확률 그래프 모델에 의한 패턴 분류 및 완성

Pattern Classification and Completion by Probabilistic Library Model

2005년 12월

서울대학교

컴퓨터공학부

김 동 혁

지도 교수 : 장 병 탁

요약

본 논문에서는 PLM을 이용해서 숫자 인식에 사용하는 데이터를 분류하는 패턴 분류 문제와, 데이터를 적절한 방법으로 오염시킨 뒤, 이를 복구하는 패턴 완성 문제를 접근해보고자 한다. 그리고 이러한 패턴 분류와 패턴 완성이라는 학습 문제에 접근하기 위해서 숫자 인식 데이터를 여러 가지 정보 이론적인 측면에서 분석하였다. 이 중에서 엔트로피에 대한 분석이 문제 접근에 많은 도움을 주었다.

패턴 분류 시뮬레이션 결과 비트 1 인코딩에서는 83.19%, 비트 2 인코딩에서는 84.06%의 평균적인 인식률을 보였다. 또한 비트 수를 늘려갈수록 라이브러리에 들어가는 DNA 종류 수 보다는 늘어가는 비트 수가 인식률에 기여하는 점이 크다는 것을 확인할 수 있었다. 이는 서로 연관 관계를 갖는 여러 비트들에 대한 정보를 포함할 수 있는 가능성을 높여주는 것이, 랜덤하게 초기 DNA의 종류 수를 늘려주는 것보다 더 중요하다는 예측을 가능하게 했다.

여러 조건에서 실험을 반복해서 수행한 결과, 93.53%의 인식률을 보이는 조건을 찾아낼 수 있었다. 학습 데이터의 분포를 고려한 샘플링, 업데이트 비율, Threshold, 반복 횟수의 튜닝, 랜덤 샘플링 방식의 개선을 통해서 이러한 인식률을 얻을 수 있었다.

패턴 완성 시뮬레이션에서는, 20비트 정도의 고 차원의 인코딩의 경우에 대해 학습을 효율적으로 시키는 시뮬레이션은 이미 패턴 분류 문제에서 수행했기 때문에, 1 비트와 2 비트 인코딩을 한 경우에 Correlational PLM으로 오염된 데이터를 복구하는 것에 대해 테스트를 수행했다.

시뮬레이션 결과 1 비트 그리고 2 비트 인코딩 모두, 오염 횟수를 시킴에 따라 오염된 데이터의 인식률은 떨어지고, 엔트로피는 점점 증가했다. 그리고 라이브러리 내의 모든 DNA에 대해 오염된 데이터를 복구하도록 시도한 시뮬레이션 결과에서는 1 비트 인코딩의 경우에는 97~99%의 인식률 그리고 2 비트 인코딩의 경우에는 100%의 인식률을 보였다. 그리고 1 비트 인코딩의 경우에도 복구 시도 횟수를 늘리면 약 350회 정도에서 100%의 인식률을 보였다. 하지만 이 부분은 2 비트 인코딩의 경우 약 63배나 더 많은 복구 시도를 하였지만, 비슷한 엔트로피 차이와 인식률을 보이는 조건에서의 평균 복구 횟수는 19회 정도로 비슷하였으며, 복구 시도를 많이 할수록 인식률이 더 높아지는 경향도 비슷함을 확인할 수 있었다. 또한 실제 복구가 되어가는 데이터를 실제 비트맵의 형태로 출력해본 결과, 2비트 인코딩의 경우가 훨씬 더 원본 이미지에 가까우며 눈으로 식별하기에도 좋은 형태로 복구가 이루어졌다.

목차

그림 목차

표 목차

1 서론

1.1 연구 배경

1937년 알란 튜링(Alan Turing)에 의해 튜링 머신의 개념이 나오고, 1945년 폰 노이만(J. von Neumann)의 논문인 “전자계산기의 이론 설계 서론”에서 프로그램내장 방식을 발표한 이후로, 디지털 전자 컴퓨터는 하드웨어와 소프트웨어라는 개념의 분리를 통해서 발전해 왔다. 하지만 하드웨어의 물리적인 한계 때문에 튜링 머신이 수행할 수 있고, 현재의 디지털 컴퓨터도 이론적으로는 수행이 가능하지만, 효율적으로 혹은 실질적으로 수행할 수 없는 계산이 많아지고 있다.

현재의 실리콘 하드웨어에 기반한 컴퓨터 기술이 가장 취약점을 보이는 예의 하나로 인공지능 연구에서 찾을 수 있다. 인간의 언어 능력을 모사하기 위해서는 많은 수의 메모리 인자들의 상호작용에 기반한 정보 처리가 필요한데 지금까지의 순차적 규칙 적용 방식에만 의존하는 정보처리 하드웨어나 소프트웨어 방식으로는 이러한 현상을 시뮬레이션하는 것이 근본적으로 어렵다. 즉, 현재의 컴퓨터 기술은 기호기반의 순차적인 정보처리에는 적합하지만 많은 수의 정보소자들의 상호작용에 의해서 발생하는 복잡계의 현상을 모델링하는 데는 취약하다.[장 03]

하지만 자연에서 일어나는 정보처리 과정을 살펴보면, 현재의 컴퓨터가 수행하는 정보처리 방식과는 많은 차이가 있다. 우선 자연계에서는 소프트웨어(정보)와 하드웨어(물질)이 엄밀하게 분리되어 있지 않으며, 진화를 통해서 어려운 문제들을 최소의 에너지만으로 잘 풀 수 있는 방법들을 개발해 왔다.[장 03]

1.2 연구 목적

본 논문에서는 이러한 자연에서의 정보처리 과정을 이용하는 바이오분자 컴퓨팅의 한가지로 DNA 컴퓨팅을 다루고자 한다. 바이오분자 컴퓨팅은 DNA, RNA 혹은 단백질과 같은 바이오분자들의 정보 저장 및 처리 특성을 이용한 컴퓨터 기술이다. 이런 바이오분자를 이용한 정보처리에 관한 연구는 이미 1970년대 후반부터 시도되어 왔다.[B 82] 여러 가지 사용 가능한 바이오분자 중에서 현재 가장 큰 실용화 가능성을 보이는 것은 DNA 분자이며, 이것을 이용한 기술이 DNA 컴퓨팅 기술이다.[M 98, Z 02]

이러한 DNA 컴퓨팅 모델 중에서 wDNF(Weighted Disjunctive Normal Form)으로 각각의 DNA를 인코딩해서 계산하려는 시도가 PLM(Probabilistic Library Model)이다.[ZJ 04] 본

논문에서는 이 PLM을 이용해서 숫자 인식에 사용하는 데이터를 분류(Classification)해보고 이 데이터를 적절한 방법으로 오염시킨 뒤, 이를 복구하는 패턴 완성(Pattern Completion) 문제를 접근해보고자 한다.

1.3 논문의 구성

본 논문의 구성은 다음과 같다. 제 2장에서는 PLM에 대해 간단히 소개를 한다. 그리고 3장에서는 본 논문에서 계속해서 사용할 숫자 인식 데이터를 설명하고, 이를 몇 가지 정보과학적인 방법을 이용해서 분석한다. 제 4장에서는 PLM을 이용해서 패턴 분류를 시도한 결과를 보여주며, 제 5장에서는 PLM을 이용해서 패턴 완성을 시도한 결과를 보여준다. 마지막으로 제 6장에서는 앞에서의 실험 결과를 종합하고, 고찰을 기술한다.

2 숫자 인식 문제를 풀기 위한 PLM

2.1 DNA 컴퓨팅

DNA 컴퓨팅을 수행하기 위해서는 다음과 같은 과정이 필요하다.

1. 풀고자 하는 문제에 대한 가능한 해답을 DNA 코드로 표현한다.
2. 이들 코드를 올리고(Oligo) 합성 기술을 사용하여 다량(수 피코 몰)으로 합성한다.
3. 각각의 성분 분자들을 합성기에 넣고 화학반응을 시킴으로써 가능한 모든 해를 생성한다.
4. 생성된 분자들 중에 찾는 해가 포함되어있는지를 검사하여 답을 제시한다.

이러한 원리를 이용하여 해밀톤 경로 문제를 해결할 수 있다는 것이 1994년 Adleman에 의해 처음 실험적으로 증명되었다.[A 94]

2.2 PLM의 학습 방법

PLM을 이용한 학습 과정은 다음과 같다.[ZJ 04]

1. 실험적인 분포 P(X,Y)를 표현하는 라이브러리 L이 있다.
2. 입력 패턴 x가 주어졌을 때,
3. 다음의 규칙에 따라 L을 이용해서 x를 분류한다.
   3-1. x와 매칭되는 모든 분자를 M으로 뽑아낸다.
   3-2. M으로부터 다음의 클래스로 분자들을 구분한다.
      - 라벨 $Y=y_1=1$에 해당하는 분자는 M^1으로 넣는다.
      - 라벨 $Y=y_0=0$에 해당하는 분자는 M^0으로 넣는다.
   3-3. $y^* = \arg\max_{Y\in\{0,1\}} |M^Y| / |M|$를 계산한다.

