Pusan National University

Computer Science and Engineering

Technical Report 2022-09

딥러닝 기반 워크로드 분석을 통한 SSD 성능 개선

2018246421 Ganchuluun Narantsatsralt

201824624 Ariunbold Odgerel

201824641 최성찬

지도교수 안성용

목 차

1. 서론

1.1. 연구 배경

SSD(Solid State Drive)는 비휘발성 저장 장치로 기존의 HDD(Hard Disk Drive) 보다 빠른 입출력, 저전력, 저소음, 경량성을 장점으로 갖는다. 그러나 메모리가 NAND 플래시 메모리로 구성되어 있어 데이터에 대한 제자리 쓰기 즉 덮어쓰기가 불가하다. 새로운 쓰기를 진행하기 위해서는 지우기 과정이 선행해야 한다. 이때 쓰기가 페이지 단위로 이루어지지만 지우기가 블록 단위로 이루어짐에 따라 문제가 발생한다. 하나의 페이지는 NAND 메모리 생산자에 따라 4KB 혹은 16KB이고 블록은 128 또는 256개의 페이지로 구성된다. 그러므로 하나의 블록을 지운다는 것은 256개 페이지를 삭제한다는 것이다. 이때 이미 쓰기가 수행된 동일한 주소에 새로운 쓰기가 수행된다면, 기존 블록에 쓰여있는 페이지는 invalid 페이지가 된다. 그리고 쓰기는 블록 내 비어있는 페이지에 진행된다. 이때 비어있는 페이지를 갖는 블록의 수가 임계치 이하로 적어졌을 때, GC(Garbage Collection)가 수행되어 페이지가 모두 쓰여진 블록을 지운다. 지우기 작업 중 블록 내에 남아있는 모든 valid 데이터들을 비어있는 블록에 쓰기를 수행하고 해당 블록의 모든 페이지를 지운다. 이로 인해 추가적인 쓰기가 발생되고 이는 추가적인 수행시간을 요한다. 나아가 플래시 메모리는 각 셀마다 쓰기 횟수가 정해져 있어, 해당 횟수가 찬다면 해당 셀은 수명을 다하여 더 이상 값을 저장할 수 없게 된다. 이는 wear leveling이라 불리우며, 이를 통해 추가적인 쓰기는 SSD의 성능과 수명에 악영향을 줌을 알 수 있다.

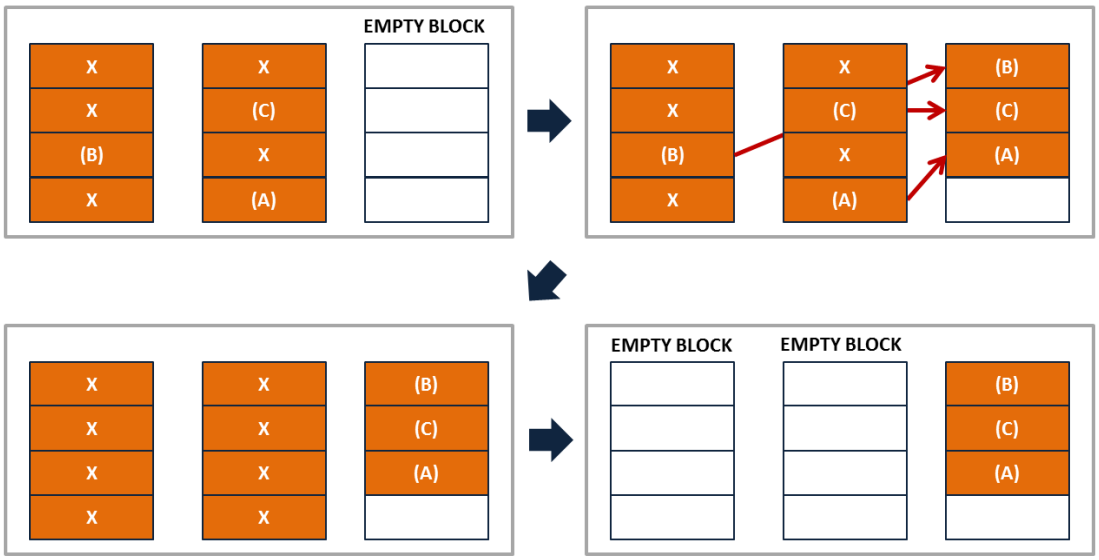


그림 1 블록 지우기 과정

1.2. 연구 목표

OS가 요청한 쓰기가 WR(Write Requested)라 할 때, GC 작업으로 인해 추가적인 쓰기가 발생한다. 고로 WR에 추가적인 쓰기를 더한 쓰기 횟수가 NAND 메모리에 쓰여진 총 쓰기이다. 이를 NAND Write라 하며, 해당 NAND Write을 WR로 나눈 쓰기 요청과 총 쓰여진 양의 비율을 WA(Write Amplification)이라 한다. 해당 연구의 목표는 추가적인 쓰기의 발생 수를 줄여 WA를 낮추는 것을 목표로 한다. 이를 개선하기 위해서는 블록 지우기시 valid 데이터의 수가 적어져야 한다. 이를 위해 머신러닝을 통한 이진분류를 수행하여 쓰기 요청이 된 주소가 자주 업데이트 될 것인지를 예측할 필요가 있다. 자주 업데이트되는 주소를 Hot, 업데이트 빈도가 적은 주소를 Cold로 분류하여 Hot과 Cold 주소가 각각 다른 블록에 위치한다면, Hot 주소를 갖는 블록은 빠르게 invalid 페이지로 채워질 것이며, Cold 주소를 갖는 블록은 느리게 invalid 페이지로 채워질 것이다. 이는 결과적으로 블록 지우기 수행 시 발생하는 valid 데이터 이동이 적어지게 한다.

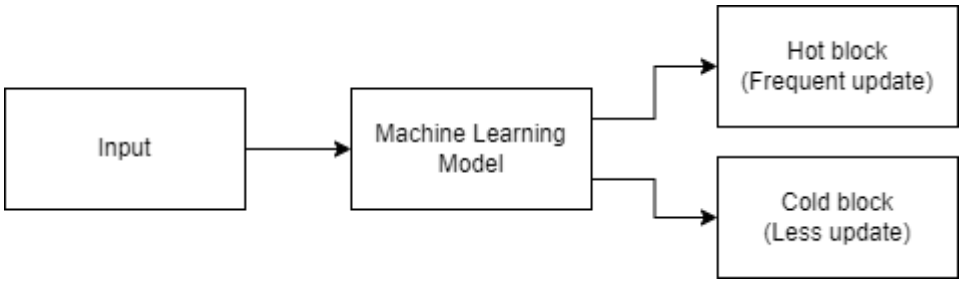


그림 2 연구 목표도

2. 연구 배경

2.1. 기존 Hot Cold 분류

SSD GC로 인해 발생하는 부하를 줄이기 위한 Hot/Cod 분류에 관한 연구는 꾸준히 진행되어왔다. 이에 다양한 접근방식이 제시되어왔다. 주로 Hot Cold를 어떻게 분류할지가 나뉜다. 먼저 한 연구[1]는 워킹 로그 블록과 Cold 로그 블록을 이용하여, Cold 블록들을 분류해내는 데에 집중했다. 작동 방식은 다음과 같다. 블록 지우기가 수행될 때, 블록 내에 존재하는 valid 페이지를 워킹 로그 블록에 저장한다. 이로써 워킹 로그 블록에는 지우기 과정을 한 번 살아남은 페이지들만이 존재하게 된다. 이후 쓰기 과정 중 해당 워킹 로그 블록 역시 가

득 차게 되면, 다시 그 안에 남아있는 valid 페이지들을 Cold 로그 블록으로 이동시킨다. 즉 로그 블록을 일종의 버퍼로 두어, 한 번 더 솎아진 데이터들만을 Cold 블록으로 구분하는 방식이다. 해당 방식은 Hot Cold 구분 없이 Greedy 알고리즘만을 사용한 결과보다 쓰기 처리율이 최대 18% 향상했다. Greedy 알고리즘은 블록들 중, invalid 페이지가 가장 많은 블록들에 대해 먼저 지우기를 수행하는 알고리즘이다.

기계학습을 이용한 연구 결과[2] 역시 존재한다. 해당 연구는 I/O 트레이스를 비지도 학습인 K-Means 알고리즘을 통해 분류했다. 매개변수로는 각 LBA(Logical Block Address)에 대한 접근 빈도와 접근 시간 간격을 사용했다. 그러나 K-Means 알고리즘은 순차적으로 주어지는 I/O 요청에 대해 매회 분류를 수행할 시 수행시간이 비대해진다는 단점을 가진다. 이에 I/O 쓰기 요청을 버퍼에 모아두고, 미리 정해둔 임계치에 도달하면 버퍼에 있는 데이터에 대해 분류를 수행한다는 해결 방법을 제안했다. 이를 위해 해당 버퍼와 분류를 수행하는 하드웨어 설계를 제시했다. 또한 분류에 있어서도, 기존의 Hot Cold에 추가로 Warm을 도입했다. 이러한 세분화된 구분을 통해 Cold 데이터에 더욱 업데이트가 적은 데이터만을 담고자 했다. 이러한 K-Means 알고리즘을 이용한 방식은 단순성, 유연성 등에 강점을 갖는다. 해당 연구는 위와 같은 SSD 하드웨어 유닛을 도입하여 기존 SSD 하드웨어보다 26.3%에서 57.7%의 성능 개선을 얻었다.

위 연구의 K-means에 사용된 매개변수에 시간 간격의 표준편차를 추가한 연구[3]가 존재한다. 표준편차가 의미하는 바는 작은 표준편차는 접근 간격이 일정함을 의미하며, 큰 표준편차는 접근 간격이 일정하지 않음을 의미한다. 이를 통해 접근 빈도와 접근 시간 간격이 비슷한 군집에 대하여, 접근 표준편차가 더 큰 군집을 더 차갑다고 판단했다. 또한 해당 연구에선 해당 변수들에 데이터 표준화를 수행하여 특별히 큰 값을 갖는 요소가 분류에 주는 영향을 감소시켰다. 나아가 접근 빈도에 1, 접근 시간 간격 평균에 0.8, 접근 시간 간격의 표준편차에 0.7이라는 가중치를 부여했다. 해당 연구는 성능 비교를 수행하지 않았다. 또한 해당 방식은 데이터에 대한 추가적인 전처리 과정을 동반하므로 앞서 언급한 실시간 K-means 군집화와 달리 실시간에 사용하기에 부하가 크다. 이에 해당 연구의 Hot Cold 구분 방식을 토대로 이번 연구에서 레이블링을 수행하여 이번 연구를 통해 작성한 모델 학습에 사용하였다.

2.2. 기계학습 모델

2.2.1. RNN(Recurrent Neural Network)

RNN은 이전 계층의 출력을 숨겨진 상태로 입력으로 사용할 수 있는 신경망이다. 예를 들어 영화 장면마다 어떤 상황이 벌어지는지 분류해야 한다. 이때 현재 어떤 상황이 일어나는지를 이해하려면 이전에 어떤 상황이 일어났는지를 이해해야 한다. 이렇듯 이전 데이터의 정보가 현재 정보를 처리하는 데에 필요한 상황에 RNN이 사용된다. 아래 표 1은 RNN 신경망 모델이 갖는 장단점을 보여준다.

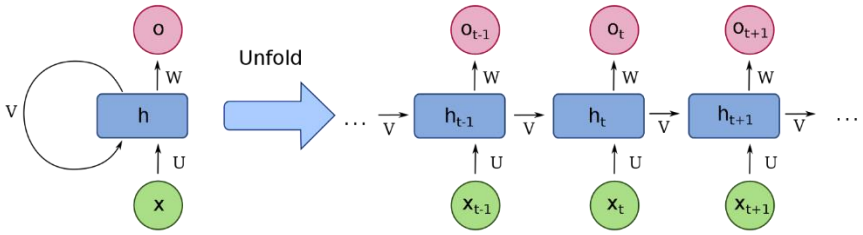


그림 3 RNN 내부구조

표 1 RNN 장단점

장점	단점
• 모든 입력 변수 길이가 처리 가능하다 • 출력 크기에 따른 모델 크기 상승이 없다. • 시계열 정보 처리가 가능하다. • 가중치가 보존된다.	• 더 긴 처리시간이 필요하다 • 시간적 텀이 긴 시계열 정보는 소실된다. • 미래 정보는 처리에 포함되지 않는다.

RNN이 가변길이 입력들을 처리할 수 있는 장점은 시간적 텀이 긴 시계열 정보를 처리하려 하면 신경망 결과가 악화되는 단점과 공존한다. 이러한 긴 시간 간격을 갖는 정보의 손실은 기울기 소실과 기울기 폭발에 의한 문제이다. 이는 장기적인 의존성을 처리하는 데에는 기울기가 계층을 거치며 큰 값 혹은 작은 값들이 연속적으로 곱해져 기울기가 지수적으로 늘거나 줄어드는 문제가 발생하는 까닭이다.

2.2.2. LSTM(Long Short Term Memory)

LSTM은 RNN의 일종으로 시계열 데이터에 대한 장기기억이 가능하다. RNN에서 발생하는 장기적 의존성 처리 문제를 해결하기 위해 제시되었다. 이러한 LSTM은 셀, 입력 게이트, 출력 게이트, 망각 게이트로 구성된다. 이러한 세가지 게이트와 셀은 정보를 유지할지 망각할지와 같은 정보의 흐름을 통제한다. 여기서 핵심적인 것은 셀 상태인데, 이는 아래 그림4에서 상단에 위치한 수평 라인이다. 이러한 LSTM은 정보가 셀 라인을 따라 흐르면서 그 하단에 위치한 게이트들을 이용하여 Hidden state 즉 이전 정보가 셀에 영향을 줄지 주지 않을지를 결정한다. 이러한 게이트들은 sigmoid 신경망 레이어로 이루며 점별곱을 수행한다.

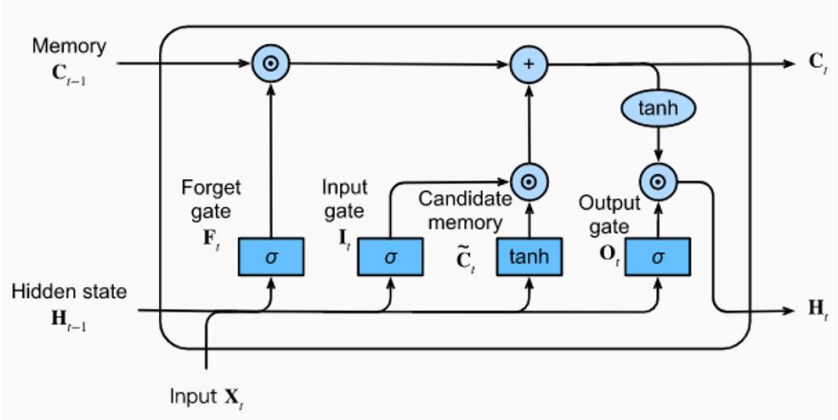


그림 4 LSTM 유닛 구조

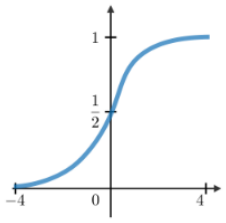
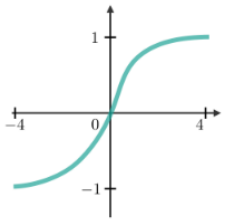
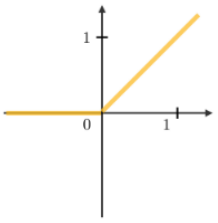
Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

그림 5 RNN과 ISTM에 사용되는 활성화함수

이러한 sigmoid 레이어 출력은 1과 0 사잇값이며 해당 출력값이 셀 라인에 이전 정보가 영향을 줄지 주지 않을지를 결정한다. 값이 0일 때 이전 정보는 게이트를 통과하지 않으며 값이 1일 때 정보가 게이트를 통과하여 셀 상태에 영향을 준다. 이러한 게이트와 셀을 통해

LSTM은 기울기 소실 및 폭발을 방지한다. 위 모델들을 해당 연구에 사용한다.

3. 연구 내용

3.1. 레이블링 데이터 작성

3.1.1. 데이터 선정 및 전처리

3.1.1.1. YCSB RocksDB SSD

초기에 모델 학습을 위해 SSD 기반 워크로드 분석[4]에 사용된 YCSB RocksDB SSD 데이터를 선정했다. 이는 UNIX blkparse를 통해 작성되었으며 다음과 같은 형식을 갖는다.

