딥 러닝을 이용한 최적 수행 시간의 정렬 알고리즘 선택 모델

Sorting Algorithm Choosing Model With Optimum Time Complexity Using Deep Learning

김홍식, Hanyang University, Department of Computer Software

Abstract

우리는 배열의 값들을 정렬하기 위해 다양한 알고리즘을 사용한다. 이들 중 시간 복잡도가 최적인 알고리즘은 시간 복잡도가 $O(n\log n)$인 Heap Sort, Merge Sort, Quick Sort 등으로 알려져 있는데, 이들 알고리즘 사이에서도 자료가 정렬되어 있는 형태에 따라 최적 알고리즘이 달라진다. 또한 데이터가 거의 정렬되어 있는 경우에는 시간 복잡도가 $O(n\log n)$인 알고리즘이 시간 복잡도가 $O(n^2)$으로 알려져 있는 알고리즘보다도 정렬 시간이 오래 걸릴 수 있다. 이 논문에서는 CNN(Convolutional Neural Network)에 기반한 딥 러닝을 이용하여 배열의 정렬 상태에 따라 정렬 알고리즘의 수행 시간을 분석하고, 이를 토대로 새로운 배열이 입력되었을 때 직접 정렬을 수행하기 전에 각 알고리즘의 수행 시간을 추정한다. 이때 입력 데이터는 배열에서 각 값과 그 값의 배열에서의 인덱스(위치)를 좌표평면에 나타낸 일정한 크기의 그림으로, 출력 데이터는 각 정렬 알고리즘의 수행 시간으로 한다. 지금까지 조사한 바에 따르면, 본 연구는 딥 러닝을 이용하여 정렬 알고리즘을 선택하는 방법에 대한 최초의 연구이다. 이 논문에서는 절대적인(absolute) 관점 및 상대적인(relative) 관점에서, 무작위, 대략적인 일차함수 형태, 대략적인 이차함수 형태의 3가지 형태로 나타난 이들 sparse matrix에 대하여 어떤 알고리즘을 사용하는 것이 최적인지를 비교 횟수, 이동 횟수, 총 비용의 관점에서 알아보았다.

I. Introduction

정렬 알고리즘은 컴퓨터 과학 분야에서 오랫동안 논의되어 왔던 주제 중 하나이다. Insertion Sort, Selection Sort, Bubble Sort와 같이 우리가 쉽게 생각할 수 있지만 수행 시간이 $O(n^2)$로 비교적 오래 걸리는 알고리즘이 있는 반면, Heap Sort, Merge Sort, Quick Sort와 같이 비교적 생각하기 어렵지만 시간 복잡도가 $O(n\log n)$로 비교적 짧은 알고리즘 역시 존재한다. 이외에도 특수한 경우에 적용 가능한 Radix Sort, 특수한 경우에만 사용할 수 있지만 시간 복잡도가 $O(n)$에 가까운 Counting Sort 등이 존재한다. 여기서 문제점이 어떤 정렬 알고리즘을 주어진 배열에 적용하는 것이 시간이 가장 적게 걸리는지인데, 시간 복잡도가 $O(n\log n)$인 알고리즘이 Heap Sort, Merge Sort, Quick Sort 등이 일반적으로 시간이 가장 적게 걸리지만, 거의 정렬된 형태의 데이터에서는 본 논문에서 실험한 바와 같이 Insertion Sort와 같이 시간 복잡도가 $O(n^2)$으로 알려진 알고리즘이 시간 복잡도가 $O(n\log n)$인 알고리즘보다 수행 시간이 적게 걸리기도 한다. 본 논문에서는 배열을 구성하는 각 값의 배열 내에서의 인덱스(위치)에 따른 값의 크기의 분포 형태를 배열의 정렬 상태라고 하고, 정렬 상태에 따라 최적의 정렬 알고리즘을 선택하는 문제를 정렬 알고리즘 선택 문제라고 하자.

이 논문에서는 정렬 알고리즘 선택 문제를 해결하기 위해 배열의 정렬 상태에 따라 어떤 알고리즘을 선택하는 것이 최적인지를 딥 러닝을 이용하여 알아내려고 한다. 즉 딥 러닝을 이용해 배열의 정렬 상태에 따른 알고리즘별 수행 시간에 대한 데이터를 학습하고, 이를 토대로 새로운 배열에 대해 그 정렬 상태를 파악하여 정렬을 수행하기 전에 그 수행 시간을 추정하는 것이다. 수행 시간이 가장 짧은 알고리즘을 최적 알고리즘이라고 정의하고, 비교 연산 횟수와 이동 연산 횟수의 합이 수행 시간에 비례한다고 가정하며, 절대적인(absolute) 관점 및 상대적인(relative) 관점에서 어떤 알고리즘을 사용하는 것이 최적인지를 알아낸다. 여기서 절대적인 관점이란 모든 알고리즘에 대해 수행 시간을 일정한 평균과 일정한 표준편차 값을 이용하여 표준화하는 것으로,

실질적으로 주어진 각 배열에 대해 어떤 정렬 알고리즘의 수행 시간이 가장 짧은지를 나타낸다. 상대적인 관점이란 각 알고리즘에 대해 해당 알고리즘을 적용했을 때의 수행 시간의 평균과 표준편차를 이용하여 표준화하는 것으로, 주어진 각 배열의 형태가 학습 데이터로 주어진 모든 형태 중에서 해당 알고리즘을 적용하기에 상대적으로 얼마나 적합한지를 나타낸 것이다.

