인공지능 과제 Ⅱ

임베디드시스템공학과 3학년 202201673 이진성

Artificial Intelligence Assignment Ⅱ

Implementation of an Korean OCR Model Using ANN

내 용

- C언어와 Python을 사용하여 한글 OCR 모델을 ANN을 사용하여 구현
- 데이터셋으로 사용하는 글자는 개인별로 다르며 본 레포트의 데이터셋은 아래와 같음
- ~~데이터셋 텍스트 삭제~~
- 이미지는 64x64 사이즈의 흑백 BMP 혹은 PNG
- 배경이 검은색, 글자가 흰색인 1비트 비트맵 이미지를 사용함
- 최소 10개 이상의 폰트를 사용할 것을 권장함에 있어 28개의 폰트를 데이터셋을 생성하는데 사용함
- 레이어는 최대 16개, 레이어당 노드는 최대 256개
- Hidden Layer는 3개 이상 14개 이하(전체 레이어는 5개 이상 16개 이하)
- 노드수는 각자 선택, 다만 가능한 최적으로 튜닝되어야 함
- 입출력 레이어는 그 크기를 달리할 수 있음
- 입출력 레이어의 구성은 데이터셋의 종류에 따라 각자 편한 방법으로 진행
- C언어와 Python은 반드시 그 방법론과 하이퍼 파라미터가 동일해야 함
- Python 코드에 있어서 데이터 입출력, 그래프 외의 추가적인 라이브러리의 사용을 엄격히 금지함
- Batch 가용 코드를 구성할 경우 실험과 테스트가 편할 것으로 사료됨
- 사용법을 본 레포트에 포함해야 함
- 소스의 구조를 잘 나타낼 것
- 실험 방법, 입출력, 결과, 최종 평가를 설명

Ⅰ. 데이터 입력 코드의 구현

1.1. 비트맵 이미지 load를 위한 struct의 정의

```c
typedef struct {
    unsigned char signature[2];
    unsigned int fileSize;
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int dataOffset;
    unsigned int headerSize;
    int width;
    int height;
    unsigned short planes;
    unsigned short bitsPerPixel;
    unsigned int compression;
    unsigned int imageSize;
    int xPixelsPerMeter;
    int yPixelsPerMeter;
    unsigned int colorsUsed;
    unsigned int importantColors;
} BMPHeader;
```

비트맵 이미지를 불러오기 위한 bitmap header struct이다. C언어의 기본 파일 입출력을 사용하여 이미지를 불러오며, 이를 위한 struct이다. 해당 struct는 ChatGPT의 지원을 통해 작성되었다. typedef_struct/image_struct.h를 통하여 정의하였다.

1.2. 비트맵 이미지 load

```c
void readBMPHeader(FILE *file, BMPHeader *header) {
    fread(header->signature, 1, 2, file);
    fread(&header->fileSize, 4, 1, file);
    fread(&header->reserved1, 2, 1, file);
    fread(&header->reserved2, 2, 1, file);
    fread(&header->dataOffset, 4, 1, file);
    fread(&header->headerSize, 4, 1, file);
    fread(&header->width, 4, 1, file);
    fread(&header->height, 4, 1, file);
    fread(&header->planes, 2, 1, file);
    fread(&header->bitsPerPixel, 2, 1, file);
    fread(&header->compression, 4, 1, file);
    fread(&header->imageSize, 4, 1, file);
    fread(&header->xPixelsPerMeter, 4, 1, file);
    fread(&header->yPixelsPerMeter, 4, 1, file);
    fread(&header->colorsUsed, 4, 1, file);
    fread(&header->importantColors, 4, 1, file);
}
```

비트맵 이미지를 1.1.의 struct에 불러오기 위한 readBMPHeader 함수를 정의하였다. 해당 함수는 1.1.과 동일하게 ChatGPT의 지원을 통해 작성되었다. load_n_preprocess/image_load.c에 작성되었다.

1.3. 이미지의 load와 배열 저장과 반환

```c
FILE* load_image_n_data(unsigned char pixels[64][64], int idx){
    char filepath[200];
    char* filepathP = filepath;
    if(idx)sprintf(filepath,"../makeImage/exportImage/text%05d.bmp",idx);
    else{
        printf("Enter Image File Path : ");
        scanf("%s",filepath);
    }

    FILE *file = fopen(filepathP, "rb");
    if (file == NULL) //...
    BMPHeader header;
    readBMPHeader(file, &header);
    if (header.bitsPerPixel != 1) //...

    int width = header.width;
    int height = header.height;

    if(width != 64 || height != 64) //...
    fseek(file, header.dataOffset, SEEK_SET); // 이미지 데이터 위치로 이동
    unsigned char raw_pixels[64][8] = {0,}; // 각 픽셀을 저장할 배열 할당
    fread(raw_pixels, 1, width * height / 8, file); // 이미지 데이터 읽기

    for(int i=0;i<64;i++){
        for(int j=0;j<64;j++){
            int byteIndex = j / 8;
            int bitIndex = 7 - (j % 8);
            pixels[63-i][j] = ((raw_pixels[i][byteIndex] >> bitIndex) & 1);
        }
    }
    return file;
}
```

이미지를 배열에 저장하기 위해 load_image_n_data 함수를 정의하였으며 입력으로 이를 저장하기 위한 64x64 사이즈의 배열과 파일 인덱스를 위한 정수를 입력받는다. 데이터셋 이미지는 특정 경로에 저장되어 있으며, 이미지 생성 순서에 따라 “text{%05d}” 포맷으로 저장되어 있다. 이를 인덱스를 사용하여 이미지의 경로 string을 생성하고, 이를 1.2.의 readBMPHeader 함수를 통해 값을 받아들인다.