259,2 13 2861663 1227.460991324 4073 C WS 499573680 + 8 [0]
[Device Major Number,Device Minor Number] [CPU Core ID]
[Record ID] [Timestamp (in nanoseconds)] [ProcessID] [Trace Action]
[OperationType] [SectorNumber + I/O Size] [ProcessName]

해당 과제는 WA를 줄이는 것을 목표로 하므로, 쓰기 이외의 작업은 관련이 없다. 고로 위 형식에서 [Operation Type]이 W 즉 Write가 포함된 데이터만을 분류했다. 다시 여기서 [Trace Action] 열에서 D(Issue Device) 즉 I/O 요청이 기기에 보내지는 단계를 다시 분류했으며, 분류 후 학습 및 시뮬레이션에 사용되지 않으므로 데이터에 해당 열을 포함하지 않았다. 항목들 중 접근 시간 간격을 확인할 수 있는 Timestamp, 논리 주소를 표기하는 LBA(Logical Block Address), 블록 쓰기 크기를 표기하는 I/O Size를 포함하였고, 그 처리 결과는 아래와 같다.

1.652816521 7487488 2048
[Timestamp] [SectorNumber] [I/O Size]

위 결과는 첫 열부터 각각 시작으로부터 경과시간, LBA, LBA에 쓰여진 블록 수를 표기한다. Blckparse 사용 시 1개 블록은 512Byte를 의미하므로 위와 같이 2048 블록은 1KB를 의미한다. 위 형식으로 가공한 데이터를 모델의 학습 및 시뮬레이팅에 사용했다. 또한 레이블

링을 위해 추가적인 처리를 수행했다. Hot/Cold 구분을 위한 특징을 추출할 수 있도록 각 LBA에 대해 접근 빈도, 접근시간 간격의 평균, 접근 시간 간격의 표준편차 그리고 총 요청된 쓰기 블록 수를 계산하여 파일로 만들었고 그 형식은 다음과 같다.

753921 90736 0.4592 23941.33982 544416
[SectorNumber] [Frequency] [Time Interval Avg] [Time Interval StdDeriv] [I/O Size]

위 데이터상에서 LBA 753921번지는 해당 워크로드에서 총 90736번 접근되었으며 0.4592초의 접근 간격 평균을 갖는다. 또한 접근간격 표준편차가 23941.33982인데 이 값이 커질수록 접근간격이 일정하지 않음을 의미한다. 마지막 항목은 총 544,416개 블록 쓰기가 요청되었음을 의미한다. 이러한 데이터 처리과정에 이후 더 나은 군집화를 수행할 수 있도록 접근 빈도가 1이어서 접근 간격 평균이 0인 LBA에 대하여 전체 LBA중 가장 큰 접근 간격 평균보다 0.1더 큰 값을 할당했다. 접근 간격 표준편차의 경우 접근 빈도가 2회 이상이어야 계산이 가능하므로, 그 이하인 경우 0값을 할당하였다.

해당 데이터를 기반으로 초기에 모델 학습을 진행했다. 그러나 시뮬레이터 작성 완료 후 Hot Cold 구분 없이 Greedy GC를 수행했을 때 WA가 1.02 정도로 이미 WA 정도가 낮아 개선을 기대하기 힘든 데이터라는 것을 확인했다. 이에 해당 워크로드 데이터 사용을 중지하고 새로운 데이터를 찾았다.

3.1.1.2. OLTP Application I/O

새로운 데이터는 UMass Trace Repositiry[5]에서 제공되는 OLTP(Online Transaction Processing) Application I/O 워크로드를 이용했다. 그 형식은 위와 달리 다음과 같다.

0,20941264,8192,W,0.551706,Alpha/NT
[Application specific unit] [LBA] [Size] [Opcode] [Timestamp] [Optional]

위 항목들 중 앞서 설명한 데이터 형식과 호환이 되도록, [Opcode]가 쓰기를 의미하는 W

인것들에서 [Timestamp] [LBA] [Size]만을 추출해서 사용했다. 해당 데이터는 시뮬레이션 수행 시 GC 트리거나 GC 수행 임계치를 조절함에 따라 WA가 7에서 2사이에 달하여 WA 개선을 테스트하기에 적절하다고 판단했다. 이 데이터 또한 위 데이터와 마찬가지로 군집화 수행을 위해 동일한 과정의 추가 처리를 진행하였다.

기계학습 모델을 학습시키기 위하여는 레이블링 데이터가 필요하다. 초기에 모델 설계에 대한 실험들을 진행해야하는 상황에, K-means 군집화 역시 작업중에 있었으므로 원활한 실험 진행을 위해 임의적인 법칙을 지정하여 임시 레이블링을 진행했다. 해당 법칙은 단순하다. 계산된 접근 빈도와 접근 시간 평균을 기반으로 하여 접근빈도가 10 이상인 모든 LBA를 hot으로 지정했다. 또한 접근빈도가 10 이하여도 접근빈도가 가장 높은 LBA의 접근 시간 간격 평균치와 같거나 작은 시간 간격 평균치를 갖는 LBA들 역시 Hot으로 지정했다. 이러한 분류를 통해 Hot으로 분류된 LBA는 전체 LBA들의 2.7%이다. 해당 레이블링은 이후 시뮬레이션 작성 후 테스트해 보았을 때, 앞서 말한 SSD 분석에 사용된 데이터에선 11%의 개선을 보였으나 OLTP 데이터에선 5% 저하를 보였다. 이는 해당 방식이 WA 개선을 보장하지 못함을 보인다.

3.1.2. K-Means 분류

앞선 2.1절에서 설명한 바와 같이 학습에 사용할 레이블링 데이터를 군집화 알고리즘을 이용하여 수행하였다. 초기에는 군집 수를 지정해주지 않아도 되는 점, 이상치에 대항 대응이 강화된 점을 들어 DBSCAN(Density-Based Spatial Clustering of Applications with Noise) 사용을 시도했다. 그러나 sklearn을 통해 사용가능한 DBSCAN 모델은 2차 공간 복잡도를 갖아, 정보량이 많은 워크로드 데이터 특성으로 인해 12GB 이상의 램 사용을 요하여 이보다 작은 공간 복잡도를 갖는 k-means를 수행하도록 하였다.

군집화에 사용된 변수는 초기에 주소 접근 빈도수, 접근 시간 간격 평균, 쓰기 블록 수를 사용했다. 또한 군집화가 특이값에 받는 영향을 줄이기 위해 표준화를 진행하였는데 이는 sklearn의 MinMaxScaler를 사용했다. 해당 함수는 데이터를 0과 1 사이의 수로 표준화하는데 이때 본래 데이터의 분포를 유지한다. 마지막으로 K 값을 설정하는 데에는 yellowbrick의 KElbowVisualizer 메소드를 사용하여 엘보우 기법을 이용했고, 결과로 K에 2를 할당했다. 그 결과 훈련 데이터의 LBA들 중 84%가 Hot 데이터로 분류되었고, 이는 부적절하다고 판단했

다. 이러한 부적절한 결과는 다음 요소들에서 기인한다. 하나는 주소 쓰기 블록 수는 데이터가 Hot인지 Cold인지 큰 영향을 주지 않는다는 점이다. 예를 들어 블록 수가 40개인 데이터가 한번도 업데이트 되지 않는다면 이는 콜드 데이터이다. 이는 해당 데이터가 invalid 페이지를 생성하지 않는 까닭이다. 블록 수가 1인 데이터라도 업데이트가 자주 된다면 많은 invalid 페이지를 생성한다. 고로 쓰기 블록 수는 hot cold 구분에 적절하지 않다. 또한 아래 그림 6에서 볼 수 있듯이 빈도수를 갖는 LBA가 전체 데이터 10% 이내에 몰려있다. 이러한 상황에서 본래 분포를 유지하는 MinMaxSclaer 표준화 방식은 이러한 편중된 데이터에 효과적이지 못하여, 결과적으로 k가 2가 되었고 효과적이지 못한 군집화 결과로 이어진 것이다.

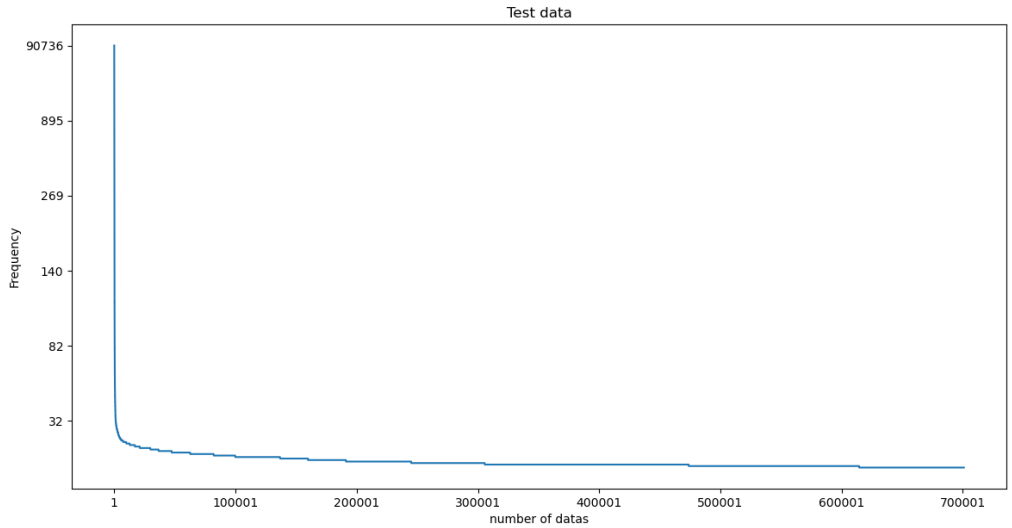


그림 6 훈련 데이터 빈도수

이에 2.1절에 설명한 연구를 토대로 변수에서 쓰기 블록 수를 탈락시키고, 주소 접근 간격의 표준편차를 추가하였다. 또한 표준화 함수를 sklearn의 StandardScaler로 교체했다. 해당 함수는 표준화 수행 시 본래 데이터의 분포를 유지하지 않는다. 이 결과 다시 엘보우 기법을 적용했을 때, k값이 4로 결정되었고 그 결과는 아래 그림과 같다.

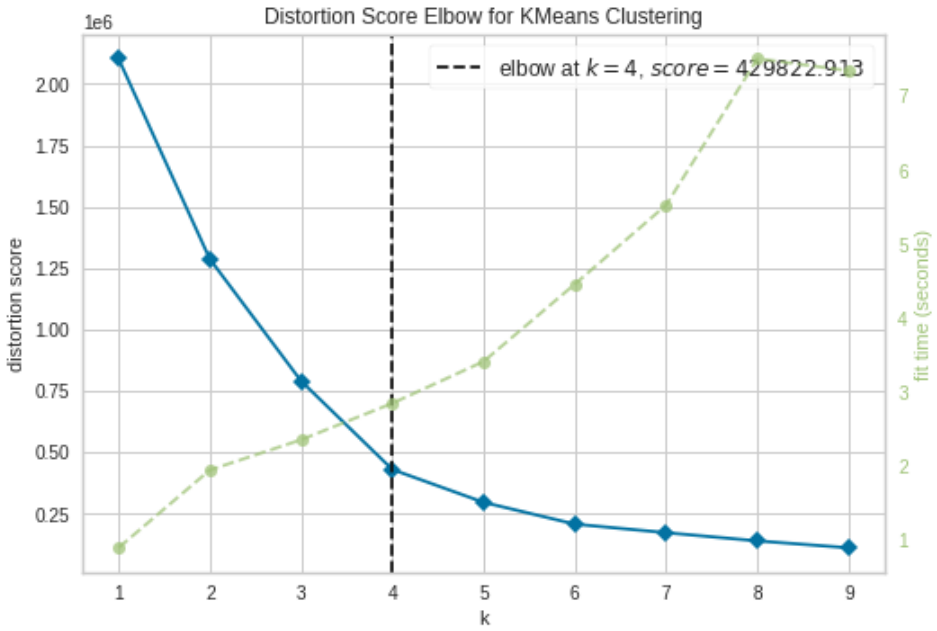


그림 7 OLTP 데이터 엘보우 기법 적용

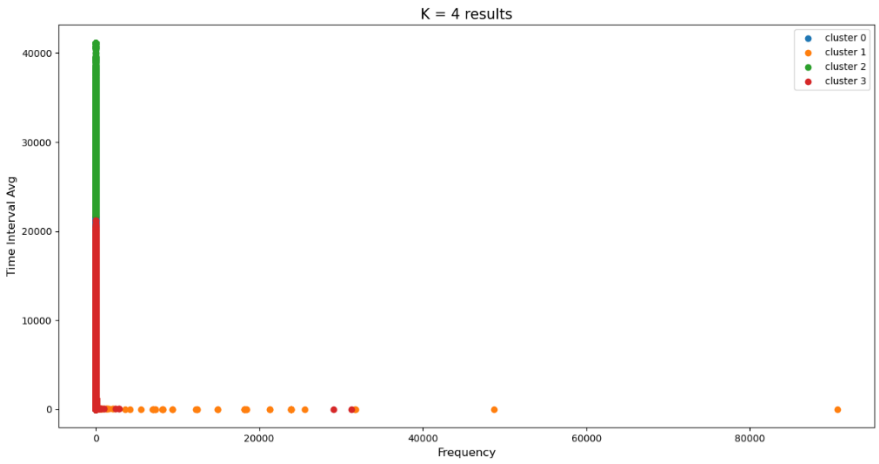


그림 8 OTLP 군집화 결과

그림 8을 통해 높은 빈도수를 갖는 데이터는 소수이며, 높은 시간 간격 평균을 갖는 데이터들이 모두 낮은 빈도수를 갖고 있음을 확인할 수 있다. 이러한 데이터들에 대한 Hot Cold 분류 평가는 아래 표 2를 통해 진행했다.

표 2 OLTP 데이터 군집화 결과

군집	접근 빈도 평균	평균 접근 간격	접근 간격 표준편차	LBA 수
0	2.26	4,354.45	955.93	149,575
1	11.15	1,990.28	26,167.93	184,942
2	1.15	39,117.26	0	101,204
3	5.97	5,373.69	16,545.31	265,152

표 2의 내용을 통해 접근 빈도가 평균이 가장 높은 군집은 1번임을 알 수 있다. 해당 군집은 가장 낮은 평균 접근간격을 가진다. 그 와중에 표준편차가 높다는 것은 이러한 데이터들의 접근 간격이 고르게 분포하지 않다는 것을 의미한다. 이러한 1번 데이터를 Hot으로 분류했을 때 전체 LBA의 26%가 Hot으로 분류됨을 알 수 있으며 이는 적절하다고 판단했다. 또한 군집 2는 가장 큰 평균 접근 간격과 가정 적은 접근 빈도 평균을 기록하여 Cold임을 확실히 보여준다. 나머지 군집 0과 3의 경우가 특이한데, 군집 3의 경우 0번 보다 높은 접근 빈도 평균을 보이나, 평균 접근 간격과 접근 간격 표준편차 모두 군집 0보다 높은 모습을 보인다. 이는 군집 0이 군집 3보다 더 짧은 접근 간격을 가지면서도 각 접근 시간이 편중되지 않음을 보인다. 그러나 WA의 증가는 invalid 페이지가 얼마나 발생하는지에 가장 큰 영향을 받으므로, 접근 빈도 평균을 가장 우선하여 평가할 필요가 있다. 그런 이유에서 3번 군집이 0번 군집보다 더 뜨겁다고 판단했다. 이러한 군집은 이후 Hot Warm Cold, 나아가 Hot Warm Cool Cold 데이터로 나누는데 사용되었다. 또한 이후 OLTP 데이터를 6:2:2 비율로 각각 훈련, 검증, 테스트 데이터로 나누었다. 이때 레이블링은 여전히 나누기 이전의 군집을 이용했다. 그러나 각각 데이터의 군집을 비교하기 위해 각 데이터에 대한 군집화를 모두 수행했으며 결과는 훈련 데이터는 군집이 나누기 이전과 거의 일치하였으나, 검증 데이터와 테스트 데이터는 이전과 다른 군집을 보였다. 특히 Hot에 속하는 LBA 수가 현저히 적었다.