기존에도 정렬 알고리즘에 대한 다양한 연구들이 있었지만, 대부분 특정 분야 또는 상황에서 적용할 수 있는 정렬 알고리즘을 제안하거나, 또는 기존의 알고리즘을 결합하거나 수정하여 새로운 정렬 알고리즘을 제안하는 것이 대부분으로, 딥 러닝 기술이 이용되지는 않았다. 이 논문은 이들과 달리 딥 러닝을 적용하여 어떤 알고리즘이 최적인지를 연구한다는 점이 가장 큰 차별점이다.

본 논문에서는 딥 러닝을 통해 배열의 정렬 상태에 따른 각 정렬 알고리즘의 수행 시간을 학습하고, 새로운 배열에 대해서 이를 추정하여 실제 수행 시간과 비교한 오차를 계산하고, 수행 시간이 가장 짧은 알고리즘을 얼마나 잘 선택하는지를 정확도(accuracy)를 통해 평가하는 실험을 앞서 언급한 절대적인(Absoulte) 관점과 상대적인(Relative) 관점에 따라 진행하였다.

본 연구는 정렬 알고리즘 관련 연구 및 정렬 알고리즘을 이용해야 하는 실무적 상황에 다음과 같이 기여할 수 있다.

- 절대적인(Absolute) 관점을 이용하는 경우를 생각해 보자. 배열의 정렬 상태를 나타내는 이미지를 $O(n)$의 시간 동안에 생성할 수 있고, 이미지의 크기는 배열의 값의 개수에 관계없이 일정하기 때문에 이미 학습된 모델에 이 이미지를 입력했을 때 어떤 알고리즘이 최적인지를 도출하는 데 걸리는 시간은 $O(1)$이므로 결론적으로 $O(n)$의 시간 안에 그 배열에 대해 어떤 정렬 알고리즘을 적용하는 것이 최적인지를 알아낼 수 있다. 일반적인 경우 데이터를 정렬하는 데 걸리는 최소 시간이 $O(n\log n)$으로 알려져 있으므로, 딥 러닝 방법론 없이 수행 시간이 $O(n\log n)$인 특정한 정렬 알고리즘을 사용한다고 가정했을 때 이보다 수행 시간이 $O(n)$ 이상 적은 정렬 알고리즘이 존재할 가능성이 있고, 따라서 $O(n)$ 이상의 수행 시간을 절약할 수 있을 것이므로 결과적으로 정렬에 수행되는 시간을 단축할 수 있다.
- 상대적인(Relative) 관점을 이용하는 경우에는 각 알고리즘에 비교적 적합한/적합하지 않은 배열의 정렬 상태를 알 수 있으므로, 특정한 정렬 상태의 배열이 주어졌을 때 정렬을 직접 수행하지 않고 각 정렬 알고리즘의 수행 시간을 딥 러닝으로 추정하여 각 정렬 상태에 각 알고리즘이 적합한지 알아낼 수 있다. 이 경우 절대적인 관점을 이용할 때와 같이 모델을 이용하여 최적 알고리즘을 추정하는 데 $O(n)$의 시간이 필요하므로, 정렬을 수행할 때 $O(n\log n)$ 또는 $O(n^2)$의 시간이 걸리는 것에 비해 상당한 양의 시간을 절약할 수 있을 것이다. 따라서 정렬 알고리즘에 대한 향후 연구에서 알고리즘의 수행 시간을 고려한 적합성을 대략적으로 추정하는 데에도 도움이 될 것이다.

II. Related Works

Curtis R. Cook과 Do Jin Kim [1] 은 거의 정렬된 리스트에서 최적의 정렬 알고리즘을 찾는 연구를 진행하였다. Straight Insertion Sort, Quicker Sort, Shell Sort, Merge Sort, Heap Sort라는 5가지 알고리즘을 비교 분석하였으며, Straight Insertion Sort와 Quicker Sort를 merging 방법과 결합한 새로운 정렬 알고리즘을 제안하였다. Aditya Dev Mishra와 Deepak Garg [2] 는 여러 가지 정렬 알고리즘을 Comparison Based Sort와 Non-Comparison Sort로 구분하여 수행 시간을 평균적인 경우(Average Case)와 최악의 경우(Worst Case)로 구분하여 정리하였고, 이를 기반으로 각 알고리즘의 장단점을 정리하였으며, 여러 가지 문제 상황에 대해 어떤 정렬 알고리즘이 적합한지를 정리하였다. D.T.V. Dharmajee Rao와 B.Ramesh [3] 는 각 정렬 알고리즘을 pseudo code를 이용하여 정의하고, 배열의 크기에 따라 각 정렬 알고리즘이 실제로 수행되는 데 걸리는 시간을 측정하여 정리하였다.