[그림 1]은 이러한 학습 과정을 그림으로 나타낸 것이다.

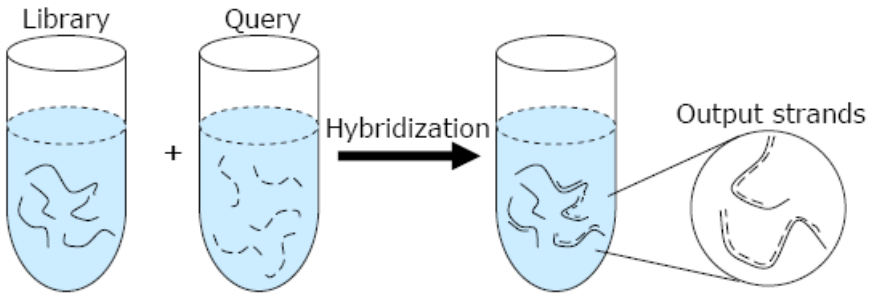


그림 1. PLM을 이용한 학습 과정

2.3 학습 데이터

2.3.1 데이터의 구성

본 논문에서는 PLM을 이용한 패턴 분류와 패턴 완성이라는 학습 문제에 접근하기 위해서 숫자 인식 데이터를 사용했다. 이 숫자 인식 데이터는 각각의 숫자를 8 * 8의 배열 형태로 표현했으며, 이러한 총 64개의 비트는 0 혹은 1의 값을 갖도록 되어있다. 학습 데이터는 총 3823개의 예제를 가지고 있으며, 테스트 데이터는 1797개의 예제를 가지고 있다. 그리고 각각의 데이터들은 0에서 9 사이의 클래스를 갖는다. [표 1]은 본 논문에서 사용한 학습 데이터와 테스트 데이터의 클래스 별 예제 수를 정리한 것이다.

표 1. 학습 데이터와 테스트 데이터의 예제 수

클래스	0	1	2	3	4	5	6	7	8	9
학습	376	389	380	389	387	376	377	387	380	382
테스트	178	182	177	183	181	182	181	179	174	180

2.3.2 엔트로피를 이용한 데이터 분석

엔트로피는 앙상블 X에 대해 다음과 같이 정의할 수 있다.

(1) $$H(X) \equiv \sum_{x \in A_x} P(x) \log \frac{1}{P(x)}$$

이를 문제의 숫자 인식 데이터에 적용하자면, X를 각각의 이미지 데이터에 대한 앙상블이라고 정의하고, Y를 0에서 9까지의 클래스에 대한 앙상블로 정의할 수 있다. 그러면 다음의 식들을 이용해서, 앙상블 X와 Y에 대해, Conditional Entropy, Marginal Entropy, Mutual Information등을 계산할 수 있다.

(2) $H(X \mid y = b_k) \equiv \sum_{x \in A_x} P(x \mid y = b_k) \log \frac{1}{P(x \mid y = b_k)}$

(3) $H(X \mid Y) \equiv \sum_{y = A_y} P(y) [\sum_{x \in A_x} P(x \mid y) \log \frac{1}{P(x \mid y)}]$

(4) $H(X) \equiv \sum_{x \in A_x} P(x) \log \frac{1}{P(x)}$

(5) $I(X;Y) \equiv H(X) - H(X \mid Y)$

위의 식을 주어진 문자 인식의 학습 데이터와 테스트 데이터 각각에 대해서 0~9 사이의 클래스가 주어졌을 때, 각각의 Conditional Entropy를 계산할 수 있다. [표 2]는 이를 정리한 것이다.

표 2. 숫자 별 데이터의 Conditional Entropy

숫자	학습 데이터	테스트 데이터
0	9.66	9.81
1	12.71	13.34
2	13.13	13.64
3	12.43	13.03
4	15.16	13.52
5	14.03	14.03
6	10.77	10.18
7	12.74	13.26
8	13.23	13.41
9	14.77	14.4

[표 2]에서 확인할 수 있듯이, 보통의 경우에는 학습 데이터와 테스트 데이터의 엔트로피가 비슷한 것을 확인할 수 있다. 단, 차이가 많이 나는 숫자는 숫자 4인데, 이 경우에는 아마도 PLM으로 학습을 할 경우에, 숫자 인식의 정확도가 다른 숫자에 비해 떨어질 것으로 예상할 수 있다.

그리고 각 숫자 별로 비교를 해보면, 숫자 0과 6의 엔트로피가 가장 낮은데 이는 그만큼 불확실성이 낮다는 것을 의미하고, 다른 숫자에 비해 인식에 성공할 확률이 높다는 것을 예상할 수 있다. 물론 이 엔트로피 값은 비트맵의 각 비트 별로 계산한 것이기 때문에, 이미지로 주어진 숫자들의 위치를 어떻게 맞췄느냐에 따라 그 값이 매우 달라질 수 있지만, 문제에 주어진 각 숫자의 비트맵에서 숫자가 위치하는 부분이 비슷하다고 가정했을 때, 엔트로

피가 낮은 숫자들이 인식하기에 더 용이할 것으로 생각할 수 있다.

[그림 2]는 [표 2]을 그래프로 정리한 것이다. 숫자 0과 6의 엔트로피가 다른 숫자에 비해 낮고, 숫자 4는 학습 데이터와 테스트 데이터의 엔트로피 차이가 크다는 것을 확실하게 확인할 수 있다.

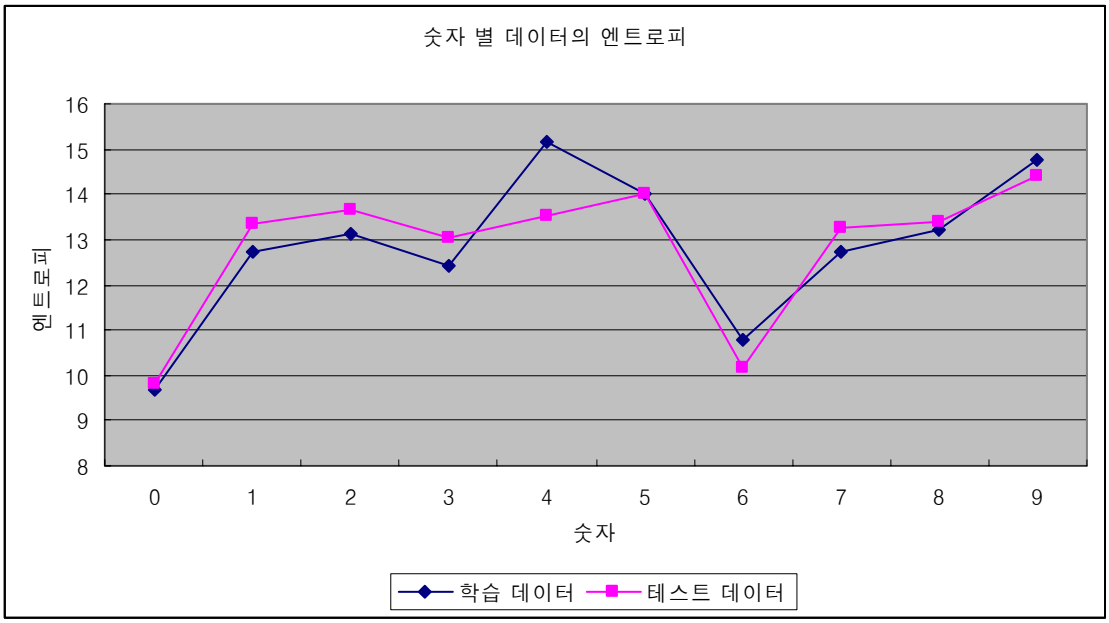


그림 2. 숫자 별 데이터의 엔트로피

그리고 식 (3)와 식(4)를 이용해서 앙상블 X와 Y에 대해 Conditional Entropy, Marginal Entropy, Mutual Information을 계산할 수 있는데, 이를 정리한 것이 [표 3]이다.

표 3. Conditional Entropy, Marginal Entropy, Mutual Information

	학습 데이터	테스트 데이터
H(X|Y)	12.87	12.86
H(X)	18.21	18.29
H(Y)	3.32	3.32
I(X;Y)	5.34	5.43

[표 3]에서 볼 수 있듯이, Conditional Entropy와 Marginal Entropy 모두 학습 데이터와 테스트 데이터에 대해 그 값이 크게 차이가 나지 않는 것을 확인할 수 있다. 따라서 이 부분만 보자면, 주어진 학습 데이터로 학습하고 테스트 데이터로 테스트를 수행하기에 엔트로피 상으로는 일단 큰 무리가 없다고 할 수 있다. 만약 학습 데이터와 테스트 데이터에 대해

Conditional Entropy와 Marginal Entropy가 크게 차이가 났다면 각 데이터들을 샘플링할 때, 이 부분에 대한 고려를 추가해야 할 것으로 보인다.