3.2. 기계학습 모델 설계 및 학습

3.2.1. 데이터 전처리

3.2.1.1. 데이터 정규화

데이터 정규화 사용의 의의는 기계학습 모델에 입력되는 특징값들에 특이값 즉 너무 큰 값이나 너무 작은 값이 주는 영향을 줄이기 위해 범위를 축소시키는 데에 있다. 이는 모델의 훈련 처리 속도와 안정성을 향상시킨다. 해당 연구에 사용된 정규화 기법은 표준 점수이다.

표준 점수란 통계학적으로 정규분포를 만들고 개개의 경우가 표준편차상에 어떤 위치를 차지하는지를 보여주는 차원 없는 수치이다.

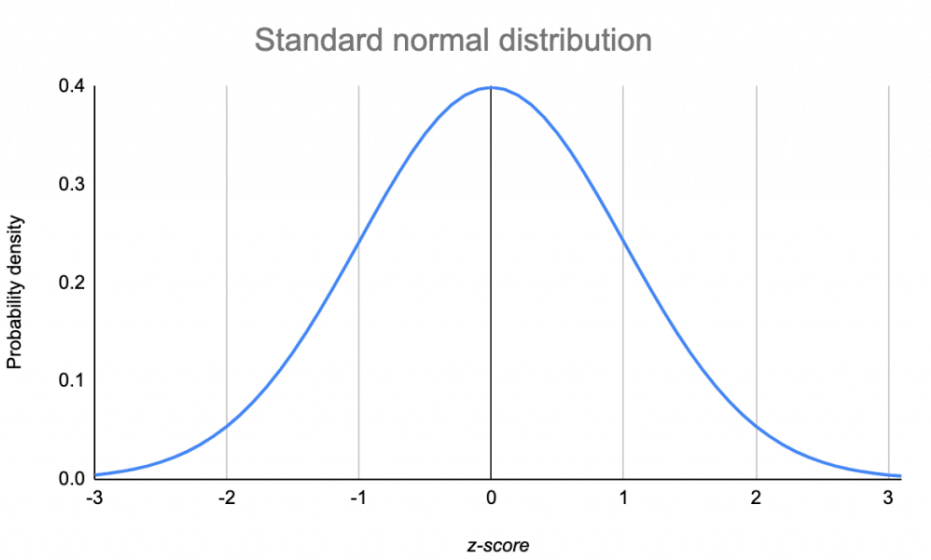


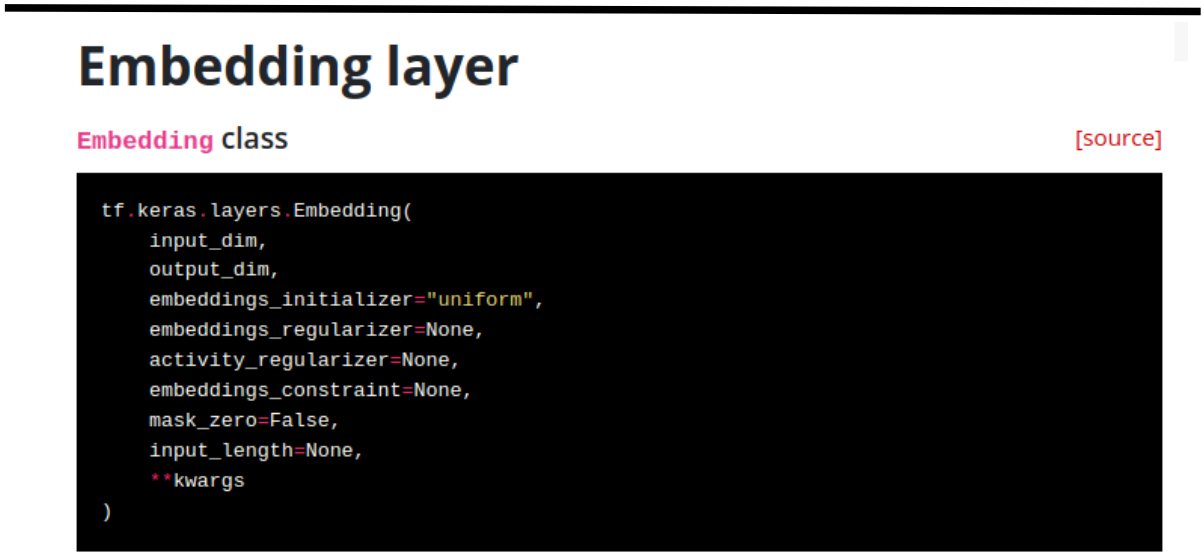
그림 9 Z-score의 범위 (표준 점수)

3.2.1.2. 데이터 임베딩

LSTM 모델 개발에 있어 가장 중요한 요소 중 하나는 토큰을 어떻게 표현하느냐이다. 이를 위한 두가지 방법이 있는데 하나는 One-hot 인코딩이고 다른 하나는 기수적 임베딩이다. One-hot 인코딩이 작동하는 방식은 다음과 같다. 10개 번호를 표기하기 위하여 다음과 같은 10개 원소를 갖는 배열이 필요하다.

1 = [0,1,0,0,0,0,0,0,0,0]
5 = [0,0,0,0,0,1,0,0,0,0]
9 = [0,0,0,0,0,0,0,0,0,1]

그러나 이번 연구에 사용되는 워크로드 데이터는 약 백오십만개의 다른 주소를 갖으며 이를 임베딩 할 필요가 있다. 이때 One-hot 인코딩을 사용한다면 백오십만 길이를 갖는 배열이 백오십만개 필요하다. 이에 해당 방식보다 기수적 임베딩을 사용했다.



Embedding layer

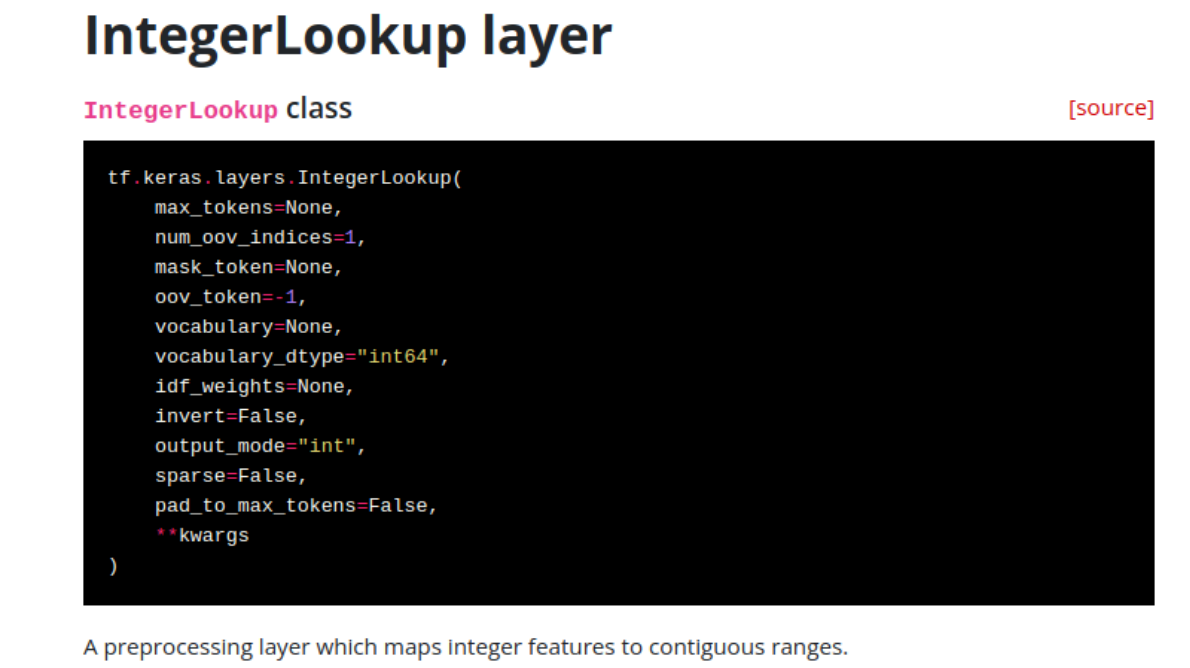
Embedding class [source]

```
tf.keras.layers.Embedding(
    input_dim,
    output_dim,
    embeddings_initializer="uniform",
    embeddings_regularizer=None,
    activity_regularizer=None,
    embeddings_constraint=None,
    mask_zero=False,
    input_length=None,
    **kwargs
)
```

Turns positive integers (indexes) into dense vectors of fixed size.

그림 10 기수적 임베딩을 위한 임베딩 계층

기수적 임베딩은 입력 주소를 실수 값을 갖는 벡터로 변환한다. 그러나 텐서플로우 api의 제한사항 때문에 임의의 스칼라 값을 사용할 수 없는 문제가 있다. 이에 이를 우회하기 위하여 스칼라 값을 선형적 순서로 정수와 맵핑하였다.



IntegerLookup layer

IntegerLookup class [source]

```
tf.keras.layers.IntegerLookup(
    max_tokens=None,
    num_oov_indices=1,
    mask_token=None,
    oov_token=-1,
    vocabulary=None,
    vocabulary_dtype="int64",
    idf_weights=None,
    invert=False,
    output_mode="int",
    sparse=False,
    pad_to_max_tokens=False,
    **kwargs
)
```

A preprocessing layer which maps integer features to contiguous ranges.

그림 11 IntegerLookup 계층

3.2.1.3. Windowing

LSTM 적용 모델에 있어 Window 방식은 모델이 몇 개의 과거 정보를 사용할지 정해주는 데에 사용된다. 예를 들어 이번 연구에 사용된 모델은 128개의 과거 정보를 사용하도록 하였고, 이에 따라 아래 그림과 같은 윈도우를 설계했다.

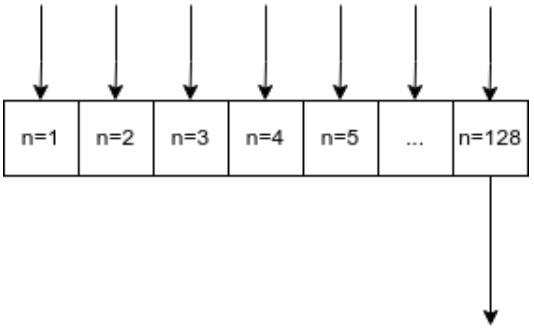


그림 12 128 넓이 윈도우

3.2.2. 모델 설계

이번 연구에 사용되는 모델은 아래와 같이 설계된다. 이때 input_1에 임베딩이 사용되어 LSTM 계층에 여러 변수를 줄 수 없다. 그로 인해 두번째 입력을 분리하여 LSTM 계층이 첫 번째 입력을 학습하도록 한 후에, 그 출력과 합산시켰다.

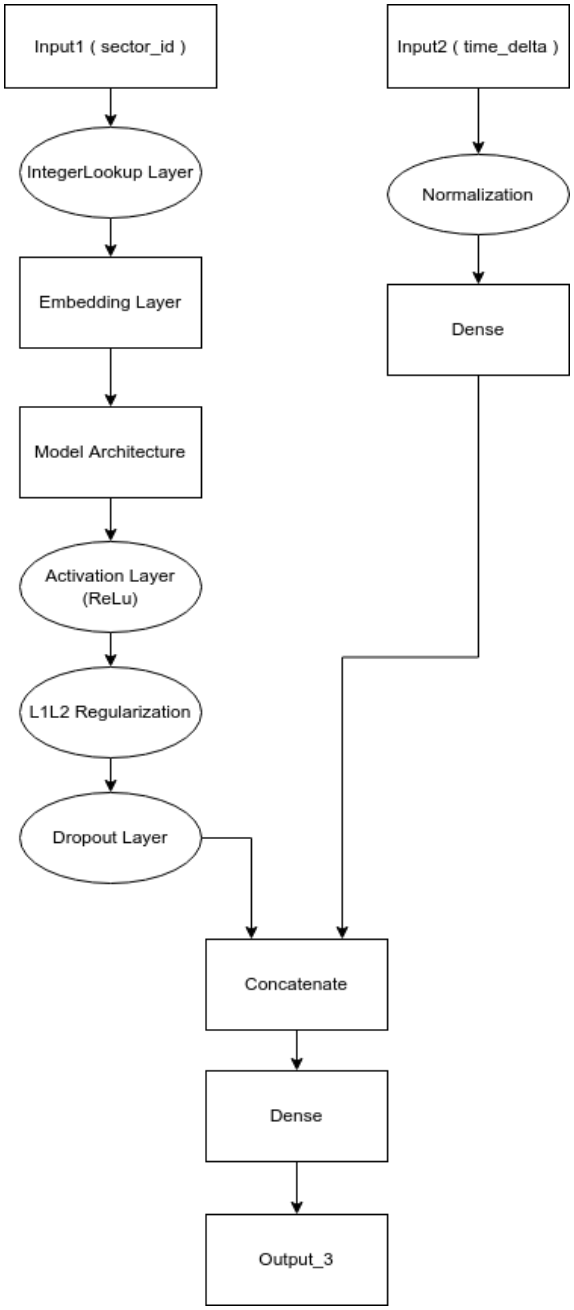


그림 13 모델 설계

위 그림과 같이 모델은 구조는 5개 부분으로 나뉘어져 있다. 이는 각각 입력계층, 데이터 전처리, LSTM 계층, 정규화 계층 그리고 출력 계층으로 되어있다. 여기서 입력 2(time_delta)는 워크로드 데이터의 timestamp로부터 계산된다. 이는 각 LBA의 접근 시간 간격이다. 만일 LBA가 단 한번만 등장한다면 이를 0으로 처리했다. 모델 학습에 더 큰 영향을 끼치는 부분은 입력 1(LBA 주소)이다. 이는 접근 빈도가 Hot/Cold 구분에 있어 가장 영향력이 큰 부분이기 때문이다. 하나의 LBA가 접근 빈도가 높다면 이를 Hot으로 접근 빈도가 낮다면 이를

Cold로 분류해야 한다. 이를 위하여는 이전 등장에 대한 정보를 기억하고 있어야 하므로 장기 기억에 강점을 갖는 LSTM 신경망을 선택한 것이다.

전처리 부분은 정수로 되어있는 LBA 주소 값을 모델이 이해할 수 있도록 변환하는 부분이다. 이에 대한 설명은 앞의 절에서 설명한 바와 같다. 이러한 LSTM 모델은 과적합에 취약한 모습을 보인다. 이번 연구에서 실험한 결과 모델은 단 3번째 에포크부터 과적합을 보이기 시작했다. 이를 극복하도록 정규화 계층을 사용했는데, 이는 각 계층이 손실함수에 주는 영향을 조정하고, 손실 함수 학습에 영향을 주는 L1L2 계수를 설정해주는 역할을 한다. 이러한 전처리 메소드는 최적화를 수행하는 옵티마이저에 따라 모델 학습의 안정성에 영향을 준다. 기울기 정도에 영향을 받는 옵티마이저의 사용은 실패의 가능성이 있다. 그러나 Adam의 경우 그러한 기울기 정도에 영향을 받지 않으므로 Adam을 모델의 옵티마이저로 사용했다.

마지막으로 출력을 살펴보면 다음과 같다. 모델의 출력은 각 Hot Warm Cold 세가지 중 이번 LBA가 어떤 온도를 갖을지 예측한다. 그 출력은 [0.2, 0.1, 0.7]의 형태이며, 각 값은 확률을 의미한다. 고로 해당 출력은 3번째 항목일 확률이 가장 높은 것이다.

3.2.2.1. 모델 구조

이번 연구에서 두개의 모델을 작성했다. 하나는 CNN(Convolutional Neural Network) LSTM 모델이며 다른 하나는 Stacked LSTM 모델이다. 그 구조는 아래 그림과 같다.