데이터셋으로 사용한 이미지는 1비트 비트맵 이미지로 이를 바로 배열 데이터로 사용할 수 없다. 이를 위해 일차적으로 64x8 사이즈의 배열을 선언하고 파일의 데이터를 저장한다. 이후 함수의 입력으로 받은 64x64 사이즈의 배열에 shift 연산을 사용하여 배열의 한 칸에 하나의

비트에 대한 값을 저장한다. 샘플 이미지의 배경이 검은색, 글자 영역이 흰색으로 배경은 0, 글자 영역은 1의 값을 갖는다. 반대로 배경이 흰색인 이미지를 load하는 경우 line 28의 연산 값에 not을 취하여 배열에 저장하면 글자 영역의 값을 1로 받을 수 있다. 해당 함수는 load_n_preprocess/load_data.c에 작성되었다.

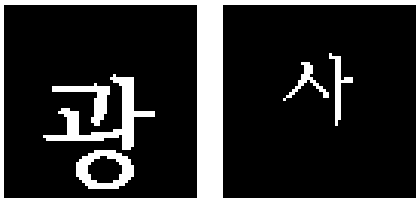

그림 1. 비트맵 이미지

1.4. 이미지의 확장

한글 이미지를 통한 OCR 모델 구현 과제를 받은 직후 떠올린 문제점은 입력 이미지에 위치한 텍스트의 위치와 크기를 알 수 없다는 것이었다. 특히 64x64 사이즈의 이미지를 사용하는 만큼 위치의 다양성과 크기의 다양성이 존재하므로 이를 일반화하지 않고 연산을 하는 것은 위험하다고 판단했다.

동시에 진행하고 있는 과제의 타 주제의 경우 증강을 할 것을 권장하였으나, 텍스트 인식의 경우 텍스트의 형태가 일정하기 때문에 증강을 하는 것 보다는 텍스트의 크기와 위치를 일반화 시키는 것이 학습 데이터셋을 경량화 시키는 등 학습 시간 등을 고려하였을 때 더 나은 방안이라고 판단하였다.

이를 위해 입력 데이터를 확장하여 64x64 이미지 내에서 꽉 차게끔 배열의 값을 변형할 필요성을 찾을 수 있었다. 이는 이미지 내에서 텍스트 영역을 bounding box를 표시하고 해당 영역을 이미지의 전체 영역으로 확장시킨다고 이해하면 된다.

```
1 typedef struct{
2     unsigned int r_s;
3     unsigned int c_s;
4     unsigned int r_e;
5     unsigned int c_e;
6     unsigned int r_size;
7     unsigned int c_size;
8 }cropRange;
```

먼저 이미지 내에서 텍스트 영역의 경계를 나타내기 위한 struct를 정의하였다. 이는 함수의 반환값으로 사용함을 통해 한번에 값을 반환하기 위해 정의하였다. r과 c는 row와 column, s와 e는 start와 end의 의미를 가진다. s를 통해 좌상단 좌표를, e를 통해 우하단 좌표를 얻을 수 있다. size는 e와 s의 차를 이용하여 해당 텍스트의 사이즈를 저장한다.이는 load_n_preprocess/preprocessing.h를 통해 정의하였다.

struct로 정의된 cropRange를 얻기 위한 함수 cropIndex 함수를 정의하였는데, 이는 앞에서 기술한 것과 같이 64x64 배열 영역 내에서 텍스트가 위치한 bounding box의 인덱스를 얻기 위한 함수이다. bounding box의 인덱스를 얻기 위한 로직은 단순하게

구현했는데, 각 column과 row 별로 값을 누적하는 배열을 만들고 값을 0으로 초기화한 후, 각 column과 row 별로 값을 누적하여 최초로 값이 0이 아닌 값이 나온 인덱스를 사용한다.

로직을 단순화 하기 위해 각 배열의 값이 0 혹은 1임을 이용하여 배열 값을 누적하고, 조건문 내에서는 해당 배열의 누적값을 바로 확인하여 0이 아닌 경우 true임을 사용하여 값을 판별하였다.

```
cropRange cropIndex(unsigned char originalImage[64][64]){
    unsigned int r[64] = {0,};
    unsigned int c[64] = {0,};
    for(int i=0;i<64;i++){
        for(int j=0;j<64;j++){
            r[i] += originalImage[i][j];
            c[j] += originalImage[i][j];
        }
    }
    cropRange returnV;
    for(int i=0 ;i<64;i++)if(c[i]){returnV.c_s = i;break;}
    for(int i=63;i>=0;i--)if(c[i]){returnV.c_e = i;break;}
    for(int i=0 ;i<64;i++)if(r[i]){returnV.r_s = i;break;}
    for(int i=63;i>=0;i--)if(r[i]){returnV.r_e = i;break;}
    returnV.r_size = returnV.r_e - returnV.r_s;
    returnV.c_size = returnV.c_e - returnV.c_s;
    return returnV;
}
```