2.3.3 Kullback-Leibler Divergence를 이용한 분석

두 개의 서로 다른 확률 분포 P(x)와 Q(x)에 대해, 다음 식을 이용해서 Relative Entropy 혹은 Kullback-Leibler Divergence를 계산할 수 있다.

(6) $$D_{DK}(P \| Q) = \sum_{x} P(x) \log \frac{P(x)}{Q(x)}$$

클래스 별로 여러 개의 학습 데이터가 존재하고, 따라서 각 클래스의 학습 데이터의 64개 비트에 대해 서로 다른 확률 분포를 갖는다. 따라서 이 확률 분포들을 이용해서, 학습 데이터의 클래스 간에 KL Divergence를 각각 계산해 볼 수 있다.

문제는 $D_{KL}(P||Q)$와 $D_{KL}(Q||P)$가 다르다는 점인데, 달리 얘기하면 클래스 0에 대한 클래스 1의 KL Divergence와 클래스 1에 대한 클래스 0의 KL Divergence가 다를 수 있다는 뜻이다. 따라서 두 클래스 사이의 Relative Entropy를 각각 구해서 두 값의 평균을 계산하는 방식을 적용했다.

[표 4]는 이렇게 해서 계산한, 클래스 0에서 9 사이의 두 클래스 간의 KL Divergence를 정리한 것이다.

표 4. 클래스 간의 KL Divergence

클래스	0	1	2	3	4	5	6	7	8	9
0	0	4.06	4.50	4.03	4.17	4.10	3.26	4.64	2.38	2.84
1		0	2.67	3.28	2.02	3.42	3.51	2.96	1.13	3.20
2			0	2.42	5.71	3.49	4.01	3.08	2.26	3.80
3				0	6.67	3.29	5.92	3.06	2.34	1.56
4					0	4.49	3.13	4.36	2.92	5.30
5						0	3.49	3.21	2.47	2.83
6							0	4.45	2.83	5.01
7								0	2.19	3.44
8									0	2.27
9										0

KL Divergence가 가장 낮은 것은 1과 8 사이의 1.13, 3과 9 사이의 1.56이다. 가장 높은 것은 3과 4 사이의 6.67과 2와 4 사이의 5.71이다. KL Divergence가 낮다는 것은 그만큼 두 클래스의 비트 분포가 비슷하다는 것을 의미하며, KL Divergence가 높다는 것은 두 클래스의 비트 분포가 다르다는 것을 의미한다.

3 실험 및 평가

3.1 패턴 분류 문제

본 논문에서는 하나의 DNA에 비트맵의 비트를 인코딩하는 숫자는 모든 인코딩 가능한 공간을 한 번에 계산할 수 있는 1과 2에 대해 먼저 실험을 수행하고, 3 이상에 대해서는 샘플링을 부분적으로 하는 방법을 통해 실험을 수행하였다. 그리고 서브 샘플링의 경우에도 여러 비트를 완전히 랜덤하게 선택하는 방법과, 각 비트가 연속적으로 분포하도록 램덤하게 샘플링하는 두 가지 방법을 적용했다.

3.1.1 학습 진행에 따른 인식률 변화

먼저 학습을 진행하면서 학습 데이터와 테스트 데이터에 대해 인식률이 어떠한 변화를 보이는지 알아보기 위해서, 학습 데이터 중에서 2개의 데이터씩 학습을 할 때마다, 학습 데이터와 테스트 데이터에 대해 인식률이 얼마나 되는지를 테스트했다. [그림 3]은 이를 정리한 것이다.

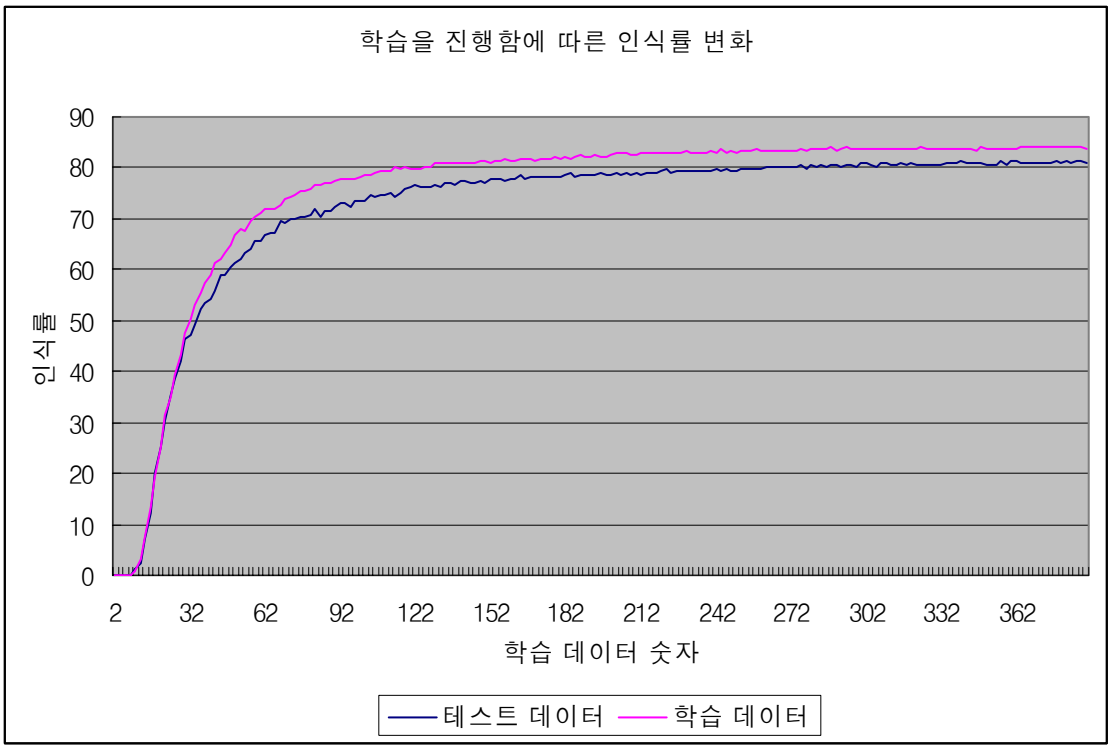


그림 3. 학습을 진행함에 따른 인식률 변화

실행 조건은 1비트 인코딩, 각 DNA별 초기 개체 수 1,000,000, 업데이트 비율 10^{-6}, Threshold는 10^{-7}, 반복 횟수는 1이다.

두 가지 데이터 모두 학습을 진행함에 따라 최종 인식률인 83.78%, 81.07%로 서서히 다가가는 것을 확인할 수 있다. 또한 학습 데이터에 대해 더 좋은 인식률을 보인다는 것도 확인할 수 있다. 흥미로운 현상은 그래프가 매끈하게 올라가지 않고 약간의 흔들림을 보이는 것인데, 이는 특정 데이터에 잘 들어맞는 DNA들이 다른 데이터를 학습하는 과정에서 희석(Dilution) 과정 중에 조금씩 사라지는 것 때문에 생기는 현상으로 이해할 수 있다. 따라서 이것에 의한 효과를 최소화 시키고, 인식률을 높이기 위해서는 학습을 진행할 때, 반복 횟수, 업데이트 비율 그리고 Threshold 값을 조절하기 위해 튜닝을 해야 한다는 예상을 할 수 있다.

3.1.2 업데이트 비율에 따른 학습의 변화

업데이트 비율에 따라 학습의 정도가 변화할 수 있다. 이를 확인해보기 위해서 업데이트 비율을 10^{-1}에서 10^{-7}까지 조절해 가면서 인식률의 변화가 어떻게 되는지 실험했다. 업데이트 정도는 비율로 했는데, 그 이유는 DNA의 초기 개체 수에 독립적으로 설정을 하면서 실험을 진행하도록 하기 위함이다.

실험 조건은 1비트 인코딩, 각 DNA별 초기 개체 수 1,000,000, Threshold는 0, 반복 횟수는 1이다. [표 5]는 이 실험의 결과를 정리한 것이다.

표 5. 업데이트 비율에 따른 인식률 변화

업데이트 비율(Log10)	인식률(%)
-1	17.91
-2	57.76
-3	82.13
-4	82.19
-5	82.52
-6	82.48
-7	82.36

업데이트 비율이 10^{-1}인 경우에는 엄청나게 저조한 인식률을 보이는데, 이는 학습을 진행할 때마다 이전에 학습된 내용이 심하게 희석되서 DNA의 개체수가 급감하기 때문에 나타나는 현상으로 이해할 수 있다. 즉, 업데이트 비율이 너무 크면 라이브러리 내의 DNA가 다양성(Diversity)를 보장하기 힘들어진다고 해석할 수 있다.