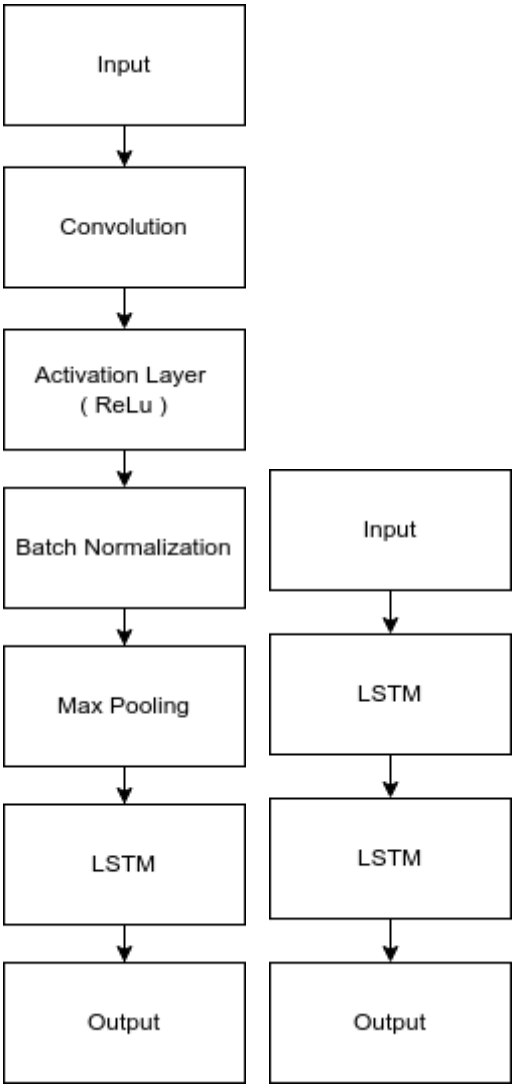


그림 14 좌 CNN LSTM 우 Stacked LSTM

먼저 CNN LSTM 구조는 입력 데이터의 특징을 추출해내는 CNN 계층을 시계열 데이터 처리를 도와줄 LSTM과 함께 수행한다. 이러한 방식은 LRCN(Long-term Recurrent Convolutional Network) 모델에서 사용되는데, 우리 구조에서는 이와는 조금 다르게 CNN을 LSTM 전반부에 위치시키기 때문에 CNN LSTM으로 설명하고자 한다.

일반적으로 CNN은 이차원 이미지 데이터에 사용된다. 그러나 CNN 모델은 일차원 시계열 데이터에 대한 특성 학습 및 추출에도 효과적이다. CNN 계층은 다음과 같이 작동한다. 먼저 Covolution 계층에서 크기 3의 필터를 통해 패턴을 학습한다. 이후 출력은 ReLu 활성함수를 통해 처리된다. 해당 함수는 음수 값들을 탈락시킨다. 이후 그 출력은 다시 배치 정규화 기법

을 통해 정규화된다. 이후 정규화된 출력은 다시 LSTM 계층에 입력값으로 사용되어 시계열 정보를 학습하게 된다.

Stacked LSTM 구조는 더욱 간단하다. 딥러닝 프로젝트들의 공통적인 특징은 중요 계층들을 여러 겹 쌓아 올린다는 것이다. 이로써 더욱 복잡하고 정확한 연산을 수행한다. 일반적인 LSTM 모델은 하나의 LSTM 은닉 계층을 사용하는데 반해 Stacked LSTM은 하나 이상의 은닉 LSTM 계층을 쌓아 연산을 수행한다. 이는 더욱 복잡한 특성에 대한 학습을 일으킨다.

3.2.2.2. LSTM 정규화

LSTM 모델은 과적합에 취약하므로 정규화가 필요하다. 이에 가중치에 정규화를 가함으로써 과적함을 줄이고 모델 성능을 개선을 도모했다. 사용한 정규화 기법은 다음과 같다. 먼저 L1L2 정규화를 사용했다. L2 정규화는 손실함수에 제곱된 계수를 더하는 방식이다. 이때 계수가 너무 크다면 가중치가 비대해져 오히려 과소적합을 일으킬 수 있다. L1 정규화는 손실에 계수 절대값을 더하는 데, 이는 손실에 페널티를 가하여 과적합을 방지시키는 의미를 갖는다. L1과 L2 정규화의 가장 큰 차이는 L1이 중요도가 낮은 특징값들을 제거하거나 축소시키는 방식으로 작동하고, L2는 과적합을 축소시키는 역할을 한다는 것이다.

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \quad \sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

그림 15 L1 비용함수 L2 비용함수

L1 정규화는 많은 특징값을 갖는 모델에 적합한데 반해 이번 연구에 사용되는 모델은 많은 특징값을 갖지 않으므로 L2 정규화를 사용했다. 이는 0.01의 하이퍼파라미터 값을 가지고 커널, 및 편중치 등을 정규화한다.

또한 사용중인 워크로드 데이터는 편중 되어있다. 약 92%의 데이터가 3개 클래스로 나눈다면 하나의 클래스에 속한다. 이럴 때 높은 출력 정확도를 갖기는 쉬우나, 이것이 단순히 한

가지 클래스를 선택하기만 하는 방식이라면 문제가 된다. 이를 방지하기 위해 우세한 클래스가 갖는 영향력을 감소시킬 필요가 있다. 이는 훈련 모델에 각 계층마다 다른 가중치를 부여함으로써 해결된다.

```
class_weight = {0: 60.,
                1: 20,
                2: 1}
```

그림 16 클래스 가중치 설정

학습과정 중 모델이 학습을 통한 향상을 멈추고 과적합하기 시작하는 경우가 있다. 이러한 경향은 많은 에포크를 돌리며 학습할 때 더 커지는데, 훈련을 중지시킬 수 없는 경우가 있다. 이를 방지하기 위해 텐서플로우에서 제공하는 빠른 종료 함수를 도입했다. 이는 2 인내 단계를 갖는데, 이는 연속적으로 2개 에포크에서 정확도 향상이 일어나지 않았을 때, 3번째 런에서 학습 과정을 정지시키는 방식이다.

3.3. 시뮬레이터

시뮬레이터 작성에 앞서 SSD 벤치마킹 및 FTL 테스트 등을 위해 존재하는 에뮬레이터를 사용하여 성능 평가를 진행하고자 하였다. 이를 위해 먼저 SimpleSSD 에뮬레이터를 테스트해보았으나, 해당 오픈소스 문서에 따른 단계를 모두 지키며 설치를 진행하였음에도 segmentation fault가 발생했다. 이러한 문제에 대한 해결책을 찾아보던 중 다른 실험 환경들에서도 같은 문제가 발생한다는 점을 발견하여 다른 에뮬레이터를 테스트하였다. 이에 FEMU를 실험하였으나 실행환경이 UNIX 혹은 LINUX여야 하는 점이 문제가 되었다. 이에 해당 환경을 구축하여 테스트하였으나 실행 후 smartctl 혹은 nvme가 작동하지 않는 문제가 발생하여 SSD WA 측정에 지속적으로 실패하였다. 해당 문제를 해결하기 위해 여러 솔루션을 적용해 보았으나, 결과적으로 해당 오픈소스 문제를 해결하는 것 보다 시뮬레이터 작성이 추후 머신러닝 모델과 병합하기에 더 적합하다는 판단 하에 시뮬레이터를 작성했다.

시뮬레이터 작성은 초기에 머신러닝 모델과의 연동 용이성을 위해 파이썬으로 작성하기 시작했다. 그러나 페이지 맵핑 테이블 작성 등에 포인터를 통한 링킹이 필요하여 다시 C++

를 통해 작성했다. 해당 시뮬레이터는 SSD 내부 FTL(Flash Translation Layer)의 작동을 모사한 것이다. FTL은 그 이름과 같이 OS가 HDD를 사용하는 환경에서 작성되었으므로, 해당 OS의 입출력 명령과 SSD 하드웨어 사이에서 번역의 역할을 한다. 이는 HDD가 저장단위를 섹터 기반으로 하며 SSD는 페이지와 블록을 기반으로 하는 차이로 인한 것이다. OS 입출력 및 파일구조도 이에 맞춰 구성되어 있어 SSD에 사용하기 위해 변환 과정을 거쳐야 한다. 또한 앞서 연구배경에서 설명한 바와 같이 wear leveling이 발생하여 SSD의 성능과 수명에 악영향을 주지 않도록, 모든 블록 및 페이지에 쓰기가 고루 이루어지게 하는 역할을 수행한다. 더하여 GC 작업을 FTL에서 수행한다. 해당 시뮬레이터에서는 WA 측정에 의의가 있으므로 FTL이 갖는 여러 작업들 중 이러한 OS I/O에 대한 페이지 맵핑과 GC 만을 다루며 wear leveling은 수행하지 않도록 하였다.

WA 향상 결과를 확인하기 위하여, 두가지 버전을 작성했다. 하나는 Hot/Cold 구분없이 Greedy GC 만을 수행하는 방식으로 해당 버전의 WA가 비교 기준점으로 사용된다. 다른 하나는 기계학습 모델을 사용하여 Hot Cold 예상 값을 토대로 이들을 각기 다른 블록에 쓰기를 수행한다. 이는 Cold LBA들이 같은 블록에 모이게 함으로써, 해당 블록내 invalid 페이지가 희소하게 발생하여 블록 지우기 대상이 되지 않도록 하여 결과적으로 WA를 낮추기 위함이다. 이러한 Hot Cold 분류에서 나아가 Hot Warm Cold, Hot Warm Cool Cold 역시 테스트할 수 있도록 코드를 작성했다. 이러한 테스트에 있어 SSD 블록 개수, 페이지 크기, GC 트리거 임계치, 블록 지우기 비율 등을 매개변수를 통해 수정할 수 있도록 하여 다양한 환경에 대한 실험을 진행할 수 있도록 하였다. 이러한 두 시뮬레이터는 기본적으로 앞선 3.1.1 절에서 설명한 훈련에 사용되는 워크로드를 한줄 씩 읽어들이며 시뮬레이션을 진행한다.

시뮬레이션 후 출력은 각기 Requested Write, Additional Write, Nand Write, WA로 이루어진다. Requested Write는 IO 명령에서 요청된 쓰기 개수이며, Additional Write는 GC로 인해 다른 블록에 쓰이게 되는 valid 페이지 개수를 의미한다. Nand Write는 이름과 마찬가지로 실제로 Nand 저장장치에 쓰여진 페이지 수를 의미하며 이는 Requested Write + Additional Write에 해당된다. WA는 Nand Write를 Requested Write로 나눈 것으로 요청된 쓰기와 실재로 수행된 쓰기의 차이를 보여준다. 즉 WA가 1이라는 것은 요청된 쓰기와 실재 수행된 쓰기의 크기가 같다는 것이며, WA가 더 커지는 것은 추가적인 쓰기가 더욱 빈번히 발생하여 추가적인 쓰기 부하가 발생하였음을 보여준다. 이러한 WA 및 Nand Write를 통한 성능개선 여부는 아래 4장에서 설명하도록 한다.

시뮬레이터 작성 후 훈련 데이터에 대한 실험은 모델 개발에 영향을 끼치게 되었다. 앞서 설명한 K-means를 통한 군집 데이터를 해당 연구에서 모델 학습의 레이블 데이터로 사용한다. 이는 모델이 100%의 정확도를 갖을 때 레이블 데이터로 활용하는 군집을 정확히 예측한다는 것을 의미하며 동시에 군집을 통한 개선도 이상을 기대할 수 없음을 의미한다. 이에 해당 군집을 활용하여 훈련 데이터에 대해 Hot/Cold 분류를 수행해 WA 감소율을 측정해보았다. 페이지 크기 4KB, 블록당 페이지 개수 128, 블록 개수 1800, GC 촉발 임계치 25%, GC 수행 임계치 85%로 했을 때의 결과이다. AW는 Additional Write, RW는 Requested Write, NW는 Nand Write를 의미한다. 또한 H는 Hot, W는 Warm, L은 Cool, C는 Cold를 표기한다.

표 3 OLTP 데이터 군집 시뮬레이션 결과

	Vanilla	HC	HWC	CLW	HWLC
RW	5,628,247	5,628,247	5,628,247	5,628,247	5,628,247
AW	30,562,423	27,603,307	25,336,272	26,646,022	24,806,075
NW	36,190,670	33,231,554	30,964,519	32,274,269	30,434,322
WA	6.43019	5.90442	5.50163	5.73434	5.40742

위 결과로 보았을 때 HWLC로 나눈 경우가 가장 낮은 WA를 보였다. CLW를 테스트 한 이유는 군집의 형성에 따라 CLW과 HWC 보다 높은 경우가 존재하기 때문이다. 이는 현재 사용중인 테스트 데이터의 경우에서도 확인할 수 있는 데, 이는 해당 데이터의 Hot 군집이 아주 작은 까닭에서 비롯한다. 또한 Cold와 Cool 군집의 차이가 클수록 CLW를 사용하는 것이 HWC를 사용하는 경우보다 나은 결과를 보였고, 그 차이가 작을수록 HWC가 나은 결과를 보였다. 위 OLTP의 경우 CLW가 HC 보다는 좋은 결과를 보이나 HWC가 그 보다 좋은 결과를 보이는 것을 확인할 수 있다. 또한 HWLC는 모든 데이터에서 가장 좋은 결과를 보였다.

테스트 데이터 상에서 테스트 데이터 군집을 테스트했을 때, NAND Write가 5,826,693에서 HWLC 결과가 4,294,552로 감소해 26% 감소율을 보였다. 그러나 훈련데이터 군집과 테스트 데이터 군집의 차이로 인해 훈련 데이터를 통해 학습한 모델이 테스트 데이터의 군집을 예측할 때 낮은 정확도를 보였다. 이로인해 훈련, 테스트 데이터로 나누기 이전 전체 데이터에 대해 군집화한 데이터로 훈련 데이터를 학습시켰을 때, 테스트 데이터에 대해서도 예

측 정확도가 개선되었다. 그러나 전체 데이터 군집으로 테스트 데이터를 테스트 했을 때 WA 개선도가 11%로 이전보다 낮은 개선도를 가졌다.이는 범용성의 증가가 개별 데이터 특화에 따른 개선율을 희생한다는 것을 보인다.

4. 연구 결과 분석 및 평가

4.1. 모델 학습 평가

4.1.1. 이진분류 모델

4.1.1.1. Plain LSTM

해당 모델은 Hot/Cold 이진분류를 수행하도록 설계하였다. 그러나 해당 모델은 초기에 20 에포크 만큼 학습을 하도록 하였는데 15번째 에포크에서 빠른 종료 함수가 발동되어 종료되었고 이는 손실 값이 개선되지 않았음을 의미한다. 또한 전체 에포크 중 첫번째 에포크가 검증 데이터에 대해 가장 높은 정확도와 가장 적은 손실 값을 보였다. 오히려 에포크가 증가함에 따라 검증 데이터에서의 결과는 악화되었다. 결과로 세번째 에포크 부터 모델이 과적합을 일으키기 시작했다.

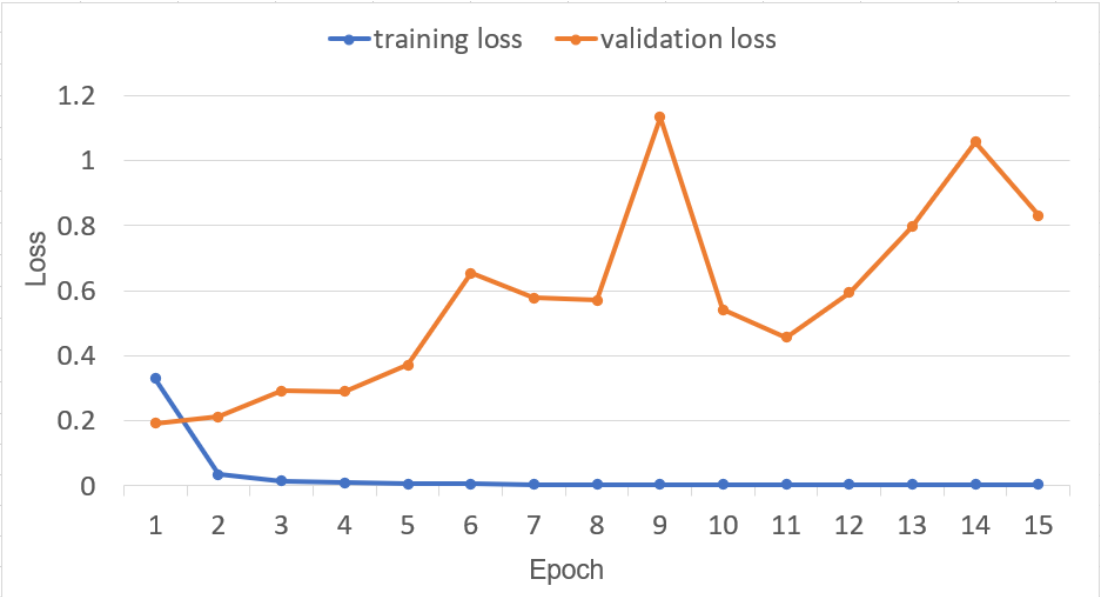


그림 17 Plain LSTM 손실 값 추이. 파란선은 훈련 손실 값을 의미하고 주황선은 검증 손실 값을 의미한다.