해당 함수를 통해 cropRange로 정의된 struct를 반환하는데, 이를 통해 이미지를 확장시킨다.

```
void imageStretch(unsigned char originalImage[][64],
                  unsigned char returnImage[][64],
                  cropRange cropRangeInput){
    for(int i=0;i<64;i++){
        for(int j=0;j<64;j++){
            unsigned int oR = round(cropRangeInput.r_s+cropRangeInput.r_size * i/63);
            unsigned int oC = round(cropRangeInput.c_s+cropRangeInput.c_size * j/63);
            returnImage[i][j] = originalImage[oR][oC];
        }
    }
    return;
}
```

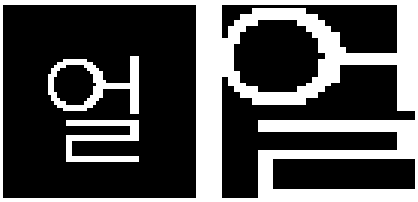

그림 2. 원본 이미지와 확장 이미지

이미지 확장은 imageStretch 함수를 통해 이루어지는데, 원본 배열과 확장된 값을 넣기 위한 배열, 그리고 cropIndex 함수를 통해 반환된 인덱스를 입력인자로 받는다. cropRange struct의 값을 통해 원본 배열에서의 위치와 확장된 배열에서의 위치의 관계를 파악하고 확장된 배열을 생성한다. 이 두 함수는 load_n_preprocess/preprocessing.c에 작성되었다.

1.4. 이미지의 축소

입출력 레이어의 크기를 달리할 수 있다는 단서조항이 있음에도 불구하고 64x64 이미지의 모든 픽셀을 입력 레이어에 입력하고자 하면 4,096개의 노드가 필요하다. 이를 기본 레이어 당 노드 조건으로 제시한 256개로 축소하고자 하였다. $\sqrt{256}=16$으로 256개의 입력 노드를 사용하기 위해서는 16x16 사이즈의 이미지를 입력으로 사용하면 된다. 이를 위해 64x64 사이즈를 갖는 배열에서 4x4 영역을 묶어 평균값을 한 노드의 입력값으로 사용하고자 하였다.

```
void image64to16(unsigned char originalImage[][64],
                 double returnImage[][16]){
    for(int i=0;i<16;i++){
        for(int j=0;j<16;j++){
            unsigned int sum = 0;
            for(int x=0;x<4;x++){
                for(int y=0;y<4;y++){
                    sum += originalImage[i*4+x][j*4+y];
                }
            }
            returnImage[i][j] = ((double)(sum)) / 16;
        }
    }
    return;
}
```

해당 연산을 위해 Image64to16을 정의하였으며, 평균값을 연산하기 위해 이중 for loop 내에서 4x4 픽셀의 값을 누적하여 0 이상, 16 이하의 값을 만든 후 double로 casting한 후 16으로 나눠 하나의 픽셀 값으로 치환하였다. 이를 통해 16x16 사이즈의 배열을 만들었다.

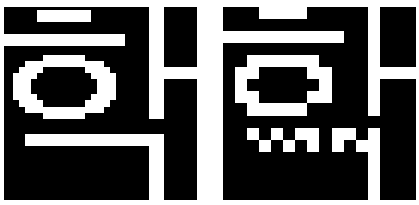

그림 3. 64x64 사이즈의 비트맵 확장 이미지와 16x16 사이즈의 gray scale 다운샘플링 이미지

II. 출력 레이어의 구성과 Target 구성

2.1. 유니코드

유니코드는 문자와 기호를 컴퓨터 시스템에서 일관되게 표현하고 처리하기 위한 국제 표준으로, 한글 또하 유니코드를 통해 나타낼 수 있다. 이때 한글은 한 글자가 3byte의 용량을 사용한다. 이때 초성 19개, 중성 21개, 종성 28개의 조합을 통해 한글을 유니코드로 나타낸다. 본 과제에서는 이를 활용하여 출력층과 target을 구성하였다.

2.2. 인덱스와 유리수의 변환

19개의 초성은 0~18의 값을, 21개의 중성은 0~20의 값을, 28개의 종성은 0~27의 값을 가진다고 볼 수 있다. 이를 0 초과 1 미만의 범위에 매칭하고자 하였다. 이때 갑시 0과 1이 되지 않도록 하기 위해 값에 1을 더하고 최대값보다 1만큼 더 큰 값이 1이 되도록 매핑하였다. 초성이 'ㅊ'인 경우 인덱스가 14이므로 $\frac{1+14}{1+19+1}$으로 변환된다. 이와 반대로 이와 같은 유리수를 입력받았을 때 인덱스로 변환할 수도 있는데, 종성의 경우 30(1+28+1)을 곱한 후 1을 뺀 값을 반올림 하여 원래 인덱스로 추론한다.

2.3. 출력 레이어의 구성

출력 레이어는 3개의 노드로 구성한다. 이때 각 노드의 값은 초성, 중성, 종성의 값을 가진다. 이때 출력층의 활성화 함수는 Sigmoid를 사용함으로서 0 초과, 1 미만의 값을 반환하도록 하였다.