10^{-3} 이하의 업데이트 비율에서는 비교적 안정적인 인식률을 보이는데, 따라서 이후 실험에서는 업데이트 비율을 10^{-5}에서 10^{-7} 사이의 값을 사용하기로 했다.

3.1.3 Threshold 값에 따른 학습의 변화

주어진 테스트 데이터에 대해 테스트를 수행할 때, 원하는 클래스의 값이 나왔는지 여부를 확인하는 방법을 여러 가지로 할 수 있는데, 가장 단순한 방법이 Positive한 DNA의 개체 수가 Negative한 DNA의 개체 수보다 1개라도 많으면 참이라고 결론짓는 방법이다. 즉, Majority Voting을 하는 것인데, 실험 결과 이는 그다지 좋지 않은 방법임을 알게 되었다.

[그림 4]는 실험 조건이 1비트 인코딩, 각 DNA별 초기 개체 수 1,000,000, 업데이트 비율은 10^{-7}, Threshold는 0, 반복 횟수는 1인 경우에 학습을 진행해 나감에 따라 인식률의 변화를 정리한 것이다.

흥미로운 현상은 2개의 데이터만 랜덤하게 뽑아서 학습을 진행했음에도 불구하고, 무려 65.83%의 놀라운 인식률을 보였다. 문제는 그래프에서 볼 수 있듯이, 학습이 매우 불안정하게 진행이 되며, 심지어 끝 부분에서는 인식률이 조금씩 덜어지는 경향도 확인할 수 있다.

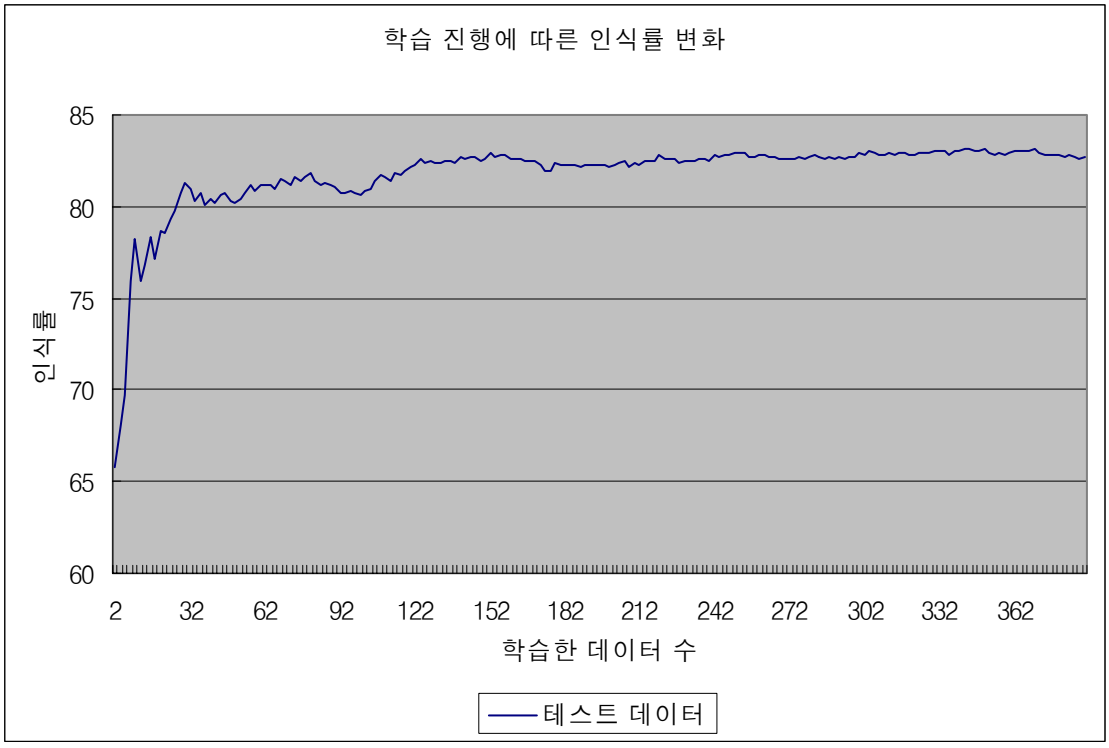


그림 4. Threshold가 0일때 학습

따라서 Threshold를 0으로 하는 Majority Voting을 하지 않고, Threshold를 전체 DNA 종류 수에 비례하는 값으로 설정할 수 있도록 했다.

더 심각한 문제는 Threshold를 0으로 했을 때보다, 적절한 Threshold 값을 주고 테스트 데이터에 대해 반복 학습하는 경우에 더 좋은 성능을 보인다는 점이다. 이 부분은 이어서 더 자세히 살펴보도록 하겠다.

3.1.4 학습 반복에 따른 인식률의 변화

업데이트 비율을 작게 하고, Threshold 값을 적절히 설정한 뒤, 학습을 반복적으로 수행하면 속도는 느리지만, 안정적으로 Local Minima에 도달하게 할 수 있다. 이러한 아이디어를 바탕으로 Threshold 값을 적절히 설정하고, 학습을 반복적으로 수행하면서 인식률이 어떻게 변하는지 테스트를 수행했다.

다음은 실험 조건을 1비트 인코딩, 각 DNA별 초기 개체 수 1,000,000, 업데이트 비율은 10^{-6}, Threshold는 10^{-6}인 경우에 학습 횟수를 늘려가면서 인식률의 변화를 정리한 것이다.

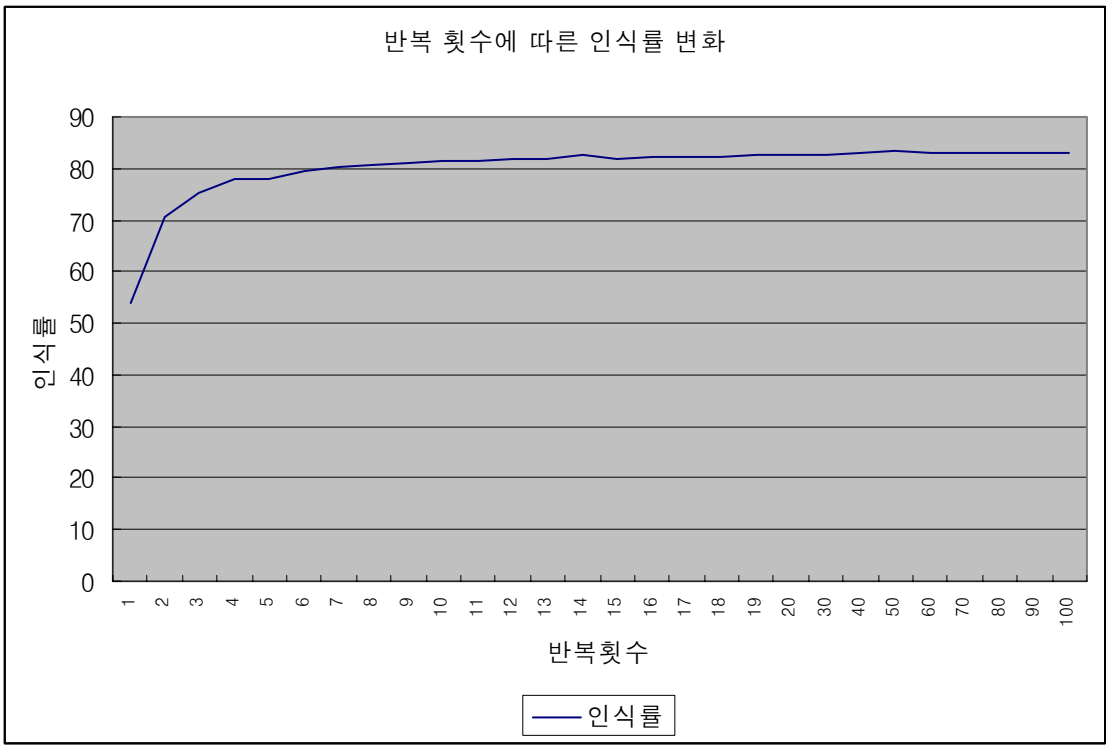


그림 5. 반복 횟수에 따른 인식률 변화

반복 횟수를 늘려감에 따라 처음에는 53.86%였던 인식률이 점자 향상되어, 80회 반복했을 경우에는 83.19%까지 올라간 것을 확인할 수 있다. 눈여겨 볼 점은, Threshold를 0으로 한 경우에는 82%대의 인식률을 보인 반면, Threshold를 주고, 반복 학습을 시킨 경우에는 인식률이 83% 이상 보였다는 점이다.