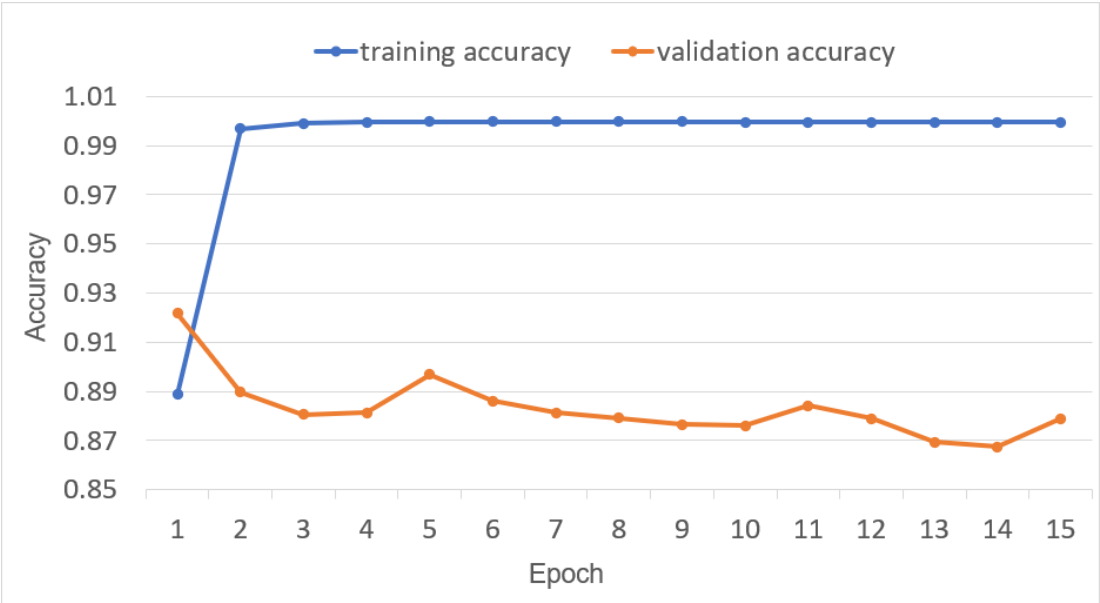


그림 18 Plain LSTM 정확도 추이. 파란선은 훈련 정확도를 의미하고 주황선은 검증 정확도를 의미한다

표 4 Plain LSTM 에포크당 정보

	train_loss	train_accuracy	val_loss	val_accuracy
epoch1	0.3296	0.8887	0.1926	0.9217
epoch2	0.0348	0.997	0.2129	0.8896
epoch3	0.0145	0.999	0.2925	0.8805
epoch4	0.0094	0.9995	0.2903	0.8814
epoch5	0.0068	0.9997	0.3709	0.8969
epoch6	0.0059	0.9997	0.6533	0.886
epoch7	0.0053	0.9997	0.5776	0.8814
epoch8	0.0051	0.9997	0.5714	0.8792
epoch9	0.0051	0.9997	1.1331	0.8766
epoch10	0.0049	0.9996	0.5411	0.876
epoch11	0.0049	0.9996	0.4568	0.8842
epoch12	0.0050	0.9996	0.5947	0.8790
epoch13	0.0049	0.9996	0.7966	0.8692
epoch14	0.0051	0.9996	1.0582	0.8673
epoch15	0.0051	0.9996	0.8310	0.8788

표 5 Plain LSTM 손실 값 및 정확도

Evaluation loss	Evaluation accuracy
0.8855	0.8708

4.1.1.2. CNN-LSTM

해당 모델역시 Hot/Cold 이진 분류를 수행한다. 해당 모델은 매 에포크 마다 정확도가 늘었다. 그러나 아래 표 5번에서 확인할 수 있듯이 검증 데이터 상의 최대 정확도는 3번째 에포크에서 발생했고 이는 0.9225였다. 이는 해당 모델에서도 3번째 에포크부터 과적합이 발생했다는 것인데 이는 아래 그림을 통해 볼 수 있듯이 검증 데이터 상의 정확도가 감소하는 것을 통해 확인할 수 있다. 해당 모델은 위 Plain LSTM과 달리 빠른 종료 함수 발동 없이 20개 에포크를 모두 훈련했다.

표 6 CNN-LSTM 손실 값 및 정확도

Evaluation loss	Evaluation accuracy
0.4433	0.8756

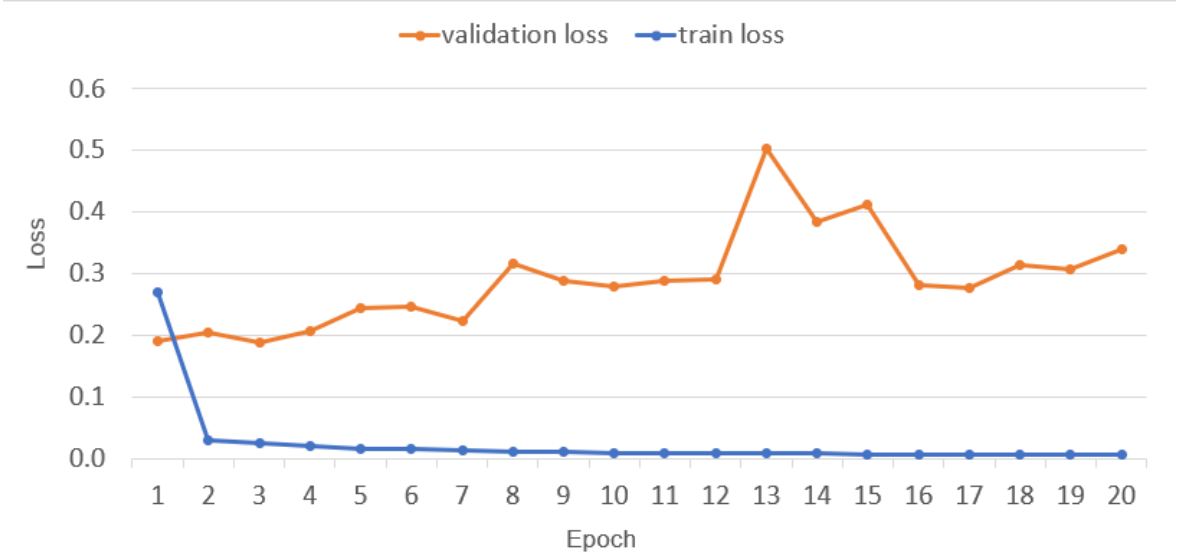


그림 19 CNN LSTM 손실 값 추이. 검증 데이터 상 손실 값이 지속적으로 상승한다.

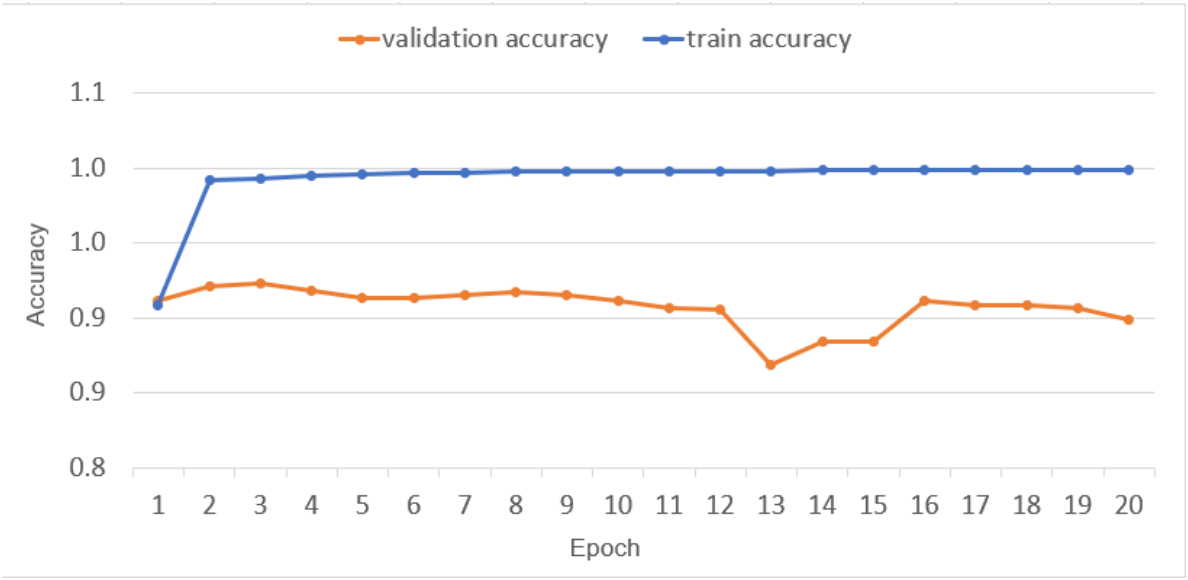


그림 20 CNN LSTM 정확도 추이. 3번째 에포크부터 검증데이터 정확도가 감소한다.

표 7 CNN LSTM 에포크 별 결과

	train_loss	train_accuracy	val_loss	val_accuracy
epoch1	0.2705	0.9086	0.1915	0.9115
epoch2	0.0303	0.9917	0.2047	0.9209
epoch3	0.0257	0.9932	0.1893	0.9225
epoch4	0.0204	0.9948	0.2070	0.9183
epoch5	0.0172	0.9958	0.2436	0.9134
epoch6	0.0150	0.9965	0.2460	0.9132
epoch7	0.0130	0.9970	0.2242	0.9155
epoch8	0.0120	0.9973	0.3162	0.9172
epoch9	0.0114	0.9975	0.2879	0.9150
epoch10	0.0099	0.9979	0.2799	0.9112
epoch11	0.0094	0.9980	0.2886	0.9062
epoch12	0.0090	0.9980	0.2901	0.9056
epoch13	0.0085	0.9982	0.5023	0.8689
epoch14	0.0082	0.9982	0.3849	0.8839
epoch15	0.0079	0.9983	0.4114	0.8836
epoch16	0.0073	0.9985	0.2812	0.9109
epoch17	0.0065	0.9987	0.2771	0.9087
epoch18	0.0060	0.9988	0.3151	0.9087
epoch19	0.0061	0.9988	0.3083	0.9060
epoch20	0.0058	0.9989	0.3394	0.8985

4.1.1.3. Stacked LSTM

해당 모델 역시 Hot/Cold 이진분류를 수행했다. 해당 모델에선 2번째 에포크가 검증 데이터 상에서 가장 좋은 결과를 보인다. 그 이후로는 과적합을 보인다. 검증데이터 상 정확도 역시 첫번째 에포크 이후로 감소한다. 훈련데이터상에서는 지속적인 개선을 보였고 빠른 종료

함수 역시 호출되지 않았다. 해당 모델은 다른 모델과 비교했을 때 더 낮은 정확도를 보였다.

표 8 Stacked LSTM 손실 값 및 정확도

Evaluation loss	Evaluation accuracy
1.064	0.8173

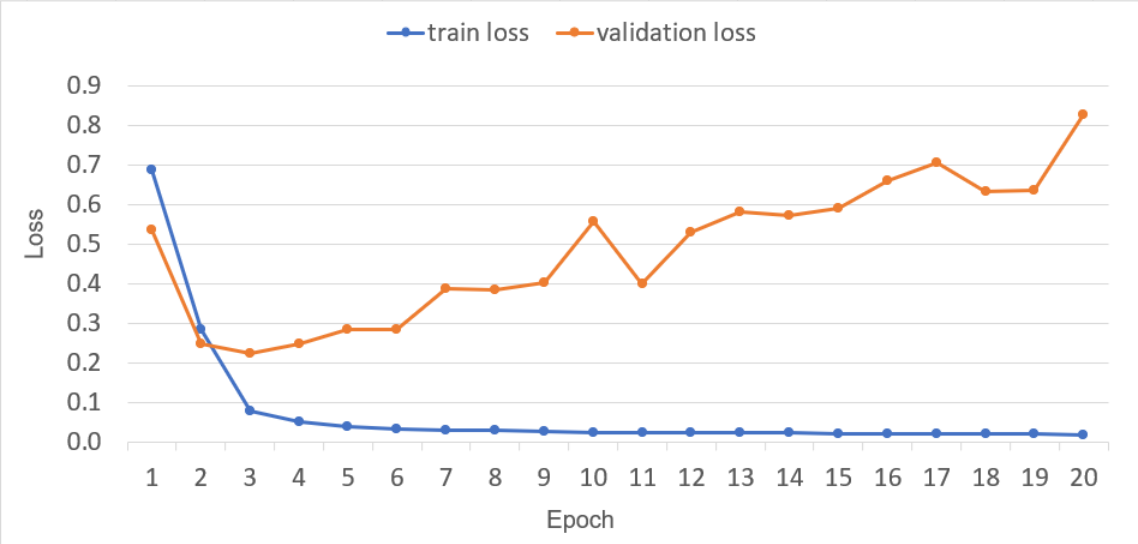


그림 21 Stacked LSTM 손실 값 추이. 검증 데이터상 3번째 에포크까지 개선을 보이다 이후 악화한다.

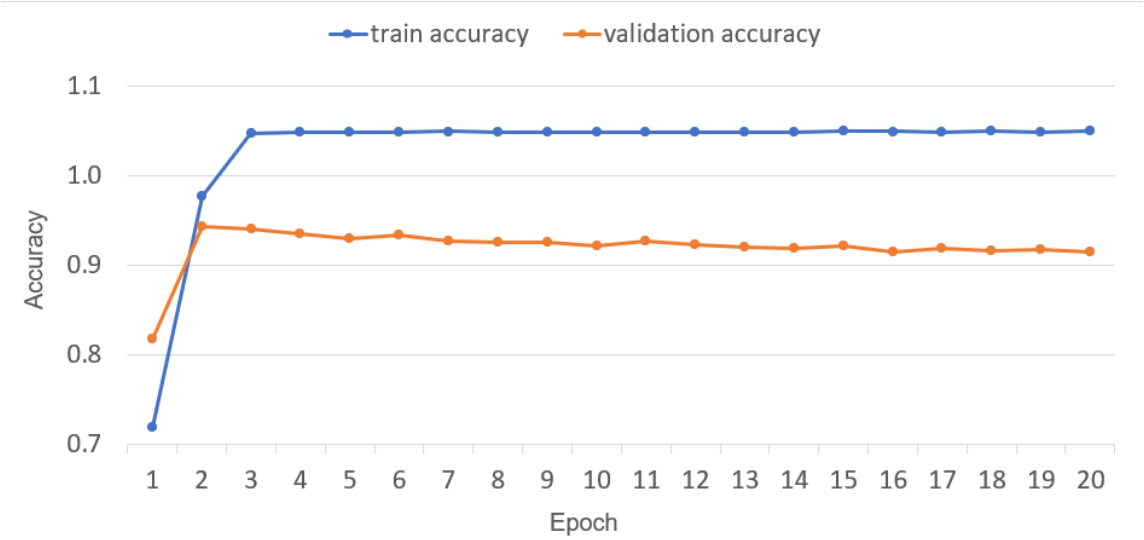


그림 22 Stacked LSTM 정확도 추이. 검증 데이터상 2번째 에포크 이후로 정확도가 하락하기 시작한다.

표 9 Stacked LSTM 에포크별 정보

	train_loss	train_accuracy	val_loss	val_accuracy
epoch1	0.6862	0.6696	0.5368	0.7684
epoch2	0.2838	0.9265	0.2491	0.8935
epoch3	0.0793	0.9973	0.2255	0.8913
epoch4	0.0512	0.9992	0.2478	0.8855
epoch5	0.0400	0.9994	0.2847	0.8801
epoch6	0.0340	0.9994	0.2855	0.8843
epoch7	0.0303	0.9994	0.3863	0.8770
epoch8	0.0290	0.9993	0.3861	0.8761
epoch9	0.0282	0.9993	0.4026	0.8763
epoch10	0.0258	0.9994	0.5561	0.8718
epoch11	0.0250	0.9994	0.3999	0.8771
epoch12	0.0242	0.9994	0.5292	0.8734
epoch13	0.0242	0.9993	0.5822	0.8707
epoch14	0.0236	0.9993	0.5734	0.8693
epoch15	0.0221	0.9995	0.5917	0.8714
epoch16	0.0219	0.9994	0.6616	0.8644
epoch17	0.0219	0.9994	0.7047	0.8686
epoch18	0.0207	0.9995	0.6314	0.8660
epoch19	0.0214	0.9994	0.6357	0.8679
epoch20	0.0194	0.9995	0.8272	0.8648

4.1.2. 다중분류 모델

4.1.2.1. Plain LSTM

해당 모델은 향후 실험할 모델들과의 비교대상을 두기 위해 실험했다. 해당 모델부터 Hot Cold 분류가 아닌 Cold Cool Warm으로 분류했는데, 이는 시뮬레이터 실험을 통해 테스트 데이터 상에서 Hot Warm Cold 분류보다 Cold Cool Warm 분류가 더 나은 WA 개선도를 보였기 때문이다. 해당 모델에서 두번째 훈련 에포크부터 과적합이 발생했다. 검증 에포크는 첫번째 에포크가 가장 좋은 결과를 보였고 이후 악화했다. 훈련 에포크에서도 6번째 에포크에서 가장 높은 정확도를 보였으며 이는 과접합의 또다른 증거를 보인다.