2.4. 인덱스와 유리수를 위한 struct의 선언, 변환 함수

2.1.과 2.2.의 패턴에 따라 하나의 한글에 대한 초성, 중성, 종성을 저장하기 위한 구조체를 만들고자 하였다.

```
typedef struct {
    int first;
    int middle;
    int last;
} WORDIDX;
```

```
typedef struct {
    double first;
    double middle;
    double last;
} WORDVAR;
```

WORDIDX와 WORDVAR struct를 선언하여 인덱스와 유리수 값을 저장 할 수 있도록 하였다. 이는 typedef_struct/korean_word.h에 정의하였다.

둘 간의 변환을 위해 2.2.에서 설명한 방법으로 변환하기 위한 함수 convert_IDX2VAR과

convert_VAR2IDX를 정의하였다.

```
WORDVAR convert_IDX2VAR(WORDIDX idx){
    WORDVAR returnV;
    returnV.first = (double)(idx.first + 1) / 21.0;
    returnV.middle = (double)(idx.middle + 1) / 23.0;
    returnV.last = (double)(idx.last + 1) / 30.0;
    return returnV;
}
```

```
WORDIDX convert_VAR2IDX(WORDVAR var) {
    WORDIDX returnV;
    returnV.first = (int)round(var.first * 21.0) - 1;
    returnV.middle = (int)round(var.middle * 23.0) - 1;
    returnV.last = (int)round(var.last * 30.0) - 1;

    if (returnV.first < 0) returnV.first = 0;
    if (returnV.first > 18) returnV.first = 18;
    if (returnV.middle < 0) returnV.middle = 0;
    if (returnV.middle > 20) returnV.middle = 20;
    if (returnV.last < 0) returnV.last = 0;
    if (returnV.last > 27) returnV.last = 27;

    return returnV;
}
```

convert_IDX2VAR은 load_n_preprocess/load_target.c에, convert_VAR2IDX는 word_inference/word_inference.c에 작성하였다. 또한 convert_VAR2IDX를 통해 추출된 인덱스를 통해 유니코드를 생성할 수 있도록 convert_IDX2Char를 정의하였다.

```
char* convert_IDX2Char(WORDIDX idx) {
    int unicode = 0xAC00 + (idx.first * 21 * 28) + (idx.middle * 28) + idx.last;
    char* result = (char*)malloc(sizeof(char)*4);
    result[0] = (char)((unicode >> 12) | 0xE0);            // 첫 바이트
    result[1] = (char)(((unicode >> 6) & 0x3F) | 0x80);    // 둘째 바이트
    result[2] = (char)((unicode & 0x3F) | 0x80);           // 셋째 바이트
    result[3] = 0;
    return result;
}
```

convert_IDX2Char는 convert_VAR2IDX와 동일한 word_inference/word_inference.c에 작성하였다.

2.5. Target의 load

1.3.에서 이미지의 생성 순서에 따라 “text{%05d}” 형식으로 이미지를 생성한다고 하였다. 이미지를 생성하는 과정에서 동시에 텍스트 파일에 “text{05d}_{02d}_{02d}_{02d}”와 같은

형식으로 텍스트를 저장하도록 하였다. 이때 text{05d}는 생성되는 이미지와 동일한 숫자이며, {02d}_{02d}_{02d}는 생성된 이미지의 글자를 유니코드의 한글 패턴에 따라 초성, 중성, 종성을 저장한 것이다. 이를 통해 이미지에 쓰여있는 글자가 무엇인지 파악할 수 있다.

이를 사용하여 이미지를 불러옴과 동시에 학습 이미지에 대한 target 값를 불러오고자 하였다.

```
WORDIDX find_target(int idx) {
    WORDIDX returnV = {0,0,0};
    FILE *file = fopen("../makeImage/TextIndexCode.txt", "r");
    if (file == NULL) //...
    char line[256];
    char target[16]; // To store text{%05d}
    sprintf(target, "text%05d_", idx);
    while (fgets(line, sizeof(line), file)) {
        if (strncmp(line, target, strlen(target)) == 0) {
            if (sscanf(line + strlen(target),
             "%02d_%02d_%02d",
             &returnV.first, &returnV.middle, &returnV.last) == 3) {
                fclose(file);
                return returnV; // Success
            }
        }
    }
   //...
}
```

target 값을 불러오기 위한 find_target을 정의하였으며, 입력으로 이미지의 파일이름과 동일한 인덱스를 받는다. 해당 함수는 target 값이 저장되어 있는 텍스트 파일을 열어 해당 인덱스의 값을 찾아 뒤에 연결된 3개의 숫자를 읽어들여 WORDIDX에 저장함으로서 target 값의 인덱스를 가져와 반환한다. 해당 함수는 load_n_preprocess/load_target.c에 작성하였다.

III. Batch 구성과 thread를 통한 가속

3.1. Batch의 구성

본 과제에서는 121개의 텍스트에 대해서 28개의 폰트를 사용하여 데이터셋을 구성하였다. 때문에 흔히 2의 제곱으로 batch의 사이즈를 정하는 것과 달리 하나의 폰트의 모든 텍스트를 하나의 batch에 대한 사이즈로 사용하고자 하였다. batch 하나의 사이즈는 121이다.

