 제대로 인식을 할 수 있는지에 대한 궁금증이 생겨서 간단하게 일반화 특성도 평가해보았다. 3겹 교차검증을 거친 결과, 원본 데이터와 특징추출 데이터에 대해 각각 약 81%, 약 92% 정도로 괜찮게 나온 편이었다.

<3겹 교차 검증(원본)>

- 3세트로 나눔, 제일 좋은 조건들로 실행
- 전체 데이터를 1000(data 1), 1100(data 2), 1100(data 3)으로 분류

train	*train*	*test*	오차율	학습시간(sec)	전체 정확도(%)
data 1	data 2	data 3	0.016911	196	75.454545
data 3	data 1	data 2	0.013723	196	78
data 2	data 3	data 1	0.003684	197	86.7
교차검증 결과 (평균)			0.011439333	196.3333333	80.051515

[표 10. 3겹교차 검증 결과 (원본 데이터)]

<3겹 교차 검증(특징)>

- 3세트로 나눔, 제일 좋은 조건들로 실행(데이터 셋만 변경)
- 전체 데이터를 1000(data 1), 1100(data 2), 1100(data 3)으로 분류

train	*train*	*test*	오차율	학습시간(sec)	전체 정확도(%)
data 1	data 2	data 3	0.000622	177	92
data 3	data 1	data 2	0.000623	179	90.181818
data 2	data 3	data 1	0.000459	188	93.7
교차검증 결과 (평균)			0.000568	181.3333333	91.960606

[표 11. 3겹교차 검증 결과 (특징추출 데이터)]

또한, 1차 프로젝트에서 MLP를 사용하지 않고 내가 정한 특징들로 패턴 인식을 했을 때는 3, 4, 5, 8, 9에 대한 인식률이 낮았었는데, 이번 프로젝트에서는 어떤 숫자에 대한 인식률이 낮을지 궁금했다. 변이들을 변화시켜 가며 학습을 시킬 때, 숫자별 정확도를 출력하여 MLP를 사용하였을 때 인식에 더욱 많은 노력이 필요한 숫자를 확인해보았다. 그 결과, 특정 숫자의 인식률이 떨어지지는 않았으나, 가장 인식하기 쉬운 숫자인 1은 항상 다른 숫자보다 높은 인식률을 보였다.

마지막으로, 컴퓨터의 상태에 따라 결과값들이 계속 바뀌기 때문에 한꺼번에 학습을 진행해서 데이터를 확인해야 하는 점이 어려웠다. 그래서 학습 시간을 줄이는 효율적인 조건들에 집중하여 실험하였는데, 더 좋은 사양을 사용한다면 학습 시간이 아닌 오로지 인식률에만 집중하여 실험해볼 수 있을 것이라 생각한다.