**2022 병렬분산 프로젝트**

**선형함수로 이루어진 신경망으로   
이미지 분류하기**

팀 테니스라켓삽니다

IT정보공학과 202012178 김아은

IT정보공학과 202018392 박나현

**목차**

**1. 서론** ───────────────────── 3

**2. 본론**

2.1 데이터셋 ───────────────────── 3

2.2 모델 ───────────────────── 4

* 1. Flowchart ───────────────────── 5
  2. 코드 설명 ───────────────────── 6

**3. 결론**

3.1 결과물 ───────────────────── 9

3.2 프로젝트 수행 후기 ───────────────────── 10

1. **서론**

신경망 학습 시 처리해야할 데이터 양이 많을수록 학습 속도가 느려지고 메모리가 부족해진다.

대부분의 딥러닝 코드는 Python으로 구현된다. 하지만 Python은 다른 프로그래밍 언어에 비해 속도가 느리기 때문에 실시간 데이터에 대응하기 어렵다. 따라서 속도의 문제점을 개선하기 위해 실제 어플리케이션에서는 주로 C/C++를 사용한다.

이때 실제 사회에서도 GPU 사용 부분에 대한 고민이 필수적으로 일어난다. 우리도 이미지 분류를 위한 학습 과정에 많은 연산이 존재하여 이 부분에 대해 고민하고 CPU와 GPU에 따른 신경망 Neural Network 학습 속도를 비교하려 하였다.

**2. 본론**

**2.1 데이터셋(MNIST)**

**텍스트, 낱말맞추기게임이(가) 표시된 사진

자동 생성된 설명**

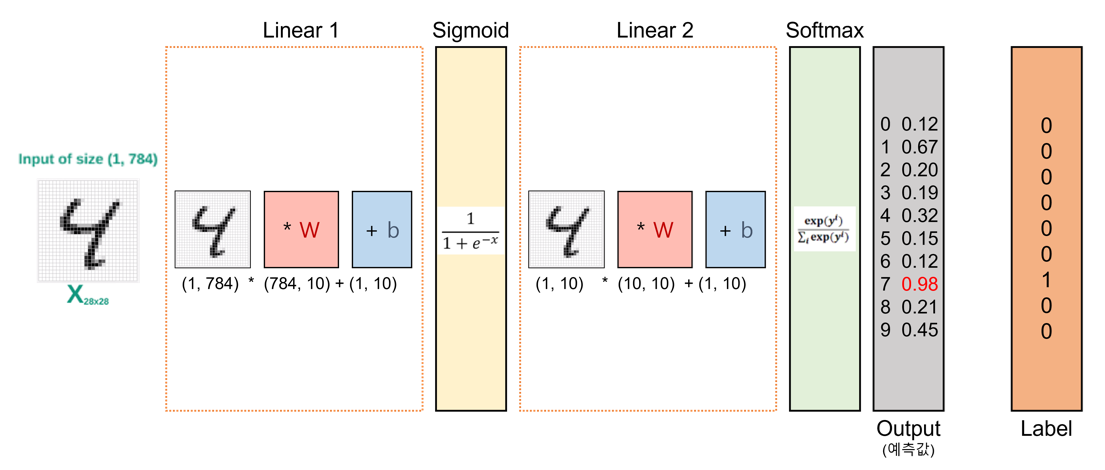
* MNIST는 computer vision dataset 중 하나로, 손으로 쓰여진 이미지로 구성된다.
* 이미지는 28x28 픽셀로 크기가 표준화되어 있다.
* 이번 프로젝트에 사용한 데이터 양은 총 42000개로, 자세한 내용은 다음과 같다.