3.2. Thread 구성

Batch를 통한 연산은 순차적으로 진행될 필요가 없다. 때문에 각 batch는 독립적으로 동작하여도 무방하며, 병렬로 연산하게 할 수 있다. 이를 위해 학습의 순전파와 역전파, 테스트의 순전파를 batch 사이즈의 수만큼 thread를 호출하여 thread를 통해 연산을 수행하도록 하였다.

3.3. Pthread의 사용

Pthread는 리눅스 시스템에서 thread를 사용할 때 사용한다. 본 과제의 제출자가 메인으로 사용하는 시스템은 맥으로 pthread를 사용할 수 있으며, 윈도우 시스템에서도 WSL을 사용하면 pthread를 사용할 수 있다. 초기에는 윈도우에서의 구동을 위해 docker 컨테이너를 사용하는 것을 고려하였으나, 설치과정을 비롯한 변수들을 고려하였을 때 윈도우 시스템에서는 WSL을 사용하는 것이 더 합리적이라고 판단하였다.

3.4. thread 입력 인수 struct 정의

Thread를 통해 순전파와 역전파 연산을 수행하게 되는데, pthread를 사용할 경우 입력인자를 사용하기 위해서는 포인터를 사용해야한다. 이를 위해 순전파와 역전파에 대한 입력인자로 사용하기 위한 struct를 정의하였다.

```
typedef struct{
    int thread_id;

    int fileIdx;
    int L;
    int* N;
    double** Z;
    double** H;
    double*** weight;
    WORDVAR* target;

    int size;
}forwardThreadData;
```

forwardThreadData는 순전파를 위한 struct로 이미지 파일의 인덱스와 신경망에 대한 정보, target값을 저장하기 위한 WORDVAR을 요소로 가진다.

```
typedef struct{
    int thread_id;

    int fileIdx;
    int L;
    int* N;
    double** Z;
    double** H;
    double*** weight;
    WORDVAR* target;

    double** delta;
    double learningRate;

    int size;
}backwardThreadData;
```

backwardThreadData는 역전파를 위한 struct로 forwardThreadData와 유사하나 각 노드의 delta와 학습률을 추가적으로 가지도록 구성하였다.

3.5. CUDA 가속

CUDA 가속을 사용하기 위해서는 Nvidia Developer Toolkit을 비롯한 다양한 세팅이 필요하다. 이를 윈도우와 리눅스 등 다양한 시스템에서 동작하도록 코드를 구성하기에는 시간적 여유가 부족했다. 또한 thread를 적용하였을 때 레이어와 노드의 사이즈에 따라 1 epoch을 연산하는데 걸리는 시간이 0.5s에서 5s 사이임을 감안하였을 때 CUDA 가속을 필수적으로 사용할 필요성을 느끼지 못하였다. 때문에 본 실험에서 CUDA 가속의 적용은 배제하였다.

Ⅳ. 가중치 초기화와 활성화 함수

4.1. 가중치 초기화

가중치를 초기화하는데 일반적으로 많이 사용되는 단순 랜덤, Xavier, He 초기화에 대한 함수를 작성하였으며 이를 initializeWeight 함수를 통해 연산하도록 작성하였다. 이때 Random, Xavier, He를 define하여 사용하였다.

```
double initializeWeight(double min,
                        double max,
                        int outputNodeSize,
                        int inputNodeSize,
                        int initializeType){
    if(initializeType == Random){
        return initializeRandom(min,max);
    }else if(initializeType == Xavier){
        return initializeXavier(outputNodeSize, inputNodeSize);
    }else if(initializeType == He){
        return initializeHe(outputNodeSize);
    }
    printf("initializing type error\n");
    exit(1);
}

double initializeRandom(double min, double max){
    return (((double)rand() / RAND_MAX) * (max - min) + min);
}

double initializeXavier(int outputNodeSize, int inputNodeSize){
    double limit = sqrt(6.0/(outputNodeSize+inputNodeSize));
    return (((double)rand() / RAND_MAX) * 2 * limit - limit);
}

double initializeHe(int outputNodeSize){
    double stddev = sqrt(2.0/outputNodeSize);
    return (((double)rand()/RAND_MAX) * 2 * stddev - stddev);
    // return (stddev * ((float)rand() / RAND_MAX));
}
```

4.2. 활성화 함수의 정의

활성화 함수 또한 4.1.에서 InitializeWeight 함수와 같이 하나의 함수로 연산할 수 있도록 별도의 함수를 선언하였다. 출력층에서 softMax를 사용하지 않기 때문에 Sigmoid, Tanh, ReLU, leaky ReLU에 대해서만 처리하도록 코드를 구성하였다.

```c
double activationFunction(double input, int activate){
    input = clip(input, -10.0, 10.0);
    if(activate == SIGMOID){
        return sigmoid(input);
    }else if(activate == TANH){
        return tanh_clip(input);
    }else if(activate == RELU){
        return relu(input);
    }else if(activate == LEAKY_RELU){
        return leaky_relu(input,0.1);
    }
    return -9999.9999;
}
```

역전파 과정에서 사용하는 미분한 활성화 함수를 위한 함수를 추가적으로 선언하였다.

```c
double activationFunctionDerivative(double input, int activate){
    if(activate == SIGMOID){
        return sigmoid_derivative(input);
    }else if(activate == TANH){
        return tanh_derivative(input);
    }else if(activate == RELU){
        return relu_derivative(input);
    }else if(activate == LEAKY_RELU){
        return leaky_relu_derivative(input,0.1);
    }
    return -9999.9999;
}
```

activateFuncion과 activationFunctionDerivative는 training source 내에 위치해 있으며 함수 내에서 호출하는 함수는 math_func/math_func.c에 작성하였다.