3.1.5 1 비트 인코딩 시

우선 하나의 DNA가 숫자 이미지 비트맵에서 하나의 비트씩 인코딩을 해서 숫자 인식을 수행했다. 이렇게 인코딩을 하고, 주어진 학습 데이터와 테스트 데이터를 이용해서 숫자 인식을 한 결과는 [표 6]과 같다. 실험 조건은 1비트 인코딩, 각 DNA별 초기 개체 수 1,000,000, 업데이트 비율은 10^{-6}, Threshold는 10^{-6}인 경우에 학습 횟수 80회 이다.

표 6. 비트 1씩 인코딩한 경우

숫자	성공률
0	98.31
1	76.92
2	86.44
3	87.43
4	78.45
5	91.20
6	92.81
7	92.73
8	61.49
9	65.55
평균	83.19

[표 6]에서 볼 수 있듯이, 평균 인식 성공률은 83.19%이고, 인식 가장 잘된 숫자는 0과 3, 6, 7이고, 인식이 가장 안된 숫자는 8과 9였다. 또한 4도 비교적 낮은 인식률을 보였다. 0과 6이 인식이 잘 된 이유는 아마도 앞에서 살펴본 것처럼 0과 6에 해당하는 비트맵 정보의 엔트로피가 낮았기 때문으로 추측할 수 있다. 마찬가지로 엔트로피가 높았던 8과 9 그리고 학습 데이터와 테스트 데이터의 엔트로피 차이가 컸던 4도 낮은 인식률을 보인 것을 알 수 있다.

3.1.6 2 비트 인코딩 시

[표 7]은 우선 하나의 DNA가 숫자 이미지 비트맵에서 두 개의 비트씩 인코딩을 해서 숫자

인식을 수행한 결과이다. 실험 조건은 2비트 인코딩, 각 DNA별 초기 개체 수 1,000,000, 업데이트 비율은 10^{-6}, Threshold는 10^{-7}, 학습 횟수는 30회이다.

표 7. 2 비트씩 인코딩한 경우

숫자	성공률(%)
0	98.57
1	77.73
2	87.26
3	87.69
4	77.81
5	90.91
6	93.63
7	94.11
8	64.05
9	66.36
평균	84.06

앞의 결과와 비슷하게, 엔트로피가 낮은 0과 6 등이 인식률이 높고, 엔트로피가 높거나 학습 데이터와 테스트 데이터의 엔트로피 차이가 큰 4, 8, 9가 인식률이 낮았다. 하지만 비트 1씩 인코딩 한 경우보다 2씩 인코딩한 경우가 평균 인식률이 84.06%로 0.87% 더 높았다.

3.1.7 학습 데이터 샘플링 방법

[표 8]에서 확인할 수 있듯이, 숫자 인식을 위해 주어진 데이터들은 클래스에 따라 데이터의 개수가 조금씩 다르다. 이 부분은 약간 문제가 될 수 있는데, 왜냐하면 학습 데이터가 많은 클래스의 경우에는 많은 학습 데이터로 인해 그에 특정한 DNA들의 개체 수를 늘릴 수 있지만, 반면 나머지 9가지 클래스에 대해 학습된 DNA의 개체 수들이 상대적으로 감소하기 때문에 전체적인 성능은 저하될 수 있다.

표 8. 클래스 별 학습 데이터 개수

	0	1	2	3	4	5	6	7	8	9	합계
개수	376	389	380	389	387	376	377	387	380	382	3823

따라서 학습할 때, 데이터 개수가 가장 많은 389개에 대해 랜덤하게 클래스 별로 389개씩 학습 데이터를 뽑아서 학습했다. 즉, 학습 데이터가 가장 적은 클래스 0이 376개의 데이터를 가지고 있지만, 학습은 랜덤하게 389개를 뽑아서 진행했다. 이렇게 함으로써 인식률을

향상시킬 수 있었다.

[표 9]은 학습 데이터 샘플링 방법을 변경하기 전/후에 따라 1 비트 인코딩과 2 비트 인코딩에서 인식률이 어떻게 향상되었는지를 정리한 것이다. [표 9]에서 확인할 수 있듯이, 상당히 많이 인식률이 향상되었음을 알 수 있다.

표 9. 학습 데이터 샘플링 방법 변경 전/후 인식률 변화

	1 비트 인코딩	2 비트 인코딩
수정 전	78.57	82.35
수정 후	83.19	84.06

참고할 것은 수정 전의 인식률은 Threshold를 0으로 하고 반복 학습을 하지 않은 실험 조건도 포함되어있다는 점이다. 따라서 인식률의 향상이 다른 실험 조건의 변경에 의한 것으로 생각할 수도 있지만, 앞에서 살펴본 것처럼 Threshold의 수정과 반복 학습을 통한 인식률의 향상은 합쳐서 1~3%정도라는 점을 감안하면, 샘플링 방법의 변경을 통해서 인식률이 어느 정도 향상되었다는 결론을 조심스럽게 내릴 수 있다.

3.1.8 3 비트 이상 인코딩 시

비트 이상의 인코딩에서는 각 DNA로 표현할 수 있는 공간을 모두 표현하면 실질적인 시간 안에 계산을 완료할 수 없기 때문에, 이 중에서 일부분의 DNA 종류만을 샘플링해서 실험을 진행했다. 서브 샘플링하는 방법은 가장 단순하게 랜덤 샘플링을 생각할 수 있다.

[표 10]는 인코딩된 DNA의 종류를 랜덤하게 선택하고, 업데이트 비율을 10^{-6}, Threshold를10^{-7}로 고정했을 때, 인코딩 비트 수를 3에서 6까지 그리고 각각의 비트 수에 대해 전체 DNA 종류 수를 100,000, 200,000, 300,000, 500,000, 700,000, 1,000,000으로 늘려가면서 실험을 한 결과를 정리한 것이다.

표 10. 3 비트 이상 인코딩 시, 전체 DNA 종류 수에 따른 인식률 변화

Pop.	Bit 3	Bit 4	Bit 5	Bit 6
100000	83.63	84.36	83.30	84.24
200000	82.97	84.41	84.25	84.37
300000	83.41	83.75	85.08	85.84
500000	83.13	83.69	84.91	85.58
700000	83.24	84.02	84.86	85.44
1000000	83.36	84.52	84.47	84.72

평균	83.29	84.13	84.48	85.03

[표 10]에서 확인할 수 있듯이, 각 비트 수에 대해 DNA 종류 수를 최대 10배까지 늘렸음에도 불구하고 인식률에는 큰 차이가 없는 것을 확인할 수 있다. 이는 랜덤하게 선택된 DNA 중에서 인식률을 결정하는데 중요한 영향을 미치는 것들이 존재한다는 것을 암시해준다. 달리 이야기하면, 랜덤한 방식으로 DNA를 인코딩해서 샘플링하는 것보다는, 선의 진행 방향처럼 문자 인식에 더 많은 정보를 줄 수 있는 정보를 고려해서 샘플링하는 것이 보다 좋은 결과를 줄 것이라는 예측을 가능하게 한다.

또 한가지 관찰할 수 있는 사항은, 각 비트 수에 대해 전체 개체 수에 의한 인식률의 변화는 저조하지만, 비트 수가 늘어감에 따라 평균 인식률이 점점 증가하는 것을 확인할 수 있다. 이는 DNA 인코딩에 들어간 비트 수가 늘어갈수록, 비트맵 상의 각 비트들이 가지고 있을 수 있는 상관 관계에 대한 정보를 포함할 수 있는 가능성이 늘어가는 것으로 해석할 수 있다.

표 11. 랜덤 샘플링, 업데이트 비율 10^{-6}인 경우, 비트 수에 따른 인식률 변화

Bit	인식률(%)
3	83.46
4	84.24
5	84.48
6	85.47
7	86.16
8	86.64
9	86.69

[표 11]은 랜덤 샘플링, 업데이트 비율 10^{-6}, Threshold는 10^{-7} 그리고 DNA 종류 수를 2,000,000으로 한 경우에, 비트 수에 따른 인식률의 변화를 정리한 것이다. 앞에서 언급했던 경향을 다시 한 번 확인할 수 있는 데이터이다. [표 10]에서 실험했던 최대 개체수인 1,000,000보다 2배 높은 DNA 종류 수를 적용한 이유는, 실험 시간이 많이 걸리더라도 최대한 다양한 DNA 종류를 라이브러리에 넣음으로써 인식률을 높이고, DNA의 다양성이 부족함으로 인해 잘못된 해석을 하는 것을 막기 위함이었다.

3.1.9 3 비트 이상 인코딩 시, 랜덤 선택 변경

[표 10]에서 랜덤 샘플링은 그다지 효율적이지 않은 조건이라는 사실을 깨달았다. 따라서 N개의 비트를 샘플링해야 할 때, 처음 0번째 비트는 랜덤하게 샘플링하고 나머지 N-1개의

비트는 이전 비트에서 북서/북/북동/서/동/남서/남/남동으로 연속적으로 이동하도록 랜덤하게 샘플링하는 방법으로 약간 변경했다.