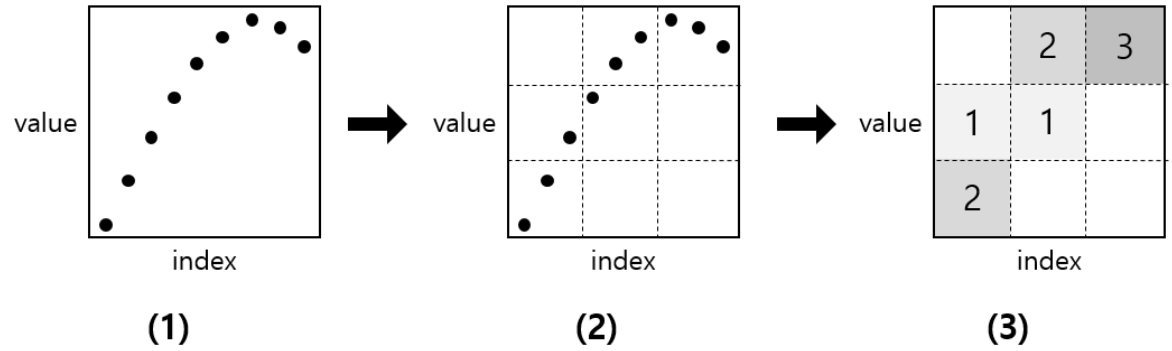

Figure 1. Array State Image를 생성하는 알고리즘. (1) 은 배열에서 각 값의 index(위치)와 함께 그 값(value)을 나타낸 것이고, (2) 는 이를 N*M개의 grid로 분할한 것이다. (2) 의 각 grid에 있는 점의 개수를 계산하여 (3) 과 같은 이미지를 생성한다.

[1] 은 거의 정렬된 리스트라는 특정 형태의 배열로 한정한 반면 이 논문에서는 이것을 포함한 다양한 형태의 배열에서 어떤 알고리즘이 최적인지를 알아내고, [2] 는 단순히 각 알고리즘의 시간 복잡도와 장단점, 그리고 어떤 문제 상황에서 쓰이는지를 정리한 반면, 이 논문에서는 딥 러닝을 통해 이것을 알아낸다. [3] 은 [2] 와 유사하며, 시간 복잡도 대신 각 알고리즘의 수행 시간을 측정하여 정리하였다.

III. Proposed Model and Experiment Design

III-1. Overview

본 논문에서는 배열의 정렬 상태를 그림으로 나타낸 것(Array State Image)을 입력 데이터로, 각 알고리즘별 수행 시간을 출력 데이터로 하여 Convolutional Neural Network (CNN) [4] 을 이용하여 배열의 정렬 상태에 따른 알고리즘의 수행 시간을 학습하고, 이를 토대로 테스트용 Array State Image에 대하여 알고리즘의 수행 시간을 얼마나 잘 예측하고 적절한 알고리즘을 얼마나 잘 선택하는지를 평가하는 실험을 진행한다.

III-2. Input Data - Array State Image

본 논문의 모델에서는 입력 데이터로 배열의 정렬 상태를 그림으로 나타낸 이미지를 이용한다. 배열의 정렬 상태를 그림으로 나타내기 위한 알고리즘은 **Figure 1.** 과 같다. **Figure 1.** 을 보면, (1) 은 해당 배열에서 각 값의 배열 내에서의 index와 함께 그 값의 크기를 좌표평면상에 나타낸 것이다. 이때 좌표평면에 표현되는 값의 범위는 index (x축)의 경우 0부터 (배열에 있는 값의 개수)-1 까지이고, value (y축)의 경우 해당 배열의 모든 값 중 최솟값부터 최댓값까지로 한다. 이 좌표평면에서 해당 값의 index가 클수록 오른쪽에 위치하고, 값이 클수록 위쪽에 위치한다. 이제 이 좌표평면을 (2) 와 같이 index 구간과 value 구간의 크기가 모두 서로 같은, 즉 x축과 y축의 간격이 모두 일정한 N*M개의 grid로 나눈다. 마지막으로 (2) 에 있는 각 grid에 몇 개의 값이 속하는지를 (3) 과 같은 행렬을 이용하여 나타내고, 이것을 입력 데이터로 사용할 이미지로 지정한다. 이때 이 행렬의 각 원소는 (2) 의 각 grid에 해당한다.

III-3. Output Data - Performance Time of Each Algorithm

본 논문의 모델에서는 출력 데이터로 각 배열에 각 정렬 알고리즘을 적용했을 때의 비교 연산 및 이동 연산의 수행 횟수를 다음과 같이 변환한 값을 이용한다. Absolute 관점의 경우에는 모든 알고리즘에 대해 일정한 평균 및 표준편차를 이용하여 표준화하며, 이 값은 실제 평균 및 표준편차와는 무관하다. Relative 관점에서는 각 알고리즘에 대해 학습 데이터에 해당 알고리즘을 적용했을 때의 연산 수행 횟수의 평균과 그 표준편차를 이용하여 표준화한다. 여기서 각 배열의 Arr-

알고리즘	a_0	a_1	a_2	a_3	a_4	a_5
	Bubble Sort	Heap Sort	Insertion Sort	Merge Sort	Quick Sort	Selection Sort
평균 시간	$O(n^2)$	$O(n\log n)$	$O(n^2)$	$O(n\log n)$	$O(n\log n)$	$O(n^2)$
최악 시간	$O(n^2)$	$O(n\log n)$	$O(n^2)$	$O(n\log n)$	$O(n^2)$	$O(n^2)$

Table 1. 각 알고리즘이 어떤 알고리즘을 의미하는지와 평균 및 최악 수행 시간 [2] 을 나타낸다.