4.3. 활성화 함수의 clipping

```c
double sigmoid_derivative(double x) {
    double sig = sigmoid(x);
    return sig * (1.0 - sig);
}
double tanh_clip(double x) {
    // Clipping input to prevent overflow
    if (x > 20) x = 20;   // Avoid overflow
    if (x < -20) x = -20; // Avoid underflow
    return tanh(x);
}
```

Sigmoid와 tanh를 활성화 함수로 사용하는 상황에서 실행 초기에 20 epoch 내에서

가중치와 노드들의 값이 nan이 발생하는 문제가 발생하였다. 이를 해결하기 위해 해결책을 탐색하였으며, 이 과정에서 clipping에 대한 내용을 찾았다. 이를 통해 sigmoid와 tanh에 clipping을 적용하여 특정 값을 벗어나지 못하도록 처리하였다.

V. 평가 지표

5.1. Error

기본적으로 error는 MSE(Mean Squared Error)를 사용하였다. WORDVAR의 간격이 0.04($\frac{1}{25}$)이라고 가정하였을 때(초성, 중성, 종성은 각 19, 21, 28개), convert_VAR2IDX 연산을 하는 과정에서 반올림을 하기 때문에 target과 추론 값의 차이가 0.02 이내일 경우 신뢰할 만하다고 평가할 수 있다. 이때 0.02의 제곱은 0.004이므로 MSE가 0.004보다 작을 때 해당 모델이 어느정도 신뢰성이 높다고 판별할 수 있다. 이와 같은 방식으로 MSE를 통해 모델의 신뢰도를 평가한다. Test set에 대한 MSE를 사용한다.

5.2. Accuracy

기본적으로 모델의 테스트 출력에 따른 정확도를 판별한다. 한 글자 당 초성과 중성, 종성 3개의 데이터가 나오며, Test set에서 target과 일치하는 비율로 판별한다.

5.3. ±Accuracy

5.2.의 Accuracy는 MSE가 0.0025에서 0.0011 사이일 때 높은 값을 보인다. MSE가 이보다 높은 경우에는 accuracy는 현저히 떨어지게 된다. 이때 연속된 인덱스를 통해 글자를 추정하는 방식에 따라 MSE가 떨어질수록 target에서 멀어지는 인덱스를 도출할 가능성이 높아진다. 이에 따라 1개 내지 2개의 인덱스 차이가 나는 값이 도출되는 정확도를 추가적으로 구한다. 이를 코드 상에서는 calc_accuracy_pm1과 calc_accuracy_pm2로 나타낸다.

MSE와 Accuracy, Accuracy_pm1, Accuracy_pm2를 연산하는 과정은 아래와 같다.

```
int all_accuracy = 0;
int cor_accuracy = 0;
int cor_pm1_accuracy = 0;
int cor_pm2_accuracy = 0;

for(int i=0;i<BATCH_SIZE;i++){
    WORDVAR error;
    error.first  = target[i].first  -H[i][L-1][0];
    error.middle = target[i].middle -H[i][L-1][1];
    error.last   = target[i].last   -H[i][L-1][2];
    error.first  = error.first  * error.first;
    error.middle = error.middle * error.middle;
    error.last   = error.last   * error.last;

    double error3 = error.first + error.middle + error.last;
    error3 /=3;
    LOSS_MSE += error3;

    WORDIDX output_IDX   = convert_VAR2IDX((WORDVAR)
                                           {H[i][L-1][0],
                                           H[i][L-1][1],
                                           H[i][L-1][2]});
    WORDIDX for_accuracy = convert_VAR2IDX(*target);

    unsigned int ac_1 = (output_IDX.first  - for_accuracy.first);
    unsigned int ac_2 = (output_IDX.middle - for_accuracy.middle);
    unsigned int ac_3 = (output_IDX.last   - for_accuracy.last);
    unsigned int ac_1_s = (unsigned int)(ac_1 * ac_1);
    unsigned int ac_2_s = (unsigned int)(ac_2 * ac_2);
    unsigned int ac_3_s = (unsigned int)(ac_3 * ac_3);
    if(ac_1_s == 0)cor_accuracy++;
    if(ac_2_s == 0)cor_accuracy++;
    if(ac_3_s == 0)cor_accuracy++;
    if(ac_1_s < 2) cor_pm1_accuracy++;
    if(ac_2_s < 2) cor_pm1_accuracy++;
    if(ac_3_s < 2) cor_pm1_accuracy++;
    if(ac_1_s < 5) cor_pm2_accuracy++;
    if(ac_2_s < 5) cor_pm2_accuracy++;
    if(ac_3_s < 5) cor_pm2_accuracy++;
    all_accuracy +=3;
}
calc_accuracy     = ((double)cor_accuracy)     / ((double)all_accuracy);
calc_accuracy_pm1 = ((double)cor_pm1_accuracy) / ((double)all_accuracy);
calc_accuracy_pm2 = ((double)cor_pm2_accuracy) / ((double)all_accuracy);
}
LOSS_MSE /= BATCH_SIZE * TEST_S;
```