[표 12]은 이러한 이웃 랜덤 선택으로 서브 샘플링 방법을 정하고, 업데이트 비율을 10^{-6}, Threshold를 10^{-7}, 그리고 반복 횟수를 20회로 고정했을 때, 인코딩 비트 수를 3에서 6까지 그리고 각각의 비트 수에 대해 전체 DNA 종류 수를 100,000, 200,000, 300,000, 500,000, 700,000, 1,000,000으로 늘려가면서 실험을 한 결과를 정리한 것이다.

표 12. 이웃 랜덤 서브 샘플링 시 전체 DNA 종류 수에 따른 인식률 변화

Pop.	Bit 3	Bit 4	Bit 5	Bit 6
100000	84.36	85.86	85.70	86.24
200000	84.67	86.41	85.76	86.37
300000	84.41	86.65	86.08	87.84
500000	84.72	86.80	86.91	87.58
700000	84.93	86.72	87.26	87.44
1000000	85.17	86.92	87.47	86.72
평균	84.71	86.56	86.53	87.03

[표 12]에서 확인할 수 있는 사실은 일단 비트 수가 늘어갈수록 평균 인식률이 조금씩 증가하는 경향을 보인다는 점이다. 이는 이미 [표 11]에서 확인한 바가 있다. 또 한가지 짚고 넘어가야 할 중요한 사실은 [표 11]과는 달리 이웃 랜덤 서브 샘플링 방법을 취했을 때, 비트 수가 정해진 경우 전체 DNA 종류 수를 늘려감에 따라 인식률이 조금씩 나아지고 있다는 점이다. 이는 각 비트들이 서로 독립이 아니며, 선의 진행 방향에 따라 연속적으로 점들이 존재하는 것을 고려해서 서브 샘플링함으로써 숫자 인식에 기여할 수 있는 DNA들을 더 많이 라이브러리에 저장할 수 있었기 때문으로 해석할 수 있다.

3.1.10 최대 인식률을 보인 조건

[표 11]과 [표 12]의 경향성을 바탕에 두고, 90% 이상의 인식률을 보일 수 있는 조건을 찾기 위한 실험을 수행했다. 일단 인코딩 비트 수를 늘려주는 것이, 숫자 인식률에 많은 기여를 하는 것 같다는 판단에 근거해서, 인코딩 비트 수를 10과 20에 대해 학습을 진행했다.

[표 13]는 이웃 랜덤 선택으로 서브 샘플링 방법을 정하고, 업데이트 비율을 10^{-6}, Threshold를 10^{-7}, 그리고 반복 횟수를 10회로 고정했을 때, 인코딩 비트 수를 3에서 6까지 그리고 각각의 비트 수에 대해 전체 DNA 종류 수를 100,000, 200,000, 300,000, 400,000, 500,000, 1,000,000으로 늘려가면서 실험을 한 결과를 정리한 것이다. X라고 표

시된 조건은 실험을 하지 않은 조건이다.

표 13. 10과 20 비트 인코딩 시 인식률 변화

Pop.	Bit 10	Bit 20
100000	87.97	X
200000	88.75	X
300000	87.20	89.61
400000	88.36	89.54
500000	88.73	89.84
1000000	X	90.23

20 비트로 인코딩하는 경우에 90%를 넘는 조건을 찾았기 때문에, 이번에는 20 비트 인코딩, DNA 종류 수는 10,000,000, 업데이트 비율을 10^{-6}, Threshold를 10^{-7}으로 실험을 했으며, 이때 평균 인식률은 93.53%를 보였으며, [표 14]은 각 클래스 별 인식률을 정리한 것이다.

표 14. 20 비트 인코딩, DNA 종류 수 10,000,000시 클래스 별 인식률

숫자	성공률(%)
0	98.91
1	94.00
2	93.82
3	91.85
4	88.44
5	94.55
6	96.73
7	95.01
8	93.12
9	88.93
평균	93.53

[표 14]에서도 볼 수 있듯이, 여전히 숫자 4와 9의 인식률이 다른 숫자들에 비해 떨어지고, 0과 6, 7의 인식률은 상대적으로 높은 것을 확인할 수 있다. 단, 숫자 8의 인식률이 다른 조건에서는 약간 낮게 나왔는데 이번 조건에서는 상대적으로 높게 나온 것도 특이한 사항이다.

3.2 패턴 완성 문제

패턴 완성 문제에서도 패턴 분류 문제에서 사용한 동일한 데이터를 사용한다. 본 논문에서 패턴 완성 문제에 접근하면서 염두에 둔 것은 [그림 2]에서 정리한 숫자 인식의 인식률과 엔트로피와의 상관 관계다. 즉, 엔트로피가 높거나, 학습 데이터와 테스트 데이터의 엔트로피 차이가 큰 경우에는 인식률이 낮아지는 현상을 이미 앞에서 확인했다. 따라서 패턴 완성에서는 Correlation PLM을 이용해서 숫자 비트맵 정보를 학습하고, 비트맵 정보를 랜덤하게 오염시킨 후, 이를 복구하는 실험을 하게 되는데, 이때 엔트로피가 어떤 식으로 변화하는지 살펴보고 분석하였다.

3.2.1 학습 조건

본 연구를 수행하기 위해서는 먼저 Correlational PLM을 이용해서 숫자 비트맵 정보를 학습하는 조건이 필요하고, 원래 숫자 비트맵 정보에서 랜덤하게 오염된 숫자 비트맵 정보를 만드는 방법이 필요하며, 마지막으로 학습된 Correlational PLM을 이용해서 오염된 숫자 비트맵 정보를 인식이 가능한 형태로 복구하는 방법과 이렇게 해서 복구된 비트맵 정보가 오염되기 전의 비트맵 정보와 얼마나 다른지에 대한 정보가 필요하다. 이 네 가지에 대해 차례대로 살펴보도록 하자.

3.2.1.1 Correlational PLM 학습 조건

Correlational PLM을 효율적으로 학습시키는 방법에 대해서는 앞에서 이미 수행한 바 있다. 시뮬레이션으로 찾아낼 수 있었던 최적의 조건은 다음과 같다.

20 비트 인코딩, 이웃한 랜덤 비트 샘플링, DNA 종류 수는 10,000,000, 업데이트 비율을 10^{-6}, Threshold를 10^{-7}, 반복 횟수는 3으로 시뮬레이션했을 때 평균 인식률은 93.53%를 보였다.

이번에는 학습을 효율적으로 시키는 조건을 찾고 이를 이해하는 것보다, 학습 데이터를 오염시켰을 때 학습률이 어떻게 변화하는지, 그리고 이를 어떻게 Correlational PLM으로 복구시킬 것인지가 관심 사항이므로 주로 다음 조건에서 시뮬레이션을 수행한다.

1, 2 비트 인코딩, 업데이트 비율 10^{-6}, Threshold 10^{-7}, 반복 횟수 1으로 시뮬레이션 조건을 잡았다.

이 조건은 앞에서 살펴보았던 것처럼, [그림 6]과 같이 상당히 안정적으로 학습을 하는 조

건이다.