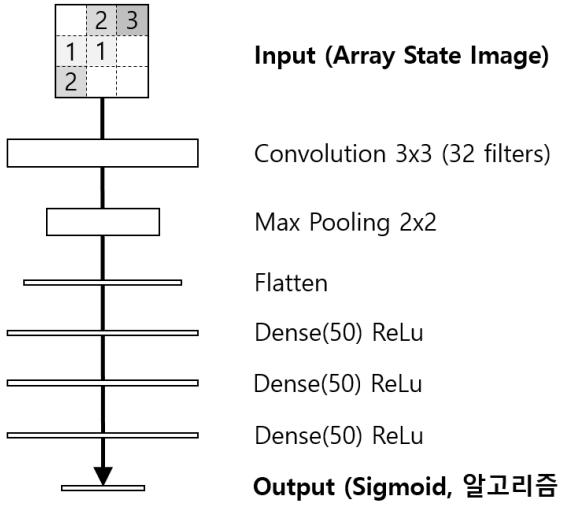

Figure 2. 입력 데이터인 Array State Image가 주어졌을 때 알고리즘 수행 시간을 학습하는 딥 러닝 모델을 나타낸다.

ay State Image와 그 배열에 대한 각 알고리즘별 수행 시간의 쌍을 하나의 데이터 쌍이라고 할 때, 데이터 쌍의 개수를 N이라고 하자. 각 데이터 쌍 data_i $(i = 0,1,\dots,N-1)$ 에 대해 ops_{A,i,a_j}는 알고리즘 a_j를 적용했을 때의 Absolute 관점에서의 원래 수행 횟수, ops_{R,i,a_j}는 알고리즘 a_j를 적용했을 때의 Relative 관점에서의 원래 수행 횟수라고 하면 ops_{A,i,a_j}, ops_{R,i,a_j}를 각각 표준화한 값 ops'_{A,i,a_j}, ops'_{R,i,a_j}는 각각 (1), (2)와 같다. 여기서 avg_{R,a_j}, std_{R,a_j}는 각각 Relative 관점에서 학습 데이터에 대해 알고리즘 a_j를 수행했을 때의 원래 수행 횟수의 평균과 표준편차를 나타낸다.

$$\mathrm{ops}'_{A,i,a_j} = \frac{\mathrm{ops}_{A,i,a_j} - 50000}{50000} \quad \dots(1)$$

$$\mathrm{ops}'_{R,i,a_j} = \frac{\mathrm{ops}_{R,i,a_j} - \mathrm{avg}_{R,a_j}}{\mathrm{std}_{R,a_j}} \quad \dots(2)$$

각 데이터 쌍에 대하여 출력 데이터는 각 알고리즘을 적용했을 때의 수행 시간, 즉 Absolute 관점의 경우 $\mathrm{ops}'_{A,i,a_j}(j = 0,1,\dots,5)$의 6개의 값, Relative 관점의 경우 $\mathrm{ops}'_{R,i,a_j}(j = 0,1,\dots,5)$의 6개의 값이며, 여기서 각 알고리즘 a_j는 **Table 1.** 과 같다.

III-4. Deep Learning Model

본 논문에서 사용된 딥 러닝 모델은 **Figure 2.** 와 같다. 입력 데이터는 크기가 N*M인 이미지이고, 여기서 3x3 Convolution, 2x2 Max Pooling을 통해 Feature를 추출한 후 Dense Layer를 통해 출력값, 즉 각 알고리즘의 수행 시간을 표준화한 값을 출력한다. 본 논문에서 수행 시간을 측정할 알고리즘이 **Table 1.** 에 나타난 것과 같이 6개이므로 출력 데이터는 총 6개의 숫자 값으로 이루어진다. Optimizer는 Adam Optimizer [5] 를 사용하였으며 learning rate는 0.001이다.