표 10 Plain LSTM 다중분류 손실 값 및 정확도

Evaluation loss	Evaluation accuracy
1.315	0.849

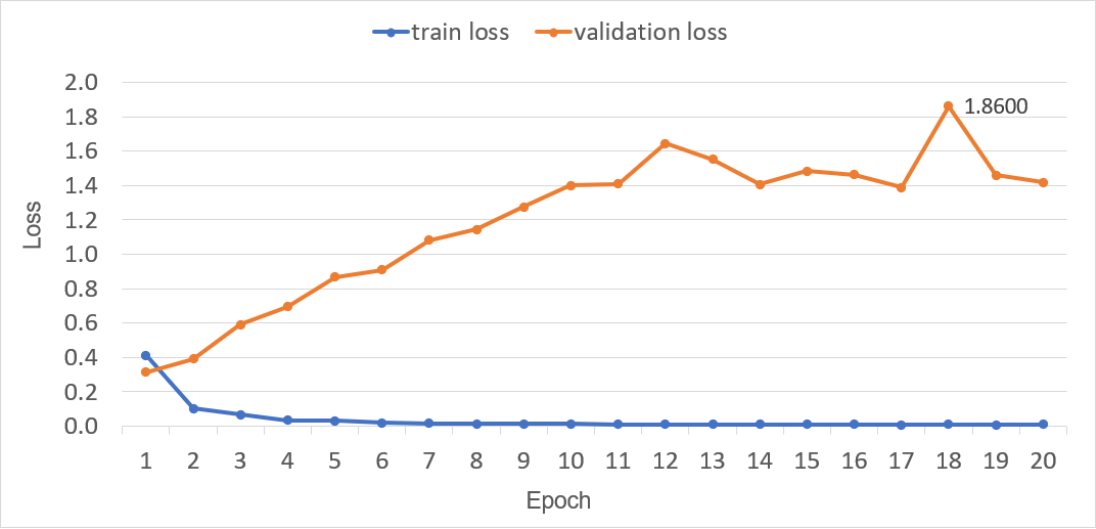


그림 23 Plain LSTM 다중분류 손실 값. 검증 데이터 상에서 지속적인 악화를 보인다.

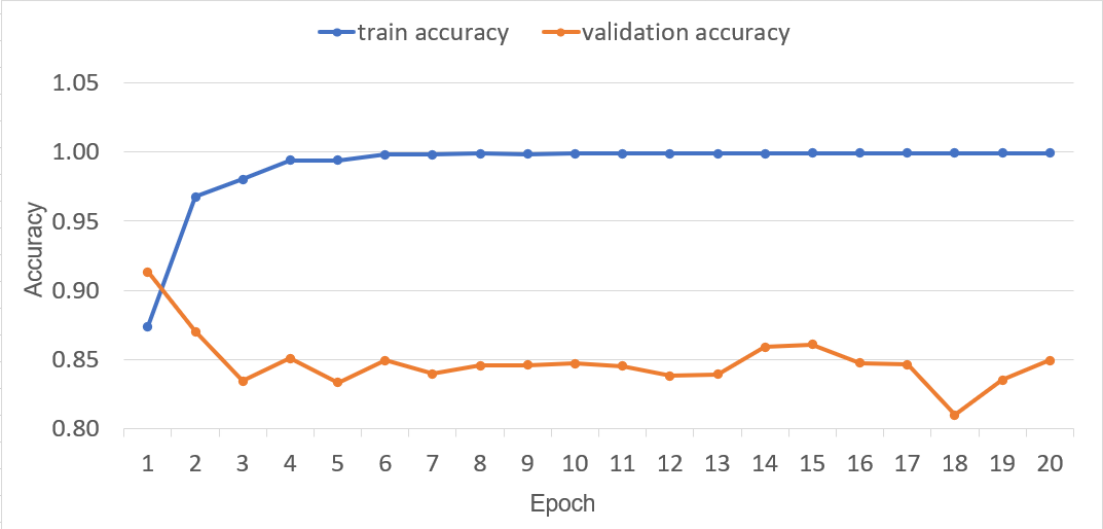


그림 24 Plain LSTM 정확도 추이. 검증 데이터 상에서 첫 에포크가 가장 높은 정확도를 보였다.

표 11 Plain LSTM 다중분류 에포크 당 정보

	train_loss	train_accuracy	val_loss	val_accuracy
epoch1	0.4124	0.8739	0.3141	0.9134
epoch2	0.1042	0.9676	0.3914	0.8700
epoch3	0.0673	0.9804	0.5903	0.8346
epoch4	0.0346	0.9940	0.6955	0.8507
epoch5	0.0315	0.9942	0.8681	0.8332
epoch6	0.0189	0.9981	0.9097	0.8495
epoch7	0.0177	0.9979	1.0805	0.8398
epoch8	0.0133	0.9989	1.1458	0.8458
epoch9	0.0136	0.9986	1.2750	0.8460
epoch10	0.0126	0.9987	1.4006	0.8471
epoch11	0.0121	0.9988	1.4099	0.8454
epoch12	0.0112	0.9989	1.6439	0.8383
epoch13	0.0107	0.9990	1.5529	0.8392
epoch14	0.0110	0.9989	1.4077	0.8591
epoch15	0.0101	0.9991	1.4847	0.8609
epoch16	0.0100	0.9991	1.4631	0.8476
epoch17	0.0093	0.9992	1.3886	0.8463
epoch18	0.0097	0.9991	1.8600	0.8097
epoch19	0.0089	0.9992	1.4585	0.8353
epoch20	0.0097	0.9992	1.4196	0.8495

4.1.2.2. CNN LSTM 다중분류

3번째 에포크에서 검증 데이터상 가장 높은 정확도를 보였다. 이것은 3번째 에포크부터 과적합을 보인다는 것을 의미한다. 에포크가 진행함에 따라 검증 에포크 정확도가 9% 가까이 떨어졌고, 손실 값은 2배 이상 증가했다. 그 결과 Plain LSTM 모델보다 더 낮은 정확도를 보였고 그 결과는 아래 표와 그림에서 확인할 수 있다.

표 12 CNN LST 다중분류 손실 값 및 정확도

Evaluation loss	Evaluation accuracy
1.346	0.817

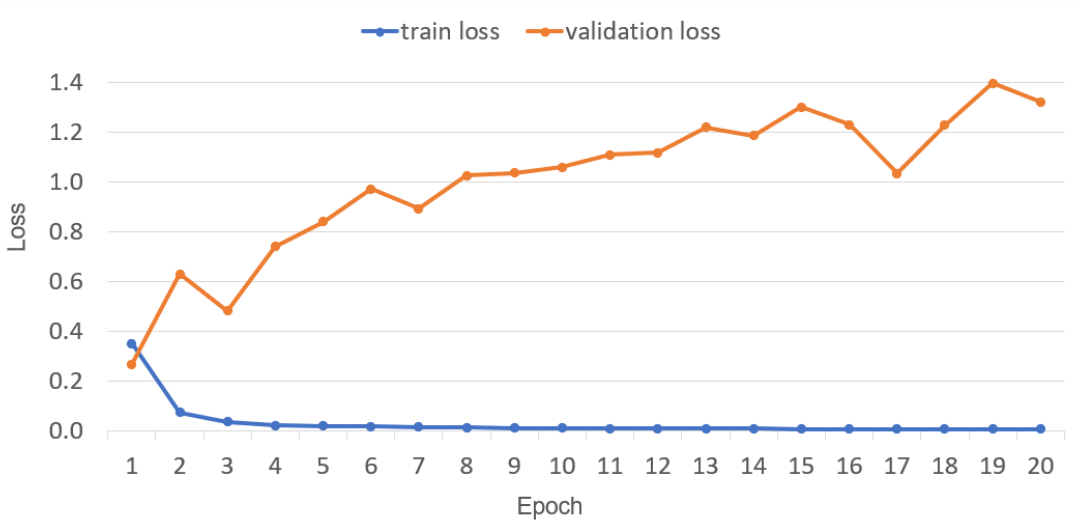


그림 25 CNN LSTM 다중분류 손실 값 추이. 첫번째 에포크에서 가장 낮은 손실 값을 보인다.

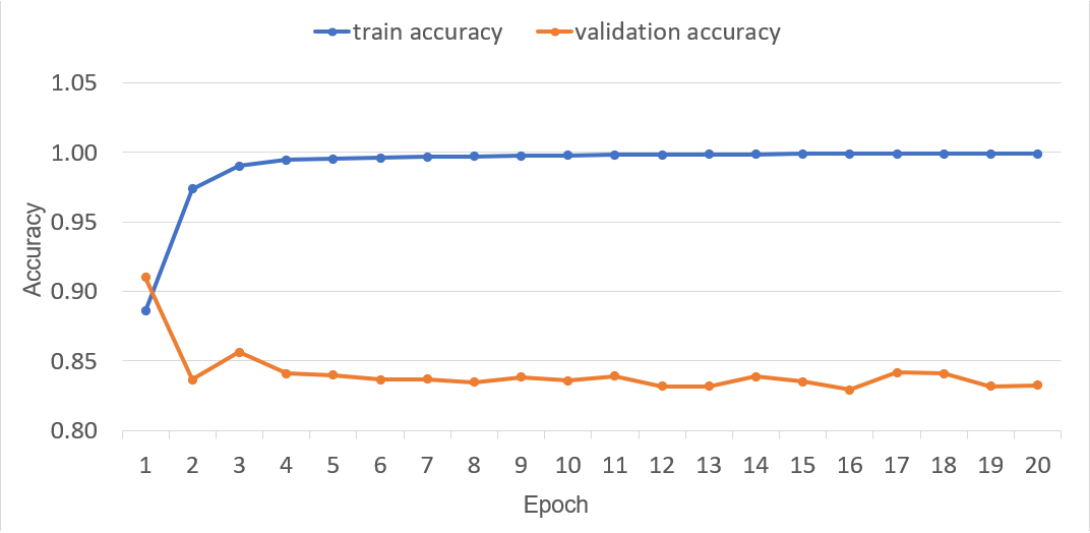


그림 26 CNN LSTM 다중분류 정확도 추이. 첫번째 에포크에서 가장 높은 정확도를 보인다.

표 13 CNN LSTM 다중분류 에포크 당 정보

	train_loss	train_accuracy	val_loss	val_accuracy
epoch1	0.3523	0.8859	0.2682	0.9101
epoch2	0.0750	0.9738	0.6313	0.8364
epoch3	0.0358	0.9903	0.4818	0.8563
epoch4	0.0231	0.9946	0.7409	0.8409
epoch5	0.0206	0.9954	0.8397	0.8397
epoch6	0.0176	0.9962	0.9717	0.8364
epoch7	0.0151	0.9969	0.8938	0.8370
epoch8	0.0137	0.9973	1.0263	0.8347
epoch9	0.0124	0.9976	1.0366	0.8385
epoch10	0.0109	0.9980	1.0587	0.8356
epoch11	0.0106	0.9981	1.1103	0.8393
epoch12	0.0098	0.9983	1.1167	0.8318
epoch13	0.0090	0.9985	1.2198	0.8320
epoch14	0.0087	0.9986	1.1865	0.8389
epoch15	0.0080	0.9988	1.3006	0.8352
epoch16	0.0078	0.9988	1.2297	0.8294
epoch17	0.0074	0.9989	1.0345	0.8418
epoch18	0.0071	0.9990	1.2285	0.8411
epoch19	0.0073	0.9990	1.3969	0.8317
epoch20	0.0069	0.9991	1.3216	0.8326

4.1.2.3. Stacked LSTM 다중분류

모델 실험 결과 실험 모델들 중 가장 높은 검증 정확도를 보였다. 그에 따라서 해당 모델의 예측 값을 토대로 시뮬레이팅을 수행하기로 결정했다. 해당 모델 역시 3번째 에포크부터 과적합하기 시작하여, 일단은 3 에포크 만큼 훈련시키고 하이퍼파라미터, 정규화, LSTM 레이어 수 조절, 클래스 가중치 조절등을 통한 정확도 개선을 시도했다.

표 14 Stacked LSTM 다중분류 에포크당 정보.

	train_loss	train_accuracy	val_loss	val_accuracy
epoch1	0.5036	0.8705	0.4200	0.9131
epoch2	0.3827	0.8772	0.4141	0.9110
epoch3	0.1568	0.9613	0.4769	0.9235
epoch4	0.1124	0.9765	0.5316	0.9129
epoch5	0.0846	0.9898	0.6312	0.9103
epoch6	0.0820	0.9879	0.6115	0.8699
epoch7	0.0660	0.9945	0.7567	0.8729
epoch8	0.0703	0.9920	0.7617	0.8708
epoch9	0.0598	0.9955	0.8383	0.8696
epoch10	0.0662	0.9930	0.8078	0.8437
epoch11	0.0587	0.9954	0.9084	0.8470
epoch12	0.0642	0.9937	1.1234	0.8345
epoch13	0.0564	0.9958	0.9287	0.8658
epoch14	0.0624	0.9943	1.0015	0.8360
epoch15	0.0575	0.9954	1.1093	0.8285

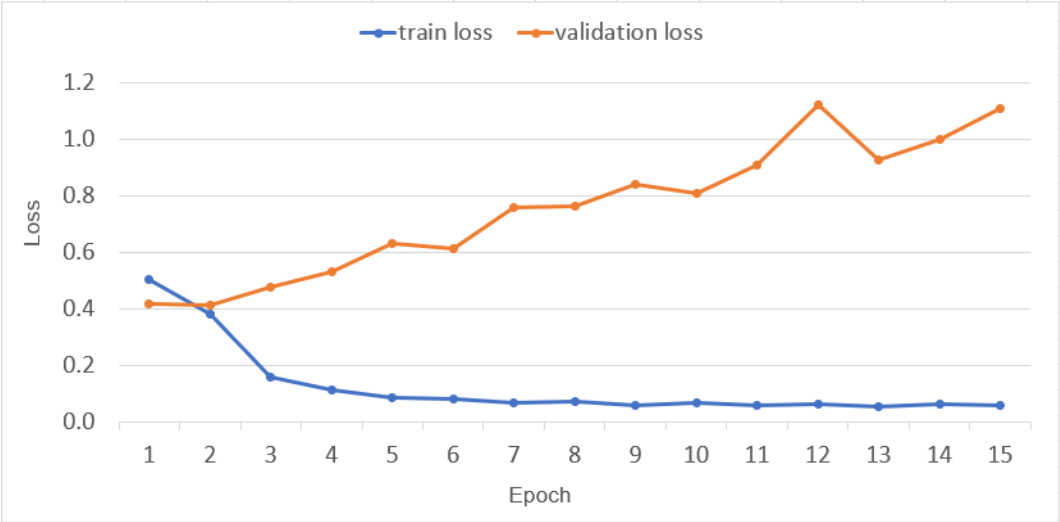


그림 27 Stacked LSTM 다중분류 손실 값 추이.