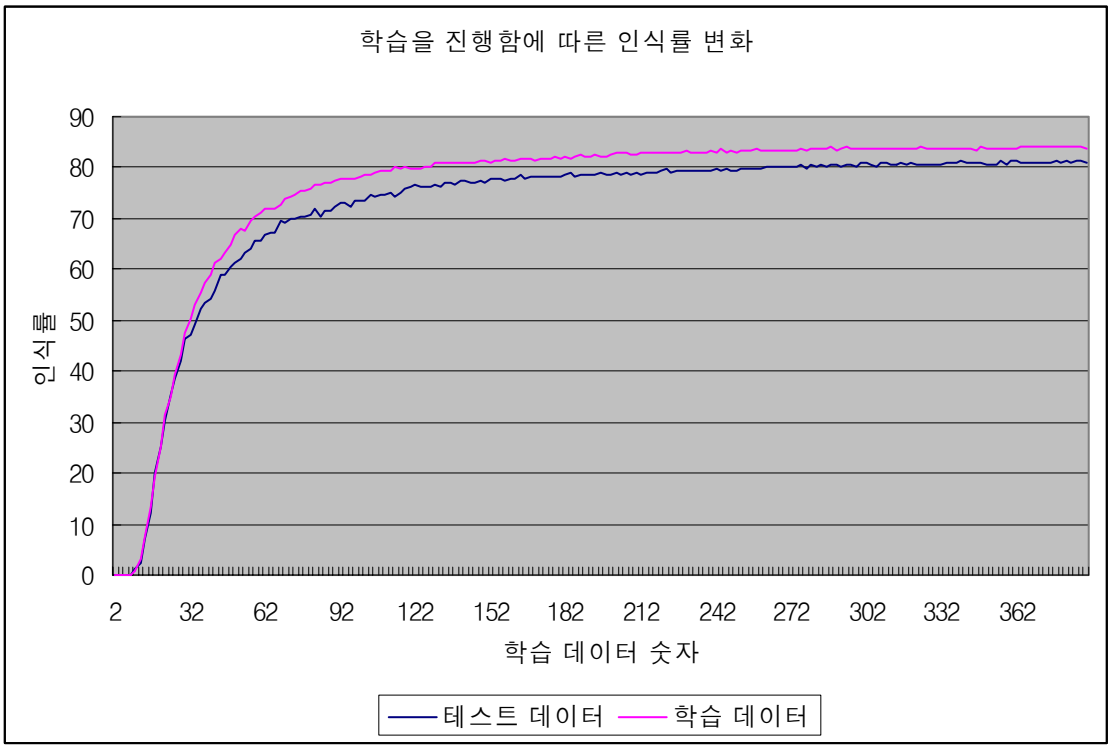


그림 6. 학습을 진행함에 따른 인식률 변화

3.2.2 오염된 테스트 데이터 생성

하나의 숫자 비트맵 정보에는 총 64개(8*8)의 비트 정보가 들어간다. 이를 랜덤하게 오염(Corruption)시키기 위해서, 주어진 횟수만큼 랜덤한 비트의 값을 0에서 1로 혹은 1에서 0으로 바꾸었다. 그리고 오염 횟수는 1에서 64까지 지정해서 시뮬레이션을 수행했는데, 비트를 랜덤하게 선택했기 때문에 오염 횟수가 증가하면서 뒤집어졌던 비트가 다시 원래 값으로 되돌아오는 것도 허용했다.

3.2.3 오염된 데이터의 복구 방법

이제 Correlational PLM으로 오염된 데이터를 복구하는 방법에 대해 살펴보자. 대략의 알고리즘은 다음과 같다.

- 복구한 숫자 비트맵 데이터 BM_INFO와 복구 시도 횟수 REPAIR_CNT, 학습된 Correlational PLM 라이브러리 LIB가 주어진다.

- 카운트 cnt를 1으로 초기화하고 REPAIR_CNT가 될 때까지 1씩 증가하며 다음을 수행한다.
  - LIB 내에서 랜덤하게 DNA Lib_i를 선택한다.
  - Lib_i의 비트 정보가 BM_INFO의 비트 정보와 일치하는지 체크한다.
  - 그리고 Lib_i와 비트 정보가 동일한 10개 클래스에 대한 DNA의 개수를 얻어온다.
  - BM_INFO에 해당하는 클래스가 다른 클래스의 DNA 개수보다 Threshold 값 이상으로 큰지 비교한다.
  - 만약 크다면 cnt를 1 증가시키고 다음 루프로 넘어간다.
  - 만약 작다면 Lib_i의 비트 인덱스 값들을 뽑아낸다. 그리고 이 비트 인덱스에 해당하는 비트의 값이 0/1이 될 수 있는 모든 조합을 찾아낸다. 만약 비트 인덱스의 개수가 Lib_i_idx_cnt개라면, 총 $2^{Lib_i_idx_cnt}$개가 나온다.
  - 앞에서 구한 DNA들 중에서 BM_INFO의 클래스에 해당하는 DNA의 개수가 다른 클래스의 DNA 개수보다 가장 많은 차이로 우세한 DNA의 비트 조합을 찾아낸다.
  - 그리고 이 DNA의 비트 정보를 이용해서 BM_INFO의 비트 정보를 수정한다.

위의 알고리즘대로 오염된 숫자 비트맵 정보를 복구하며, 만약 1000회의 복구 시도를 하였다면, 랜덤하게 라이브러리내의 DNA가 뽑힐 것이고 이 중에서 일부만이 오염된 숫자 비트맵 정보를 복구하는데 기여하게 된다.

시뮬레이션을 위한 알고리즘은 위와 같지만, 실제 실험으로 구현하기 위해서는 직렬적인 절차를 병렬적인 실험 과정으로 변환해서 수행해야 한다. 일단 REPAIR_CNT의 횟수만큼 다음의 실험을 반복해야 한다.

- 복구하고자 하는 데이터에 대해, 각 비트들과 클래스에 대한 정보를 담고 있는 DNA들을 준비한다. 학습 데이터를 학습 시킬 때 사용하는 DNA와 비슷한 과정을 거쳐 준비할 수 있으며, 클래스는 이미 알고 있으므로 비트에 대한 정보만 인코딩하고 있는 DNA면 충분하다.
- 이 DNA 중에서 한 종류를 비드에 달아서 준비하고, 여기에 학습된 PLM의 DNA들을 Denaturation시킨 뒤 Fitering한다. 그러면 해당 비트 조합에 대해 10가지 종류의 다른 클래스에 대한 DNA들이 학습된 분포에 따라 Filtering된다.
- 비드에서 DNA들을 떼어낸 뒤, 처음에 사용했던 오염된 데이터의 DNA와 각 클래스에 대한 정보만 들어있는 DNA 조각을 같이 넣고 Hybridization 시킨다. 이때 각 클래스에 대한 정보만 들어있는 DNA 조각은 각 클래스별로 서로 다른 형광 물질로 표지를 한다. 현재 실험적으로 사용하는 것은 4가지 종류이지만, 앞으로 더 늘어

날 수 있을 것으로 보인다.

- 각 클래스의 색깔에 대해 형광량을 측정한다. 만약 형광량이 가장 많이 나오는 클래스가 원래의 클래스라면 오염된 데이터의 DNA는 인식률에 기여하는 비트 조합을 인코딩하는 DNA가 된다.
- 만약 그렇지 않다면, 앞의 시뮬레이션 알고리즘처럼 이 비트 조합을 바꿔야 한다. 일단 실험에 사용한 비트 조합을 알고 있으므로, 해당 비트들에 들어갈 수 있는 값인 0/1의 조합으로 표현 가능한 모든 DNA들을 준비한다. 2비트 인코딩이라면 총 4종류가 된다. 이중 1가지는 이미 실험을 했으므로 3가지를 준비한다.
- 이 3가지 DNA에 대해서 2번째에서 4번째까지의 실험을 반복함으로써 인식률에 영향을 미치는 비트 조합을 찾아낼 수 있다.
- 이 정보를 바탕으로 원래 비트맵 데이터의 정보를 수정한다.

3.2.4 오염된 데이터의 복구 성능 평가

비트 서열이 다른 두 개의 숫자 비트맵 정보의 거리를 비교하기 위해서, 가장 간단하게 적용할 수 있는 Hamming Distance를 사용했다. 즉, 64개의 비트에 대해 값이 다른 비트가 있을 때마다 카운트를 증가시켜주는 것이다. 본 연구에서는 기본적으로 이 방법을 적용했으며, 이외에도 주어진 복구 시도 중에서 평균 몇 회 복구를 했는지 그리고 이렇게 복구를 한 숫자 비트맵 데이터의 집합에 대해 엔트로피를 조사하는 아이디어도 도입했다.

3.2.5 시뮬레이션 결과

3.2.5.1 1 비트 인코딩의 경우

1 비트 인코딩을 하고 학습 데이터에 대해 앞에서 언급한 조건으로 학습을 수행한 뒤, 오염 횟수를 증가시키면서 학습 데이터를 오염시키고, 총 128회(클래스를 제외했을 때, 라이브러리 내의 DNA 개체 수)의 복구를 수행했을 때의 인식률과 엔트로피를 측정한 결과는 [표 15]와 같다.

[표 15]에서 corruption은 오염 횟수, training은 학습 데이터에 대한 인식률, corrupted는 오염된 데이터에 대한 인식률, c_entropy는 오염된 데이터의 엔트로피 그리고 repaired는 복구된 데이터에 대한 인식률, r_entropy는 복구된 데이터의 엔트로피를 나타낸다.

표 15. 1 비트 인코딩시, 오염 횟수에 따른 인식률와 엔트로피 변화

corruption	training	corrupted	c_entropy	repaired	r_entropy
1	83.28	81.89	15.69	99.58	10.35

2	83.57	80.74	17.66	99.45	11.65
3	83.70	80.69	19.37	99.60	12.79
4	84.25	79.20	20.80	99.55	13.68
5	83.18	76.45	21.97	99.47	14.63
6	83.46	75.59	23.35	99.47	15.24
7	83.41	74.18	24.04	99.21	16.22
8	83.67	73.13	24.82	99.05	16.71
9	83.75	71.17	25.66	99.11	17.19
10	83.41	68.37	26.37	99.24	17.94
15	82.78	61.52	28.83	98.79	19.53
20	82.89	52.52	30.37	98.82	20.82
25	83.65	44.91	31.33	98.11	22.01
30	83.23	38.52	32.02	97.77	22.72
35	83.83	32.67	32.30	98.01	22.96
40	84.20	29.00	32.44	97.56	23.33
45	83.99	23.46	32.58	96.78	24.16
50	83.54	21.13	32.57	97.14	24.19
55	84.07	18.51	32.53	96.57	24.31
60	83.54	17.13	32.53	96.28	24.57
64	82.99	16.58	32.54	95.60	24.70
average	83.54				

[표 15]에서 확인할 수 있듯이, 학습 데이터에 대한 인식률은 거의 비슷하게 나오며 평균 83.54%의 인식률을 보인다. 중요한 것은 랜덤하게 오염시킨 데이터에 대한 인식률은 오염 횟수를 증가시킬수록 떨어져서 64회 랜덤하게 오염시킨 경우에는 16.58%의 인식률을 보였다.