- train data : 26880 개

- validation data: 8400 개

- test data: 6720 개

**2.2 모델**

****

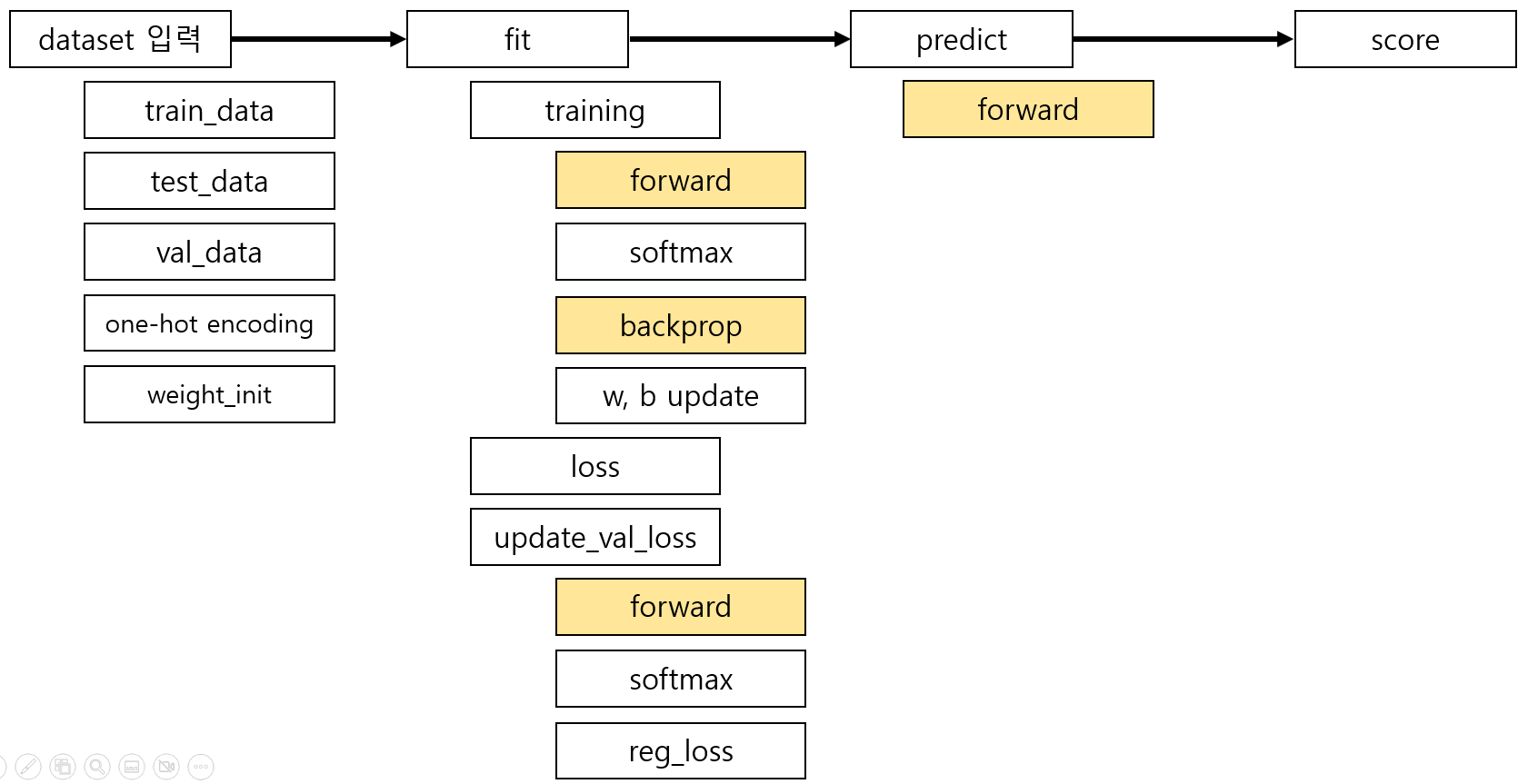
* input : 2차원 이미지를 1차원으로 변환한 값 사용

텍스트, 하얀색이(가) 표시된 사진

자동 생성된 설명

* Hidden layer : linear layer + sigmoid + linear layer
  + linear :
  + sigmoid :
* Output layer : softmax로 계산한 각 클래스별 확률로 클래스 예측
  + Softmax : , …, K
* Loss function : cross entropy
  + 정답 값과 예측 값 차이를 수치화해주는 손실함수,

**2.3 Flowchart**

****

* 표시된 노란 부분이 CPU와 GPU를 따로 구현한 부분이다.

**2.4 코드 설명**

**2.4.1 코드 흐름**

1) 데이터 전처리

train, validation, test 데이터는 각각 train.csv, val.csv, test.csv로 나뉘어 있다. main 함수에서 해당 데이터를 각각 불러와 label과 pixel 데이터를 따로 저장하였다.

y\_onehot\_encoded 함수에서는 불러온 label 데이터를 one-hot encoding으로 전처리 하였다. 만약 label이 3인 경우 (0, 0, 0, 1, 0, 0, 0, 0, 0, 0)로 변환된다.

2) weight initialization

kaiming\_init 함수에서는 linear layer에서 필요한 weight는 kaiming initialization 방법으로 초기화 하였다. 그 후 fit을 실행한다.

3) fit 🡪 training, update\_val\_loss

fit은 데이터를 한 개씩 불러와 training을 시키고 loss를 계산하여 validation까지 실행한다.

training 함수에서 모델을 데이터로 학습시키고 학습이 완료되면 loss를 계산하여 저장한다. 그 후 validation 데이터를 한 개씩 불러와 forward를 진행 후 loss를 계산하여 weight와 bias를 업데이트 시켜 더 정확한 성능을 내는 모델로 만든다.

training 함수에서는 forward, softmax, 오차 계산, backpropagation, weight와 bias 업데이트를 진행한다.

forward 함수로 linear 연산을 한 뒤에 softmax를 하여 확률로 만들어 준다. 그 후 실제 라벨 값과 예측 값을 비교하여 오차를 계산하고 backprop 함수로 grad를 계산한다. 계산된 grad를 가지고 weight와 bias 값을 업데이트 한다.

update\_val\_loss 함수는 validation을 진행하는 함수이다.