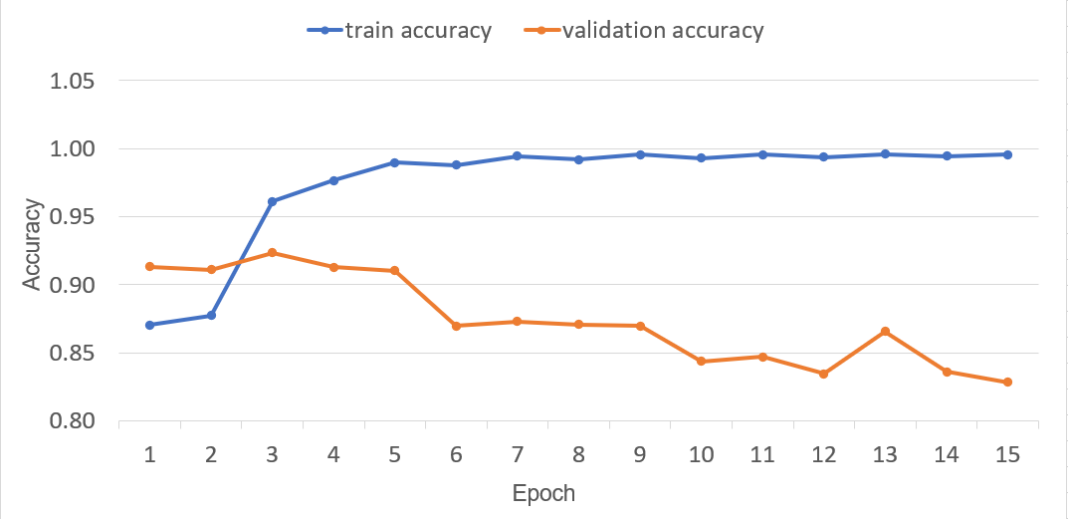


그림 28 Stacked LSTM 다중분류 정확도 추이

표 15 Stacked LSTM 다중분류 손실 값 및 정확도

Evaluation loss	Evaluation accuracy
1.064	0.8173

4.1.3. 하이퍼파라미터 튜닝

4.1.3.1. 클래스 가중치 변화

하이퍼파라미터 변화에 따른 Stacked LSTM 다중분류 모델의 변화를 실험했다. 이러한 하

이퍼파라미터 변화가 검증 데이터 상의 정확도 향상 및 손실 값 하락을 가져올 것을 예상했다. 앞선 3-2-2-2절에서 설명했듯이 클래스 별 가중치 조절을 통한 과적합 감소를 실험했다. 파라미터는 weight = {0: 50, 1: 50, 2: wcRatio}로 두었는데 wcRatio는 다음과 같이 계산했다.

```
coldCount = windowTrain["cold"].value_counts()[1]
coolCount = windowTrain["cool"].value_counts()[1]
warmCount = windowTrain["warm"].value_counts()[1]
wcRatio = (coldCount + coolCount) / warmCount
```

각 0번부터 Cold Cool Warm을 의미하고, Warm에 주어지는 가중치인 weRatio는 해당 경우에 약 0.149로 계산되었다. Batch_size 256에 위와 같은 가중치로 Stacked-CNN 다중분류 모델에 실험한 결과는 아래와 같다.

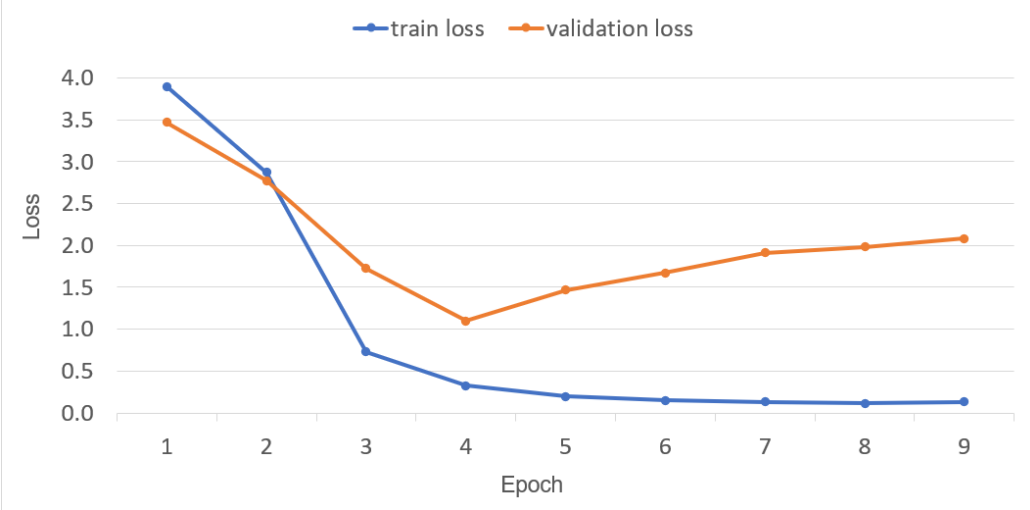


그림 29 Stacked-LSTM 다중분류 모델 가중치 변화 손실 값 추이.

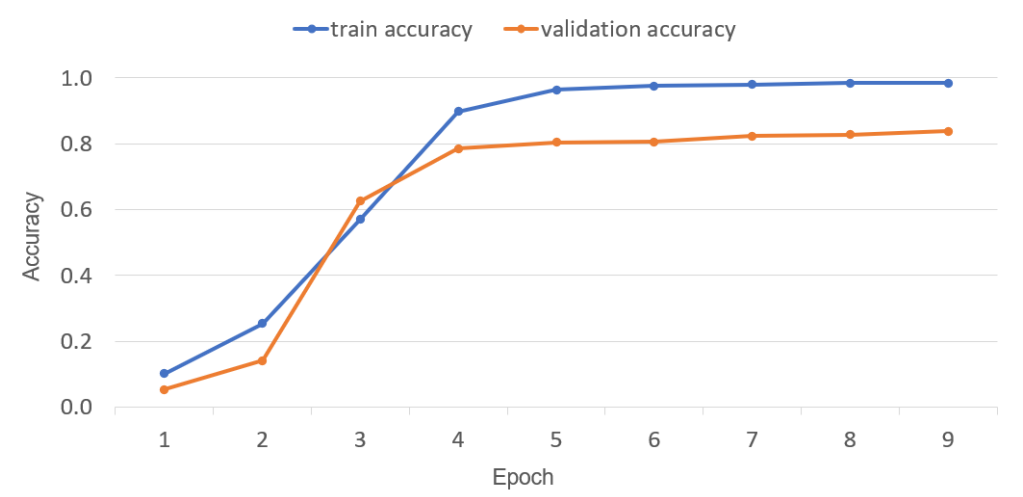


그림 30 Stacked-LSTM 다중분류 모델 가중치 변화 정확도 추이.

표 16 Stacked-LSTM 다중분류 모델 가중치 변화 에포크당 정보.

	train_loss	train_accuracy	val_loss	val_accuracy
epoch1	3.8944	0.1022	3.4675	0.0532
epoch2	2.8749	0.2546	2.7736	0.1420
epoch3	0.7318	0.5698	1.7238	0.6264
epoch4	0.3277	0.8979	1.1011	0.7853
epoch5	0.1985	0.9637	1.4672	0.8037
epoch6	0.1522	0.9757	1.6750	0.8060
epoch7	0.1354	0.9805	1.9108	0.8226
epoch8	0.1147	0.9847	1.9852	0.8279
epoch9	0.1312	0.9845	2.0806	0.8373

결과를 통해 볼 수 있듯이 에포크가 진행함에 따라서 검증 정확도가 늘어나다 9번째 에포크에서 가장 높은 정확도를 보인다. 해당 모델에 대해 에포크 9까지만 테스트했는데 이는 그보다 높은 에포크를 실험했을 때 손실 값이 급속도로 증가하여 과적합하는 모습을 보였기 때문이다. 이에 따라서 가중치를 변화시켜 테스트를 진행해보았다. 다음은 가중치를 표기와 같이 weight = {0: 60, 1: 15, 2: 1}로 하였을 때의 결과이다.

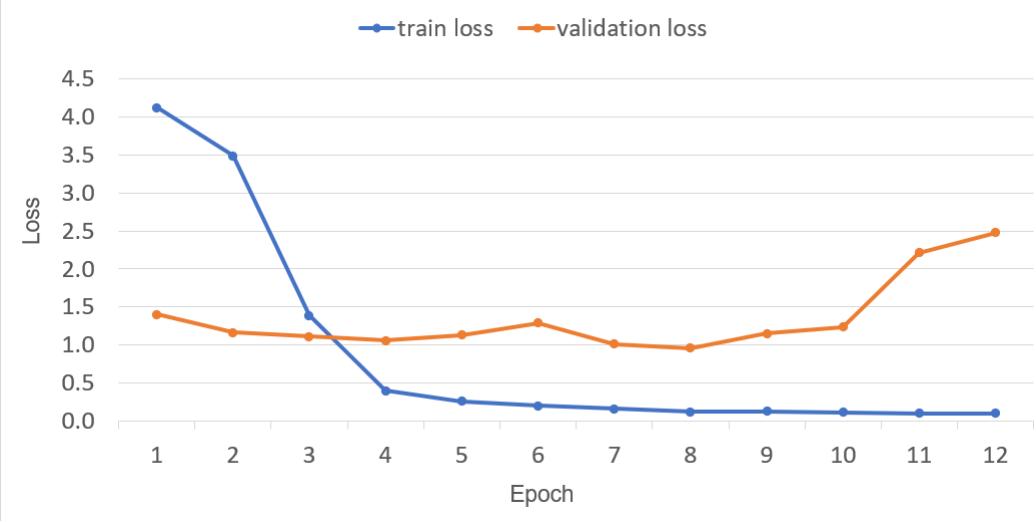


그림 31 Stacked-LSTM 다중분류 모델 가중치 변화 손실 값 추이

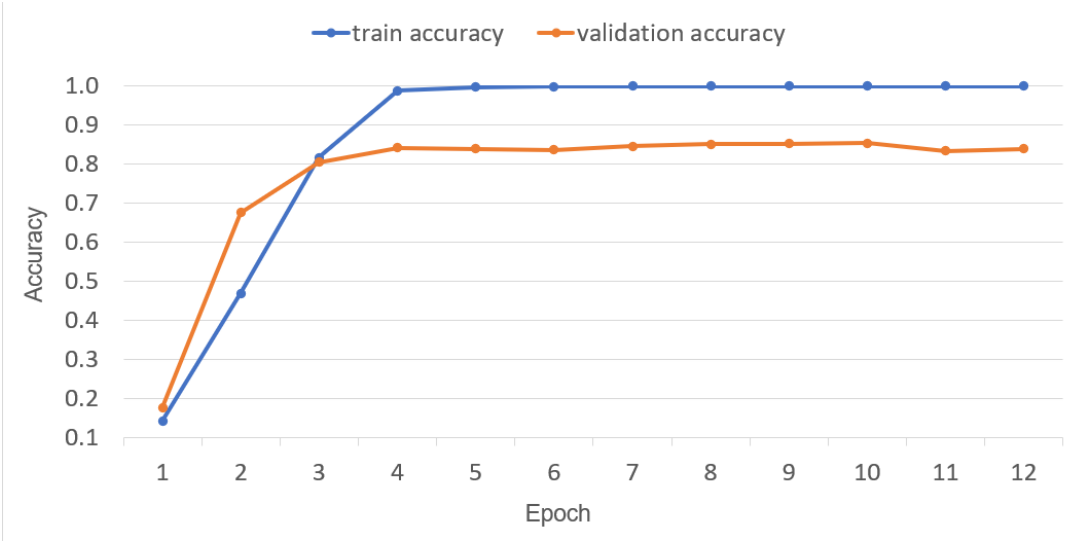


그림 32 Stacked-LSTM 다중분류 모델 가중치 변화 정확도 추이

이전 실험과 비교했을 때, 검증데이터상 정확도 및 손실 값의 방향이 더 좋았기에 이전보다 더 많은 12개 에포크에 대해 실험을 진행했다. 결과 10번째 에포크에서 가장 높은 정확도를 보였다. 이때 가장 높은 정확도를 보였으나 손실 값이 가장 낮지는 않았다. 그러나 결과적으로 해당 가중치가 이전 가중치보다 더 나은 결과를 보였다.

표 17 Stacked-LSTM 가중치 변화 에포크당 정보

	train_loss	train_accuracy	val_loss	val_accuracy
epoch1	4.1242	0.1426	1.4038	0.1770
epoch2	3.4882	0.4693	1.1606	0.6756
epoch3	1.3853	0.8161	1.1125	0.8039
epoch4	0.3952	0.9866	1.0583	0.8406
epoch5	0.2565	0.9955	1.1300	0.8381
epoch6	0.1955	0.9972	1.2902	0.8357
epoch7	0.1580	0.9979	1.0093	0.8450
epoch8	0.1214	0.9985	0.9580	0.8498
epoch9	0.1242	0.9984	1.1514	0.8509
epoch10	0.1125	0.9984	1.2325	0.8519
epoch11	0.0971	0.9988	2.2128	0.8332
epoch12	0.1026	0.9987	2.4794	0.8385

위와 같은 가중치를 이번에는 CNN-LSTM 모델에 테스트해보았다. 27 에포크를 훈련시켰으며 검증 데이터상 정확도가 에포크마다 증가했다. 그러나 문제는 검증 데이터상 손실 값 역시 함께 올랐다는 것이다. 이 역시 과적합의 요소로 볼 수 있다.

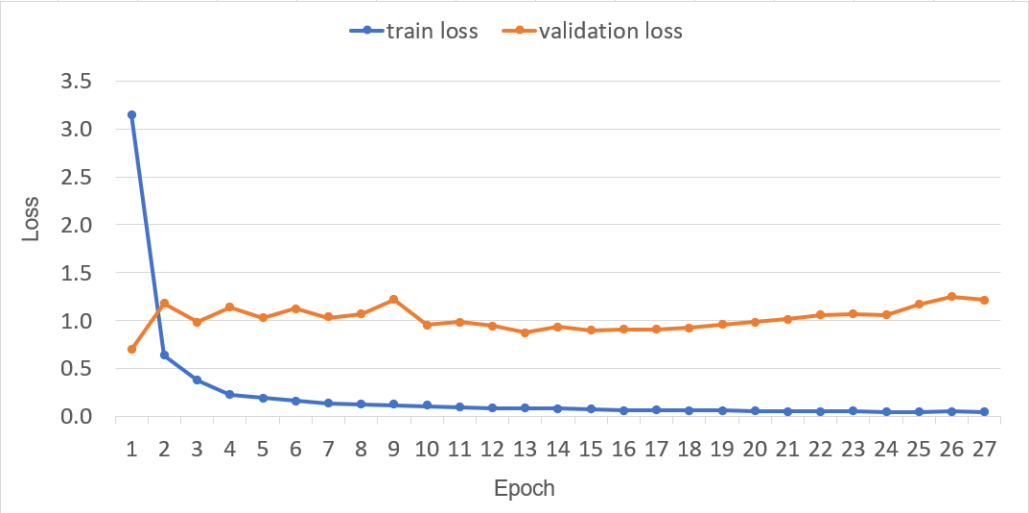


그림 33 CNN-LSTM 가중치 변화 손실 값 추이

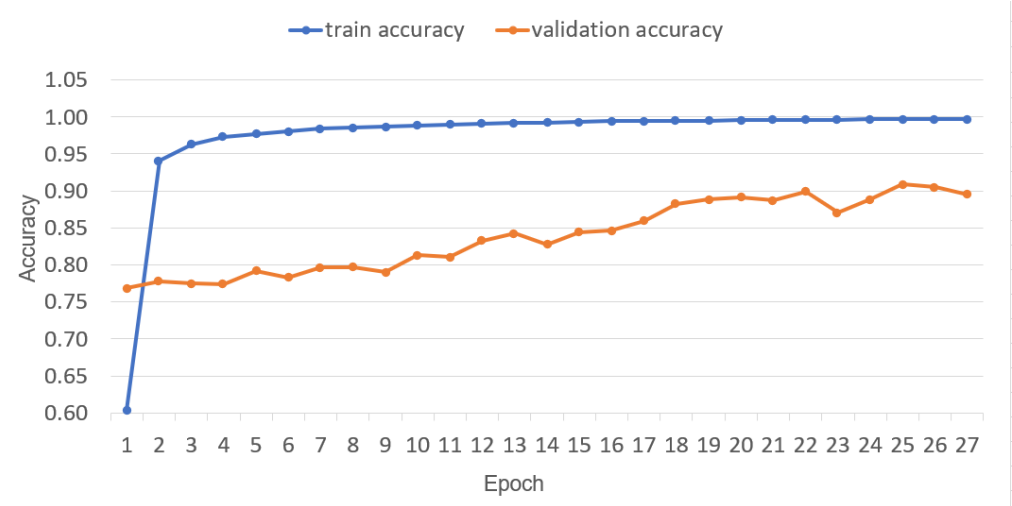


그림 34 CNN-LSTM 가중치 변화 정확도 추이

표 18 CNN-LSTM 가중치 변화 에포크당 정보

	train_loss	train_accuracy	val_loss	val_accuracy
epoch1	3.1491	0.6031	0.7022	0.7687
epoch2	0.6340	0.9410	1.1804	0.7781
epoch3	0.3766	0.9631	0.9838	0.7752
epoch4	0.2306	0.9730	1.1396	0.7744
epoch5	0.1947	0.9772	1.0272	0.7919
epoch6	0.1640	0.9806	1.1268	0.7833
epoch7	0.1407	0.9842	1.0312	0.7965
epoch8	0.1295	0.9856	1.0706	0.7974
epoch9	0.1221	0.9867	1.2208	0.7900
epoch10	0.1091	0.9887	0.9557	0.8130
epoch11	0.0997	0.9901	0.9856	0.8106
epoch12	0.0903	0.9914	0.9472	0.8332
epoch13	0.0890	0.9917	0.8749	0.8427
epoch14	0.0858	0.9925	0.9353	0.8278
epoch15	0.0782	0.9934	0.8934	0.8444
epoch16	0.0669	0.9944	0.9089	0.8461
epoch17	0.0676	0.9946	0.9098	0.8597
epoch18	0.0637	0.9951	0.9240	0.8828
epoch19	0.0640	0.9952	0.9627	0.8886
epoch20	0.0603	0.9956	0.9845	0.8915
epoch21	0.0551	0.9962	1.0149	0.8871
epoch22	0.0547	0.9963	1.0623	0.8993
epoch23	0.0586	0.9960	1.0723	0.8706
epoch24	0.0521	0.9966	1.0615	0.8888
epoch25	0.0491	0.9970	1.1705	0.9087
epoch26	0.0534	0.9966	1.2522	0.9048
epoch27	0.0507	0.9969	1.2177	0.8953

결과는 검증 데이터상에서 90%의 정확도를 보였다. 이에 앞선 가중치 변화 Stacked-LSTM 모델보다 WA 개선에 있어 더 좋은 결과를 보일 것이라 예상했다. 그러나 시뮬레이션 결과 Stacked-LSTM에 비해 저조한 결과를 보였다. 이에 해당 모델을 주 모델로 선택하지 않았다.