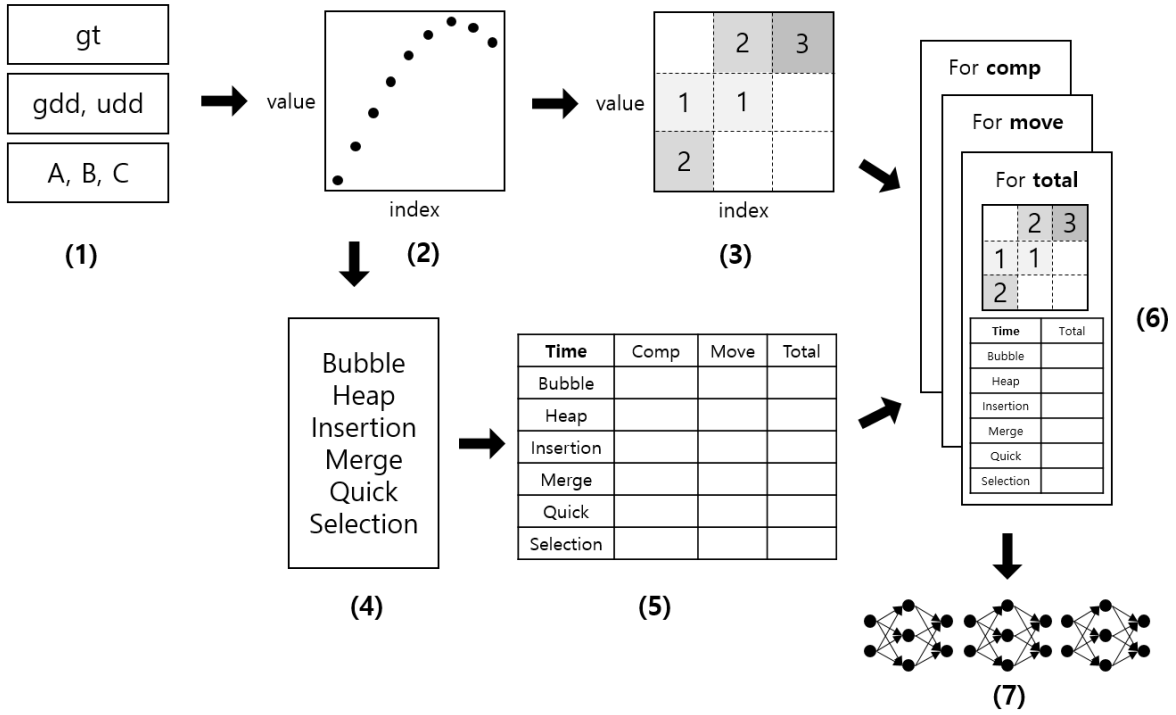

Figure 3. 학습 및 테스트용 데이터 쌍의 전체적인 생성 및 학습 과정을 나타낸다. (1)은 배열 생성을 위한 변수, (2)는 배열의 각 값에 대하여 index와 value를 표시한 것, (3)은 (2)에 대한 Array State Image, (4)는 본 논문에 사용되는 알고리즘, (5)는 알고리즘별 수행 시간을 표준화한 값, (6)은 (3)을 입력 데이터로, (5)를 출력 데이터로 하는 하나의 데이터 쌍, (7)은 **Figure 2.** 에 나타난 딥러닝 모델을 나타낸다.

```
//// gt (그래프의 형태) 결정 (0은 상수함수, 1은 일차함수, 2는 이차함수)
// i는 학습 데이터 쌍의 번호(0~1499) 또는 테스트 데이터 쌍의 번호(0~499)
// training은 학습용 데이터 쌍을 생성하는 경우 True, 테스트 데이터 쌍을 생성하는 경우 False
if training: // 학습용 데이터 쌍을 생성할 때
  if i < 300: gt = 0;
  else if i < 750: gt = 1;
  else: gt = 2;
else: // 테스트용 데이터 쌍을 생성할 때
  if i < 100: gt = 0;
  else if i < 250: gt = 1;
  else: gt = 2;

//// 변수 gdd, udd, A, B, C의 값 결정
// random(a, b)는 a 이상 b 이하의 랜덤한 실수를 반환
gdd = random(0, 50); udd = random(0, 50); // gdd는 가우시안 분포의 표준편차, udd는 유니폼 분포의 편차
A = random(-1, 1); B = A * random(-2, 0); C = random(0, 1);

//// 값이 결정된 변수 gdd, udd, A, B, C를 이용하여 배열에 데이터 입력
// y는 index 0~499까지 정의된 배열로, 주어진 배열 인덱스 x에 대하여 그 인덱스에 있는 값을 나타냄
for x in 0, 1, …, 499:
  // Gaussian(avg, std)는 평균이 avg이고 표준편차가 std인 정규분포에서 임의의 값 반환
  if gt == 0: y[x] = udd * random(-1, 1) + Gaussian(avg=0, std=gdd);
  else if gt == 1: y[x] = a*x + b + udd * random(-1, 1) + Gaussian(avg=0, std=gdd);
  else if gt == 2: y[x] = a*x*x + b*x + c + udd * random(-1, 1) + Gaussian(avg=0, std=gdd);

//// 최종적으로 도출된 배열 y를 반환
return y;
```