Ⅵ. 가중치 저장과 저장 가중치 읽기

6.1. 가중치 저장

가중치는 csv 파일로 저장한다. 레이어 수, 각 레이어 당 노드 수, 레이어별 가중치를 순차적으로 저장한다.

```c
void save_weights_to_csv(const char *filename,
                         int num_layers,
                         int *nodes_per_layer,
                         double ***weights) {
    FILE *file = fopen(filename, "w");
    if (file == NULL) //...
    fprintf(file, "%d\n", num_layers);
    for (int i = 0; i < num_layers; i++) {
        fprintf(file, "%d", nodes_per_layer[i]);
        if (i < num_layers - 1) {
            fprintf(file, ",");
        }
    }
    fprintf(file, "\n");
    for (int i = 0; i < num_layers - 1; i++) {
        int rows = nodes_per_layer[i];
        int cols = nodes_per_layer[i + 1];
        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                fprintf(file, "%lf", weights[i][r][c]);
                if (c < cols - 1) {
                    fprintf(file, ",");
                }
            }
            fprintf(file, "\n");
        }
    }
    fclose(file);
    printf("Weights saved successfully to %s\n", filename);
    return;
}
```

save_weights_to_csv 함수는 csv_operate/export_weight_csv.c에 작성하였다.

6.2. 저장 가중치 읽기

6.1.에서 csv 파일로 저장한 가중치를 추론모델에서 읽기 위해 load_weights_to_csv를 정의하였다. 6.1.에서 저장한 순서 그대로 값을 읽어온다. 이때 입력인자에 값을 받도록 하여야 하므로 모든 입력 인자에 대해 포인터로 값을 받도록 하였다.

가중치 파일 경로는 inference.c의 main 함수 내에서 입력받도록 하였으며, training.c의 구조와 동일한 변수에 그대로 값을 불러온다.

```c
void load_weights_from_csv(const char *filename,
                           int *num_layers,
                           int **nodes_per_layer,
                           double ****weight) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) //...
    fscanf(file, "%d\n", num_layers);
    *nodes_per_layer = (int*)malloc(sizeof(int) * (*num_layers));
    for (int i = 0; i < *num_layers; i++) {
        fscanf(file, "%d", &((*nodes_per_layer)[i]));
        if (i < (*num_layers - 1))fgetc(file); // ',' 읽기
    }
    fgetc(file); // '\n' 읽기
    *weight = (double***)malloc(sizeof(double**) * (*num_layers-1));
    for (int i = 0; i < *num_layers - 1; i++) {
        int cols = (*nodes_per_layer)[i + 1];
        (*weight)[i] = (double**)malloc(sizeof(double *) * (*nodes_per_layer)[i]);
        for (int j = 0; j < (*nodes_per_layer)[i]; j++) {
            (*weight)[i][j]=(double*)malloc(sizeof(double)*(*nodes_per_layer)[i+1]);
            for (int k = 0; k < (*nodes_per_layer)[i + 1]; k++) {
                fscanf(file, "%lf", &((*weight)[i][j][k]));
                if (k < (*nodes_per_layer)[i + 1] - 1) {
                    fgetc(file); // ',' 읽기
                }
            }
            fgetc(file); // '\n' 읽기
        }
    }
    fclose(file);
    printf("Weights loaded successfully from %s\n", filename);
    return;
}
```

Ⅶ.학습 중단과 학습률

7.1. 학습 중단 로직

학습 중단은 다양한 원인에 따라 발생한다. 먼저 가중치, 노드의 입출력 등에서 nan이 발생하는 경우이다. 이는 코드 작성 초기에 자주 발생하였으며, 이 경우 정상적인 학습이 불가능하기 때문에 중단한다.

학습을 진행함에 따라 error가 충분히 감소하는 적절한 하이퍼 파라미터를 적절히 도출하지 못하였다. 대부분의 상황에서 10% 내외의 정확도를 보이는 상황에서 error가 정체되는 현상이 발생하였다. 이에 error에 따른 학습 중단 코드는 일시적으로 작성하지 않았다.

때문에 측정 epoch에 학습을 중단하도록 하였다. 현재는 5000 epoch에 학습을 중단하도록 코드를 구성하였다.

7.2. 학습률

초기 학습률은 0.1%내외로 구성하여 0.001과 같은 값을 사용하였다. 이 경우 batch를 사용함에 따라 학습 속도가 현저히 떨어졌다. 이에 학습률을 1~10%까지 높여 0.01~0.1의 값을 학습률로 사용하였다. 이 경우 학습 속도가 증가하였지만 학습의 진행에 따라 학습률을 감소시킬 필요가 있었다.

현재는 1 epoch에 대한 $MSE \times 10$의 값을 학습률로 사용하고 이 값이 0.1을 넘는 경우 0.1을 학습률로 사용한다. 이를 통해 학습의 진행에 따라 감소하는 MSE에 따라 학습률도 감소하도록 하였다.

Ⅷ. 소스의 구성과 동작방법

8.1. 소스의 구성

메인 소스 코드는 training.c와 inference.로 구성된다. training.c는 학습을 위한 소스 코드이며, inference.c는 학습 모델을 통해 만들어진 가중치와 특정 이미지를 불러와 이미지의 텍스트를 추론하는 소스 코드이다.