4.1.3.2. 배치 사이즈 튜닝

위 실험에서는 배치 사이즈를 256으로 설정했다. 과적합에 대한 조사 결과 큰 배치 사이즈가 과적합을 일으킬 수 있다. 다음 그림과 표들은 배치 사이즈가 128 일 때 변화를 보인다.

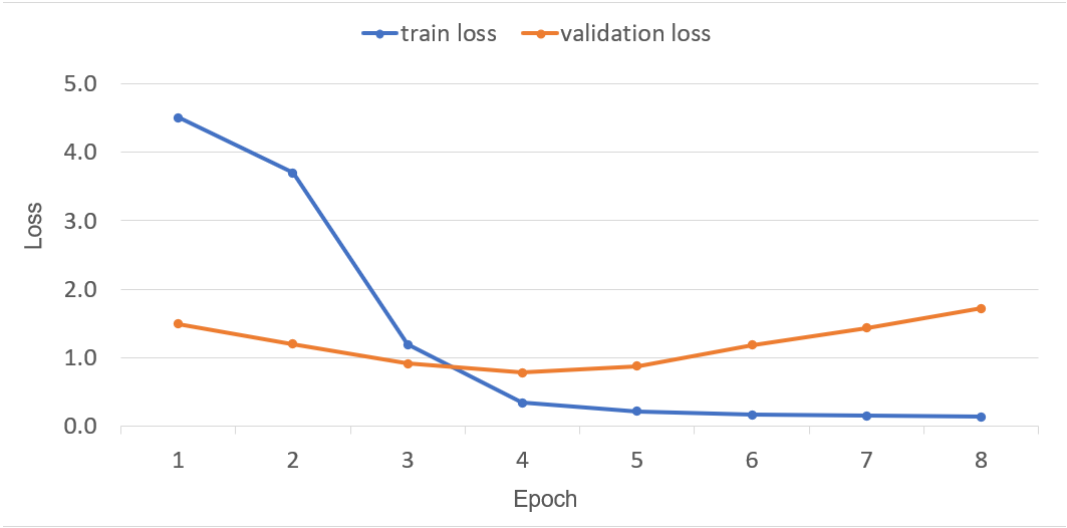


그림 35 Stacked-LSTM 다중분류 배치 사이즈 128 손실 값 추이

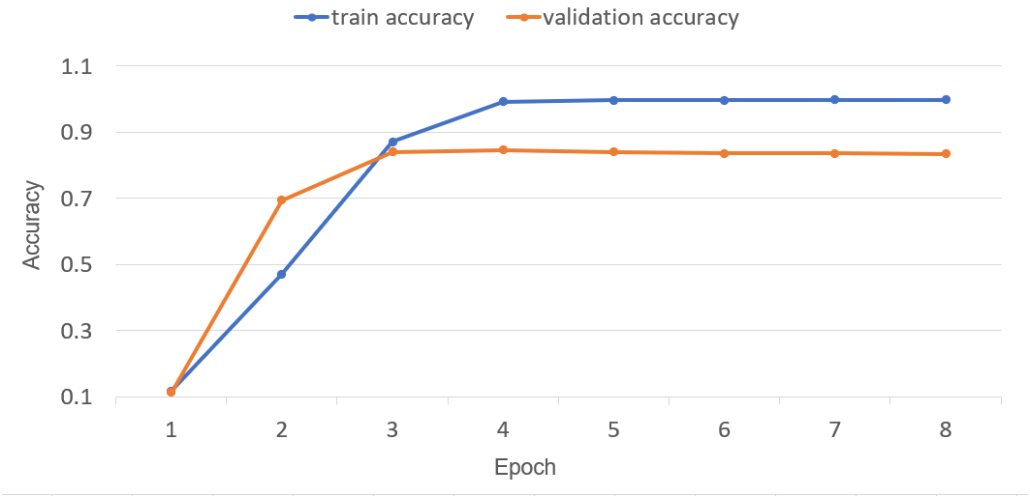


그림 36 Stacked-LSTM 다중분류 배치 사이즈 128 정확도 추이

표 19 Stacked-LSTM 배치 사이즈 128 에포크당 결과

	train_loss	train_accuracy	val_loss	val_accuracy
epoch1	4.5047	0.1173	1.4898	0.1136
epoch2	3.6996	0.4716	1.1974	0.6953
epoch3	1.1888	0.8708	0.9093	0.8404
epoch4	0.3388	0.9926	0.7769	0.8467
epoch5	0.2166	0.9965	0.8718	0.8404
epoch6	0.1614	0.9974	1.1854	0.8370
epoch7	0.1448	0.9979	1.4329	0.8366
epoch8	0.1333	0.9980	1.7148	0.8352

결과로 손실 값은 이전보다 더 줄일 수 있었으나 정확도 역시 더 낮아졌다. 4번째 에포크에서 가장 높은 정확도를 기록했다. 그러나 그 이후로 과적합 문제를 보인다. 훈련 데이터에서 정확도는 소폭 상승하기 시작했으나 검증 데이터 손실 값 역시 상승하기 시작했다.

4.2. WA 개선 정도 평가

아래 표는 이번 연구를 통해 실험한 모델들의 손실 값과 정확도의 모음이다.

표 20 모델 손실 값 및 정확도

	loss	accuracy
Plain-LSTM: 20 epochs, B1	1.3148	0.8489
CNN-LSTM: 20 epochs, B1	1.3456	0.8175
Stacked-LSTM: 15 epochs, B1	1.0635	0.8173
Stacked-LSTM: 3 epochs, B1	0.3234	0.9366
Stacked-LSTM: 9 epochs, weight-1, B1	2.461	0.8176
Stacked-LSTM: 12 epochs, weight-2, B1	2.5917	0.8227
Stacked-LSTM: 8 epochs, weight-3, B1	2.0098	0.8202
Stacked-LSTM: 10 epochs, weight-2, B1	1.8055	0.8202
CNN-LSTM: 27 epochs, weight-2, B1	0.8653	0.9066

weight-1: {cold: 50, cool: 50, hot: 1}
weight-2: {cold: 60, cool: 15, hot: 1}
weight-3: {cold: 60, cool: 20, hot: 1}
weight-4: {cold:20, cool: 4, hot: 1}
B1: 256
B2: 128

그림 37 weight 및 B의 의미. Weight는 가중치 부여, B는 배치 사이즈를 의미한다.

주황색으로 표기한 모델이 이번 실험에서 WA 개선도가 가장 큰 모델이다. 이를 다시 정리하면 아래 표와 같다.

표 21 모델별 WA 개선도

모델	NAND Write	WA 개선도
모델 사용 X	6,130,991	-
Stacked-LSTM: 9 epochs, weight-1, B1	5,964,122	2.72%
Stacked-LSTM: 12 epochs, weight-2, B1	5,964,122	2.72%
Stacked-LSTM: 8 epochs, weight-3, B1	5,526,428	9.86%
Stacked-LSTM: 10 epochs, weight-2, B1	5,526,428	9.86%
CNN-LSTM: 19 epochs, weight-2, B1	6,150,403	0.32%
CNN-LSTM Hot Warm Cool Cold	5,983,401	2.41%

표 21을 통해 가장 높은 개선도를 보인 모델은 Stacked-LSTM의 결과임을 알 수 있다. 그 중 눈에 띄는 결과는 Stacked-LSTM 10 ephocs와 12 ephocs 인데, 둘 다 같은 배치 사이즈와 가중치 부여가 사용된 경우이다. 표 20에서 에포크 12의 경우가 정확도가 0.0025 증가하고 손실 값이 0.79 증가한 것을 볼 수 있는데 그 결과 WA 개선도가 7.1% 감소했다. 이는 과적합이 결과에 악영향을 미침을 보인다.

CNN-LSTM의 경우 모든 경우에서 낮은 개선도만을 보였다. 또한 표 21 내에서 가장 낮은 손실율을 보인 두 모델만이 9.86%의 개선도를 갖았다. 이는 손실율이 정확도 보다 나은 지표임을 보여준다. 이들보다 손실율이 더 낮은 모델들은 다중분류가 아닌 이중 분류의 경우로 모두 4% 이하의 개선도를 보여 표에 반영하지 않았다. 마지막으로 Hot Warm Cool Cold로 분류하는 실험을 진행하였는데, CNN-LSTM의 경우 모두 좋지 않은 결과를 보였던 탓에 높은 개선도를 얻어내지는 못했다. 그러나 여전히 CNN-LSTM 중 가장 높은 개선도를 보였다. 이에 Stacked-LSTM에서 이와 같은 모델을 만들었을 때 9.86% 보다 높은 개선도를 보일 것을 기대했으나 일정내에 마무리하지 못하여 결과를 확인하지 못했다.

9.86% 개선은 기대했던 바에는 미치지 못하는 결과였다. 그러나 앞서 설명하였듯이 지엽적인 군집을 포기하고 더욱 범용적인 군집을 선택하였을 때 기대할 수 있는 최대 개선도는 11.4% 였으므로, 9.86%의 정확도는 크게 낮지 않은 결과이다. 이러한 이유로 인해 더 나은 군집을 만들거나, 지엽적 군집을 훈련해 낼 수 있다면 더 높은 개선도를 보일 수 있을 것이다. 여기서 지엽적 군집을 훈련해 낸다는 것은 각기 다른 분포를 갖는 워크로드 데이터의 지엽적 군집을 예측해낸다는 것을 의미하며, 이는 방대한 데이터에 대한 훈련을 요한다.

5. 결론 및 향후 연구 방향

이번 연구를 통해 Plain LSTM, CNN LSTM, Stacked LSTM 모델을 실험해보았다. 3가지 모델에 대한 실험 결과 Stacked LSTM이 가장 높은 정확도와 일정한 손실 값을 보였다. 그러나 여전히 3가지 모델 모두 과적합 문제가 지속적으로 발생했다. 이는 LSTM 모델이 일반적으로 갖는 문제이기에, 여러 정규화 기법들이 존재한다. 이에 해당 연구에서는 가중치 할당, 드롭아웃, L1L2 등을 실험했고 이중 가중치 할당이 가장 큰 변화를 일으키는 것을 확인했다. 그 이유는 사용되는 데이터가 편중도가 높고, 이로인해 과적합이 더욱 쉽게 발생하는 것이기 때문이다. 고로 가중치 할당을 통해 편중되어 있는 클래스에 대한 의존도를 의도적으로 낮추는 것이 가장 큰 효과를 보인 것이다.

해당 실험에서 군집화의 지엽성은 WA 개선도 저해에 영향을 주었다. 하나의 워크로드 데이터를 훈련, 검증, 테스트 데이터로 나누었음에도 각 데이터에 나타나는 Hot LBA와 Cold LBA의 비율이 다른 까닭에 훈련 데이터의 군집으로 훈련한 모델은 테스트 데이터의 군집을 예측해내는 데에 어려움을 겪었다. 이에 워크로드 데이터를 나누기 이전 상태에서 군집화를 수행한 후 해당 군집을 통해 나누어진 훈련 데이터를 훈련시키고 테스트 데이터에 대한 예측을 시도했다. 결과적으로 이전보다 더 높은 정확도를 보였다. 그러나 예측으로 인한 WA 개선도는 기대보다 저조한 결과를 보였다. 이는 26%의 WA 개선도를 보인 방식은 테스트 데이터만의 군집을 구하고 이에대한 시뮬레이션을 진행했다는 데에 있다. K-means가 해당 데이터의 지엽적 특성을 기반으로 군집을 진행했기에 높은 개선도를 보일 수 있었던 것이다. 그러나 실험 모델은 더 큰 범위의 전체 데이터에 대한 군집을 이용하였기에 범용성은 늘었으나 WA 개선도가 감소한 결과를 보인 것이다.

위 결과로부터 두 가지 문제가 대두된다는 것을 확인할 수 있다. 그 하나는 워크로드 데이터의 편중성으로 인한 과적합 유발이며, 다른 하나는 범용적 군집화의 부작용이다. 전자의 문제를 해결하기 위해서는 데이터 전처리 및 모델 정규화에 대한 추가적인 실험이 필요하다. 이는 그 양자가 데이터 편중성을 감소시키고, 과적합을 지연 또는 약화시킬 수 있기 때문이다. 후자의 문제는 더 어려운 문제라 할 수 있다. 군집화를 통한 레이블링이라는 방식은 K-means의 작동 특성으로 인해 군집화를 진행한 데이터에 종속적일 수밖에 없다. 이를 해결하여 더욱 범용성 있는 모델을 얻기 위해서는 방대한 데이터의 여러 군집을 모두 학습하거나, 군집화 이외의 방식을 시도해보아야 할 것이다. 앞서 자체적인 룰로 레이블링을 시도한

결과는 하나의 데이터에서는 Hot Cold Cool Warm 분류만큼의 개선도를 보였으나 다른 데이터에서는 오히려 WA가 악화되는 결과를 보였다. 이 또한 범용적인 분류 기준점을 찾는 것이 어려움을 시사한다. 이렇듯 예측을 기반으로 하는 온도 분류 방식은 온도를 구분하는 범용적이며 효과적인 방식을 찾아내는 것이 중요도를 갖는다 할 수 있다.

6. 구성원별 역할 및 개발 일정

6.1. 개발일정

6.2. 구성원별 역할

이름	역할
Ganchuluun Narantsatsralt	학습 데이터 전처리 모델 설계 및 개발 모델 훈련 및 평가
Ariunbold Odgerel	하이퍼파라미터 튜닝 모델 훈련 및 개선 데이터 분석
최성찬	레이블링 데이터 군집화 시뮬레이터 작성 및 테스트 WA 개선도 측정

7. 참고 문헌

[1] Ilhoon Shin. “Hot/cold clustering for page mapping in nand flash memory”, *IEEE Transactions on Consumer Electronics*, 57(4), 2011.

[2] B Li. “HAML-SSD: a hardware accelerated hotness-aware machine learning based SSD management”, *38th IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2019*, p. 8942140, 2019

[3] Y. h. Jeong, H. m. Jeong, J. h. Kim. “A Sophisticated Hotness Classification Scheme based on Machine Learning to Reduce SSD Write Amplification” *Proc. Of the KSC2021*, pp. 1637-1639, 2021. (in Korean)

[4] SNIA IOTTA Repository, Block I/O Traces. Available: http://iotta.snia.org/traces/block-io

[5] UMASS Trace Repository, OLTP Application I/O. Available: https://traces.cs.umass.edu/index.php/Storage/Storage