Algorithm 1. 학습 및 테스트용 데이터 쌍을 생성하기 위한 배열을 생성하는 알고리즘

IV. Experimental Results

IV-1. Dataset

학습 및 테스트용 데이터 쌍의 생성 과정은 **Figure 3.** 과 같은데, 이때 Absolute 관점과 Relative 관점에 대하여 서로 다른 데이터 쌍들을 독립적으로 생성한다. 먼저 (1) 에서 지정한 변수를 이용하여 **Algorithm 1.** 을 통해 학습용 데이터 쌍 1500개와 테스트용 데이터 쌍 500개를 생성하기 위한 배열을 생성한다. 즉, 각 배열에는 랜덤한 Gaussian 및 Uniform 분포를 갖는 오차가 있는 상수함수, 일차함수 또는 이차함수 형태로 총 500개의 값이 존재하며, 학습 데이터와 테스트 데이터 모두 상수함수, 일차함수, 이차함수 형태의 정렬 상태를 갖는 배열은 각각 전체의 20%, 30%, 50%를 차지한다. 그 다음 이들 배열을 이용하여 **Figure 1.** 에 나타난 과정을 통해 (3)과 같은 Array State Image를 생성한다. 이때 grid의 개수는 10x10으로 하며, 따라서 학습 데이터로 사용할 이미지의 크기 역시 10x10이다. 또한 이들 배열에 각 정렬 알고리즘을 적용하여 도출된 수행 시간을 **III-3**에서와 같이 표준화한 결과값을 도출하는데, 이때 (5)와 같이 Comp (비교 연산 횟수), Move (이동 연산 횟수), Total(비교 연산 횟수 + 이동 연산 횟수)로 구분한다. 이렇게 생성된 (3)과 (5)에 대해 (3)을 입력 데이터, (5)를 출력 데이터로 하는 입-출력 데이터 쌍을 (6)과 같이 생성하는데, 이때 하나의 입력 데이터 I_i $(i = 0,1,\dots)$에 대해 Comp, Move, Total을 각각 표준화한 값을 출력 데이터로 하는 총 3개의 출력 데이터 $O_{i,C}$, $O_{i,M}$, $O_{i,T}$를 각각 생성하여, 입-출력 데이터 쌍 $(I_i, O_{i,C})$, $(I_i, O_{i,M})$, $(I_i, O_{i,T})$을 생성한다. Comp, Move, Total을 각각 표준화한 값을 출력 데이터로 하는 신경망을 각각 NN_C, NN_M, NN_T라고 할 때, 이들 데이터 쌍에 대해 Comp, Move, Total을 표준화한 것을 출력 데이터로 하는 경우 각각 NN_C, NN_M, NN_T을 통해 학습 또는 테스트를 진행한다. 테스트 데이터인 경우 해당 신경망에 (3) 만을 입력하여 도출된 결과값을 (5)와 비교하여 **IV-2**에 따라 오차를 계산한다.

IV-2. Metrics

본 논문에 적용된 모델의 테스트 결과의 오차 계산 및 정확도 평가에 사용되는 Metric은 Mean Absolute Error (MAE), Mean Squared Error (MSE), Accuracy의 총 3가지로, 각각 **MAE**, **MSE**, **ACC**라고 하면 전체 테스트 데이터에 대해 각각 **(3)**, **(4)**, **(5)**와 같이 계산한다. 여기서 $O_{i,j}$, $P_{i,j}$는 **IV-1**의 학습-테스트 데이터 쌍 $D_i (i = 0,1,\dots I-1)$에 대하여 **Table 1**에 나타난 정렬 알고리즘 $a_j (j = 0,1,\dots J-1)$을 적용했을 때의 표준화된 출력값과 그 출력값에 대한 예측값을 의미한다. 여기서는 테스트 데이터가 총 500개이고 알고리즘이 총 6종류이므로 $I = 500$, $J = 6$이다.

$$MAE = \sum_{i=0}^{I-1}\sum_{j=0}^{J-1}\left|O_{i,j} - P_{i,j}\right| \quad \dots (3)$$

$$MSE = \sum_{i=0}^{I-1}\sum_{j=0}^{J-1}\left(O_{i,j} - P_{i,j}\right)^2 \quad \dots (4)$$

$$ACC = \frac{1}{I}\sum_{i=0}^{I-1} E_i \quad where \quad E_i = \begin{cases} 1 & \text{if } \underset{j}{\operatorname{argmin}} P_{i,j} = \underset{j}{\operatorname{argmin}} O_{i,j} \\ 0 & \text{otherwise} \end{cases} \quad \dots (5)$$

IV-3. Algorithm and Code Details

본 논문에서 사용한 정렬 알고리즘은 [6] 에 있는 Python 코드를 이용하였으며, 해당 코드에서 값을 비교하는 부분에 대해서는 비교 연산 횟수 1회, 배열의 값을 서로 바꾸거나 배열에 값을 할당하는 등 배열 수정 부분에 대해서는 이동 연산 횟수 1회로 처리하여 계산하였으며, 총 연산 횟수는 비교 연산 횟수와 이동 연산 횟수의 합으로 계산하였다. 본 논문 작성을 위해 Python으로 코드를 작성하였으며, numpy [7], Tensorflow [8], Matplotlib [9] 라이브러리를 사용하였다. 또한 각 신경망 NN_C, NN_M, NN_T에 대한 학습 수행 시 epoch는 1000회로 설정하였다.

	MAE	MSE	Accuracy
Comp (비교 횟수)	0.007183	0.000170	96.40%
Move (이동 횟수)	0.009400	0.000255	99.80%
Total (총 연산 횟수)	0.014330	0.000718	82.60%

Table 2. Absolute 관점에서의 전체 테스트 데이터에 대한 정확도

예측/실제	a_0	a_1	a_2	a_3	a_4	a_5
a_0	**0**	0	0	0	0	0
a_1	0	**127**	1	0	78	0
a_2	0	0	**10**	0	2	0
a_3	0	0	0	**0**	0	0
a_4	0	6	0	0	**276**	0
a_5	0	0	0	0	0	**0**

Table 3. Absolute 관점에서 총 연산 횟수가 최소인 알고리즘에 대한 예측 및 실제 (전체 배열 500개)

	MAE	MSE	Accuracy
Comp (비교 횟수)	0.100988	0.038674	88.80%
Move (이동 횟수)	0.091145	0.020817	60.40%
Total (총 연산 횟수)	0.105595	0.041698	73.20%