8.2. training.c의 로직

training.c는 학습을 위한 소스코드로 다음과 같은 방식으로 동작한다

- 레이어, 노드의 수를 입력받고 랜덤의 가중치를 생성한다.
- Batch size만큼의 thread를 생성하고 순전파 함수를 할당한다.
- 순전파를 수행한다.
    - 이미지를 로드한다.
    - 이미지의 인덱스와 일치하는 값을 텍스트 파일에서 찾아 target을 로드한다.
    - 가중치와 활성화 함수를 사용하여 순전파를 수행한다.
- thread에 역전파 함수를 할당한다.
- 역전파를 수행한다.
    - 이미지와 함께 로드한 target 값과 출력 노드의 출력값을 역전파를 수행한다.
- 가중치를 수정한다.
- Training set 만큼의 데이터셋의 학습을 수행한다.
- thread에 순전파 함수를 할당한다.
- Test set 만큼의 데이터셋의 순전파를 수행한다.
- Test set에 대한 MSE를 계산한다.
- convert_VAR2IDX를 수행하여 이미지의 텍스트와 도출값의 텍스트를 비교하여 오차를 계산한다.
- 1 epoch을 종료하면 다시 학습을 수행한다. 이때 batch가 시작하는 인덱스를 달리하여 training set과 test set을 재배정한다.

8.3. inference.c의 로직

inference.c는 추정을 위한 소스코드로 다음과 같은 방식으로 동작한다.

- 이미지의 경로를 입력받아 이미지를 로드한다.
- 가중치 파일의 경로를 입력받아 가중치를 로드한다.
- 순전파를 수행한다.
- convert_VAR2IDX를 수행하여 출력 레이어의 출력값을 인덱스로 변환한다
- convert_IDX2Char를 수행하여 변환한 인덱스를 한글로 나타내도록 한다
- 추론한 텍스트를 출력한다.

8.4. 컴파일

컴파일은 다음과 같은 코드로 한다.

gcc training.c load_n_preprocess/image_load.c load_n_preprocess/preprocessing.c load_n_preprocess/load_data.c math_func/math_func.c create_network/create_layer_n_weight.c create_network/random_weight.c load_n_preprocess/load_target.c word_inference/word_inference.c csv_operate/export_weight_csv.c -I./load_n_preprocess -I./math_func -I./create_network -I./word_inference -I./typedef_struct -o training -lm -v

gcc inference.c load_n_preprocess/image_load.c load_n_preprocess/preprocessing.c load_n_preprocess/load_data.c math_func/math_func.c create_network/create_layer_n_weight.c create_network/random_weight.c load_n_preprocess/load_target.c word_inference/word_inference.c csv_operate/import_weight_csv.c csv_operate/export_weight_csv.c -I./load_n_preprocess -I./math_func -I./create_network -I./word_inference -I./typedef_struct -o inference -lm -v

8.5. 동작

동작 시 다음과 같은 코드를 볼 수 있다.

```
nabi@ijinseong-ui-MacBookPro training_c % ./training
How many layers? Maximum 16
8
How many Nodes in Hidden Layer 1? Maximum 256
16
How many Nodes in Hidden Layer 2? Maximum 256
16
How many Nodes in Hidden Layer 3? Maximum 256
16
How many Nodes in Hidden Layer 4? Maximum 256
16
How many Nodes in Hidden Layer 5? Maximum 256
16
How many Nodes in Hidden Layer 6? Maximum 256
16
1Epoch : 1 / accuracy : 0.000000 / 0.000000 / 0.330579 / MSE : 0.106293 / prev-last : -0.106293
6Epoch : 2 / accuracy : 0.000000 / 0.000000 / 0.308540 / MSE : 0.106131 / prev-last : 0.000162

Epoch : 3 / accuracy : 0.000000 / 0.000000 / 0.303030 / MSE : 0.105300 / prev-last : 0.000832
Epoch : 4 / accuracy : 0.000000 / 0.000000 / 0.173554 / MSE : 0.104112 / prev-last : 0.001188
Epoch : 5 / accuracy : 0.000000 / 0.000000 / 0.005510 / MSE : 0.100571 / prev-last : 0.003541
Epoch : 6 / accuracy : 0.000000 / 0.000000 / 0.000000 / MSE : 0.098393 / prev-last : 0.002178
Epoch : 7 / accuracy : 0.000000 / 0.000000 / 0.000000 / MSE : 0.091275 / prev-last : 0.007117
Epoch : 8 / accuracy : 0.000000 / 0.000000 / 0.000000 / MSE : 0.085315 / prev-last : 0.005961
Epoch : 9 / accuracy : 0.000000 / 0.000000 / 0.000000 / MSE : 0.080405 / prev-last : 0.004910
Epoch : 10 / accuracy : 0.000000 / 0.000000 / 0.000000 / MSE : 0.079649 / prev-last : 0.000756
```

‘은’ 이미지를 입력으로 넣은 결과

```
nabi@ijinseong-ui-MacBookPro training_c % ./inference
Enter Image File Path : text00081.bmp
Enter Weight's File Path : weight_8_16.csv
Weights loaded successfully from weight_8_16.csv
Character: 즌
```