4) predict, score

predict 함수는 test 데이터로 학습된 모델을 가지고 예측 값을 도출한다.

score 함수를 통하여 예측된 값과 실제 값을 비교하여 정확도를 평가한다.

**2.4.2 병렬분산 처리가 필요한 연산 부분**

* 프로젝트에 사용된 연산은 행렬 합, 행렬 곱, 전체 합, sigmoid, softmax 등 많지만, 이 중 행렬 합, 행렬 곱, sigmoid 연산을 병렬분산 처리하였다. 그 이유는 (42000\*784)개의 많은 데이터가 사용되어 많은 시간이 소요될 것으로 예상하였다. 또한 이 연산들이 다른 연산에 비해 단순하다고 생각되었기 때문이다.
* 선택한 연산들이 사용되는 부분은 다음과 같다.
  + Forward 🡪 linear(dot, sum), sigmoid, loss 계산
  + Backpropagation 🡪 dot

1. dot

텍스트, 모니터, 실내, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. sum

텍스트이(가) 표시된 사진

자동 생성된 설명

1. sigmoid

텍스트이(가) 표시된 사진

자동 생성된 설명

1. gpu 사용예시

텍스트이(가) 표시된 사진

자동 생성된 설명

**3. 결론**

**3.1 결과물**

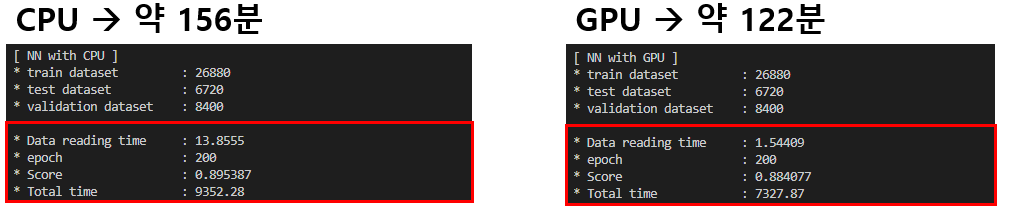
**3.1.1 epoch : 100**

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

데이터셋을 읽어오는 시간은 CPU, GPU가 각각 약 12초, 1초로 GPU가 약 12배 빨랐다. 하지만 전체 시간은 각각 약 77분, 60분으로 17분 차이가 났다.

**3.1.2 epoch : 200**



데이터셋을 읽어오는 시간은 CPU, GPU가 각각 약 13초, 1초로 GPU가 약 12배 빨랐다. 하지만 전체 시간은 각각 약 156분, 122분으로 34분 차이가 났다.

전체 실행 시간이 예상보다 큰 차이가 나지 않았다. 그 이유는 데이터를 하나씩 불러온 후 연산 과정에서만 GPU에 여러 번 접근하므로 메모리 접근 시간이 더 오래 걸렸던 것으로 예상된다.

**3.2 프로젝트 수행후기**

GPU를 효율적으로 사용하기 위해서는 많은 데이터 양과 단순한 연산 과정을 필요로 한다. 이미지 분류 계산은 이러한 조건을 만족했기에 프로젝트 주제로 선정하였다. 하지만 GPU에서는 데이터를 불러올 때마다 메모리 접근 시간이 소요되는데, 이로 인해 우리의 프로젝트 결과에서는 많은 차이를 보이진 않았던 것 같다. GPU를 사용하기 위해서는 코드를 구현하기 전에 GPU에 할당해줄 부분에 대한 구상을 미리 진행한 후 코드를 구현해야 한다는 사실을 알게 되었다.

사실 CPU 코드는 깃허브나 블로그 등에서 참고할 수 있을 것 같다고 예상되어 GPU 코드에 힘을 많이 써서 여러가지 조합으로 성능 비교를 해보려고 하였다. 그런데 예상과 달리 대부분 Python과 여러 라이브러리를 사용해서 구현한 것이 대다수였다. 우리는 이번 프로젝트를 라이브러리 없이, 오직 C/C++로만 모든 과정을 구현하려고 목표를 세웠기 때문에 참조할 자료가 없어 직접 모든 단계를 구현하였다.

시간 관계상 여러 조합으로 해보진 못했지만, 메모리에 접근하는 횟수가 많기 때문에 다른 여러 연산에 GPU 코드를 설정하면서 성능을 비교해 보면 좋을 것 같다.