Table 4. Relative 관점에서의 전체 테스트 데이터에 대한 정확도

예측/실제	a_0	a_1	a_2	a_3	a_4	a_5
a_0	**71**	0	0	2	9	0
a_1	0	**154**	0	1	4	1
a_2	67 (A)	0	**0**	1	8	0
a_3	4	2	0	**24**	0	0
a_4	11	3	0	0	**115**	11
a_5	1	0	0	0	9	**2**

Table 5. Relative 관점에서 총 연산 횟수가 최소인 알고리즘에 대한 예측 및 실제

IV-4. Experimental Results

Table 2는 Absolute 관점에서, **Table 4**는 Relative 관점에서의 전체 테스트 데이터에 대한 MAE, MSE, Accuracy를 Comp (비교 연산 횟수), Move (이동 연산 횟수), Total (총 연산 횟수)에 대하여 각각 나타낸 것이다. **Table 3**은 Absolute 관점에서, **Table 5**는 Relative 관점에서의 총 연산 횟수가 최소가 되는 알고리즘을 예측한 결과와 실제로 수행 시간이 최소인 알고리즘을 나타낸 것으로, **Table 3**과 **Table 5**에서 각 가로축은 예측된 데이터 쌍, 각 세로축은 실제 데이터 쌍의 수를 의미한다. 예를 들어 **Table 3**에서 행 a_1과 열 a_4가 교차하는 셀의 값이 78이므로, 총 연산 횟수가 최소가 되는 알고리즘이 a_1로 예측되었지만 실제로는 a_4인 데이터 쌍의 개수는 78개이다. 배열의 index에 따라 값이 나타나는 각 형태(상수함수, 일차함수, 이차함수)에 따른 최적 알고리즘의 예측값 및 실제값에 대한 결과는, 배열의 정렬 형태에 따라 어떤 알고리즘이 적합한지를 논의하기보다는 딥 러닝 모델을 통해 이를 예측하는 모델을 소개하는 것이 본 논문의 주요 목적이고, 배열의 정렬 형태를 이 3가지 카테고리보다는 수많은 다양한 카테고리로, 그리고 알고리즘의 종류를 본 논문에서 제시한 6가지 외에도 더 다양하게 확장하여 일반화하는 것이 본 논문과 관련된 미래의 연구 과제이므로 소개하지 않는다.

V. Discussion

비교 연산 횟수, 이동 연산 횟수, 총 연산 횟수 모두에 대하여, MAE 및 MSE가 Absolute 관점보다 Relative 관점에서 모두 높게 측정된 반면 Accuracy는 Absolute 관점에서 높게 측정되었는데, 이것은 비교 연산, 이동 연산, 총 연산 횟수 모두에 대하여 MAE, MSE, Accuracy 기준으로 볼 때 Absolute 관점에서의 성능이 Relative 관점에서의 성능보다 높게 측정되었다는 것을 의미한다. Absolute 관점의 경우 각 정렬 알고리즘의 절대적인 연산 횟수를 측정하는데, 이때 일반적으로 수행 시간이 $O(n^2)$에 해당하는 Bubble Sort, Insertion Sort, Selection Sort에 비해 수행 시간이 $O(n\log n)$에 해당하는 Heap Sort, Merge Sort, Quick Sort의 수행 시간이 짧으므로 보통 후자에 해당하는 3가지 알고리즘 중에서 최적의 알고리즘이 결정되고, 그렇지 않은 극히 일부의 경우 모두 Insertion Sort가 최적 알고리즘이다. 따라서 최적 정렬 알고리즘으로 결정될 수 있는 알고리즘이 비교적 한정적이다. 그러나 Relative 관점의 경우 상대적으로, 즉 정렬 상태의 차이가 있는 다른 배열들과 비교하여 각 알고리즘이 그 배열에 얼마나 최적인지를 고려하기 때문에, 어떤 알고리즘이든 절대적인 연산량에 관계없이 그 알고리즘에 최적인 정렬 형태를 갖는 배열일수록 다른 알고리즘보다 최적 알고리즘으로 선택될 가능성이 높다. 따라서 최적 알고리즘으로 결정될 수 있는 알고리즘이 비교적 다양하게 나타날 수 있다.

또한 Absolute 관점에서 총 연산 횟수를 살펴보면, Heap Sort가 최적 알고리즘일 것으로 추정했으나 실제 최적 알고리즘은 Quick Sort였던 경우가 많은데, 이것은 Heap Sort와 Quick Sort의 실제 절대적 연산량의 차이가 적어서 약간의 오차가 발생해도 최적 알고리즘에 대한 추정이 바뀔 가능성이 높다. Quick Sort가 최적 알고리즘일 것으로 추정했으나 실제 최적 알고리즘이 Heap Sort인 경우는 이에 비해 훨씬 적은데, 그 이유는 테스트 데이터에 대한 Quick Sort의 절대적인 연산량의 평균이 Heap Sort에서의 그것보다 작아서 Quick Sort가 실제로 최적 알고리즘일 가능성이 더 높기 때문이다.

Relative 관점에서 실제로 최적인 알고리즘이 Insertion Sort로 판정된 경우가 없는데, 이것은 Relative 관점에서 Bubble Sort와 Insertion Sort의 총 연산 횟수를 표준화한 값이 모든 배열에서 동일하게 나타나서, 이들 모두에 대해 최적 알고리즘이 이름순으로 앞에 있는 Bubble Sort로 판정되었기 때문이다. 이 문제가 발생한 원인은 테스트 데이터 $D_i(i=0,\dots,499)$에 대해 Bubble Sort에서의 비교 및 이동 연산 횟수를 각각 BC_i, BM_i라고 하고, Insertion Sort에서의 비교 및 이동 연산 횟수를 각각 IC_i, IM_i라고 하면 $BC_i = c, BM_i = IM_i = IC_i - 500$ (c는 상수) 의 관계가 항상 성립하고, 따라서 Bubble Sort와 Insertion Sort에서의 총 연산 횟수를 각각 BT_i, IT_i라고 하면 $BT_i = BC_i + BM_i = c + IC_i - 500$, $IT_i = IM_i + IC_i = 2IC_i - 500$로 모두 IC_i에 대한 일차식으로 나타나므로, 이것들을 평균과 표준편차를 이용하여 표준화한 값은 모든 테스트 데이터에 대해서 서로 동일하게 나타난다. 이를 고려하여 최적 알고리즘에 대한 예측이 Insertion Sort이고 실제 최적 알고리즘이 Bubble Sort로 판정된 경우 (A) 에 대해 예측값과 실제 최적 알고리즘이 일치한다고 판정하면 Accuracy는 86.60%로 증가한다.

VI. Conclusion

본 논문에서는 배열의 정렬 상태를 이미지로 나타낸 데이터를 입력 데이터로, 각 알고리즘의 절대적 및 상대적 관점에서의 수행 시간을 표준화한 값을 출력 데이터로 하는 CNN을 이용한 딥러닝 방법론을 이용하여, 배열의 정렬 상태에 따라 어떤 정렬 알고리즘을 적용하는 것이 최적인지를 절대적 관점에서 추론하는 모델을 제안하였다. 또한, 이 모델을 통해 각 정렬 알고리즘이 어떤 정렬 형태를 갖는 배열에 최적인지를 상대적 관점에서 추정하고 실제 값과의 비교를 통해 그 정확도를 측정하였다. 이러한 실험의 결과를 통해 어떤 데이터에 정렬 알고리즘을 적용하려고 할 때 먼저 그 데이터의 정렬 상태를 통해 최적의 정렬 알고리즘이 무엇인지 추론하고, 그 알고리즘을 적용하면 정렬에 수행되는 시간을 절약할 수 있을 것이다. 또한 배열의 형태에 따라 어떤 알고리즘이 적합한지를 추정하는 모델을 적용하여 배열이 주어졌을 때 특정한 정렬 알고리즘을 적

용하는 것에 대한 적합성을 보다 쉽게 파악할 수 있을 것이다.

여기서는 배열 내부의 index에 따라 상수함수, 일차함수, 이차함수 형태로 나타난 값에 랜덤한 오차가 적용된 정렬 형태를 갖는 배열들을 학습 데이터로 사용했지만, 본 논문에서 적용한 방법론을 보다 다양한 정렬 형태를 갖는 배열들을 포함한 학습 및 테스트 데이터에 및 보다 다양한 정렬 알고리즘(Shell Sort 등)에 대해 적용한다면 정렬 형태에 따라 어떤 알고리즘을 적용하는 것이 정렬에 필요한 총 연산 횟수가 가장 적게 나타나는지, 그리고 각 알고리즘의 상대적인 적합성은 어느 정도인지를 일반화하여 추정할 수 있을 것이다. 즉 새로운 정렬 형태를 갖는 배열에 대해서도 절대적인 연산량 및 각 알고리즘별 상대적인 적합성을 보다 정확히 추론할 수 있을 것이다.

References

[1] Curtis R. Cook and Do Jin Kim, Best Sorting Algorithm for Nearly Sorted Lists, Comm. ACM, November 1980, available online at https://dl.acm.org/doi/pdf/10.1145/359024.359026

[2] Aditya Dev Mishra and Deepak Garg, Selection of Best Sorting Algorithm, International Journal of Intelligent Information Processing, 2008, available online at https://www.academia.edu/download/28569137/Selection_of_best_sorting_algorithm.pdf

[3] D Rao and B Ramesh, Experimental Based Selection of Best Sorting Algorithm, International Journal of Modern Engineering Research, 2012, available online at http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.417.217&rep=rep1&type=pdf

[4] Saad ALBA WI, and Saad AL-ZA WI, Understanding of a Convolutional Neural Network, ICET 2017, available online at https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8308186

[5] Diederik P. Kingma, Jimmy Lei Ba, ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION, ICLR 2015, available online at https://arxiv.org/pdf/1412.6980.pdf

[6] ztgu, sorting_algorithms_py, Github, available online at https://github.com/ztgu/sorting_algorithms_py

[7] IEEE, The NumPy Array: A Structure for Efficient Numerical Computation, Scientific Python, 2011, available online at https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5725236

[8] Mart´ın Abadi, Paul Barham, Jianmin Chen et al, TensorFlow: A system for large-scale machine learning, Google Brain, 2016, available online at https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[9] John D. Hunter, Matplotlib: A 2D Graphics Environment, IEEE, 2007, available online at https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4160265