그리고 128회의 복구를 수행했을 때, 대부분의 경우 97%에서 99%의 인식률을 보일 정도로 복구가 이루어졌다. 흥미로운 점은 오염시키기 전의 학습 데이터와 오염시킨 뒤의 데이터 그리고 복구한 데이터에 대한 엔트로피의 변화이다. [그림 7]는 이를 그림으로 정리한 것이다.

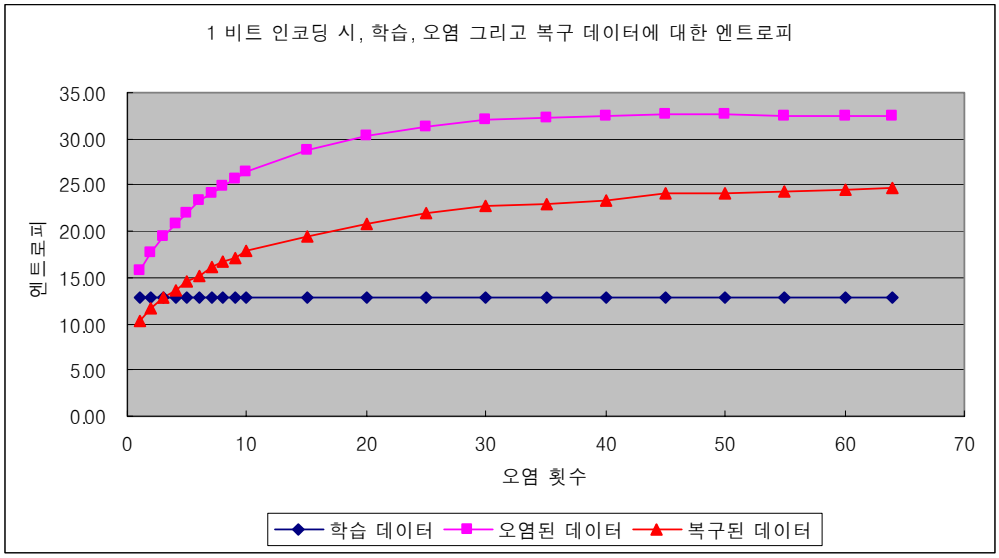


그림 7. 1 비트 인코딩 시, 학습, 오염 그리고 복구 데이터에 대한 엔트로피

[그림 7]에서 가장 먼저 확인할 수 있는 것은 오염된 데이터의 엔트로피보다 복구된 데이터의 엔트로피가 항상 낮다는 점이다. 이는 패턴 분류 문제에서 발견했던 현상으로, 데이터 집합의 엔트로피가 낮을수록 인식률이 높아지는 경향과 일치하는 부분이다.

또한 흥미로운 점은 오염 횟수가 작을 때는 복구된 데이터의 엔트로피가 원래의 학습 데이터의 엔트로피보다 작지만, 오염 횟수가 증가될수록 복구된 데이터의 엔트로피가 더 큼에도 불구하고 더 좋은 인식률을 보인다는 점이다.

이 부분은 모순처럼 보일지도 모르겠으나, 복구된 데이터의 엔트로피를 구성하는 요소를 두 부분으로 나누어서 생각할 수 있다. 즉, 오염된 데이터와 복구된 데이터의 차이에 해당하는 엔트로피 부분은 인식률에 직접적으로 기여하는 요소들에 의해 특정 패턴으로 수렴하면서 낮아진 엔트로피이며, 그 이외에 엔트로피들은 인식률에는 많이 영향을 미치지 않으면서 다양한 패턴의 비트맵 정보를 만들어내는데 기여하는 것으로 해석할 수 있다.

이 부분을 좀더 부각시키기 위해서, 오염된 데이터와 복구된 데이터의 엔트로피 차이를 정리한 것이 [표 16]이고, [그림 8]는 이를 그래프로 나타낸 것이다.

표 16. 오염된 데이터와 복구된 데이터의 엔트로피 차이

Corruption	diff_entropy
1	5.34
2	6.01
4	7.12

6	8.11
8	8.11
10	8.43
20	9.55
30	9.30
40	9.11
50	8.38
60	7.96
64	7.84
average	8.10

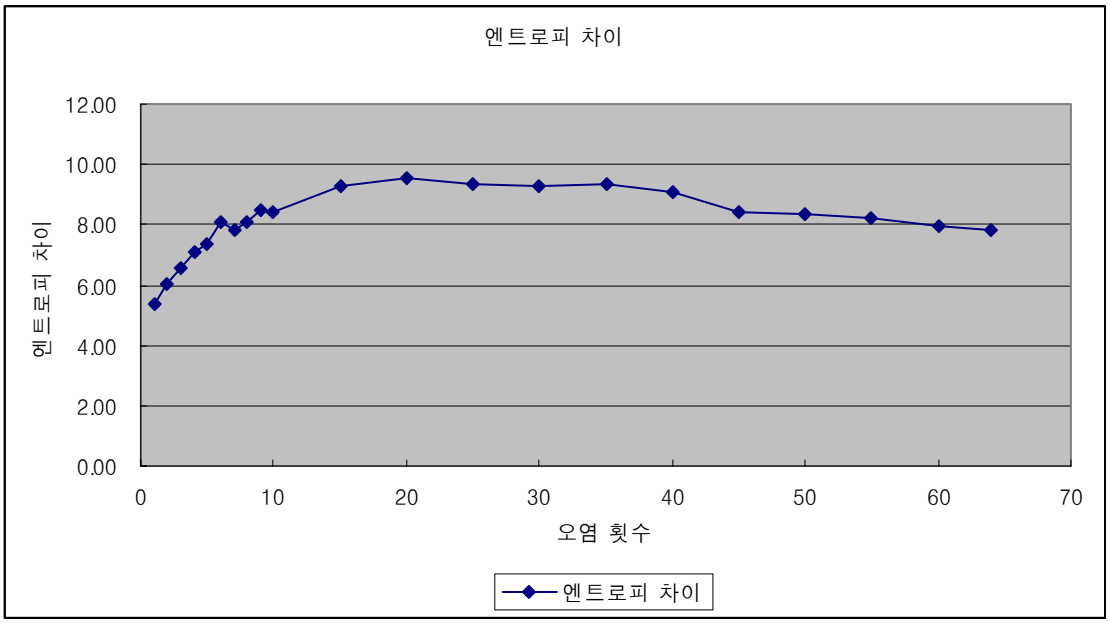


그림 8. 오염된 데이터와 복구된 데이터의 엔트로피 차이

[표 16]에서 확인할 수 있듯이 오염된 데이터와 복구된 데이터의 엔트로피 차이는 대체로 8 부근의 값을 갖는다. 이 말은 오염 과정에서 늘어난 엔트로피 중에는 인식률에 기여를 많이 하는 요소들을 예측 불가능하게 만드는 엔트로피 증가량이 약 8정도이며, 오염 과정에서 늘어난 엔트로피에서 8 정도를 제외한 나머지 엔트로피 증가량은 인식률에 영향을 그다지 미치지 않는 요소들에게 작용한 것으로 생각할 수 있다. 즉, 원래의 숫자 인식 데이터 중에서 인식률에 크게 기여하는 요소가 따로 존재한다는 점을 다시 한 번 확인할 수 있었다.

[그림 8]에서 관찰할 수 있는 경향은 오염 횟수가 증가할수록 복구 과정을 통해 줄어드는 엔트로피의 차이가 점점 커지다가, 오염 횟수가 30회 부근을 넘어서면 약간씩 줄어드는 현

상이다. 이는 랜덤하게 오염시킬 비트를 선택했기 때문에, 총 64개의 비트 중에서 30회 이상의 비트를 골라서 뒤집게 되면, 이 중에는 짝수 번 뒤집힘으로써 원래의 값으로 되돌아오거나, 3 이상 홀수 번 뒤집히는 확률이 증가함으로써 오히려 오염 횟수의 영향을 감소시키는 것으로 해석할 수 있다.

[표 17]은 오염 횟수가 증가함에 따라 총 64회의 복구 시도 중에서 실제 복구에 들어간 복구 횟수와 이로 인해 발생된 원래 데이터와 복구된 데이터의 거리를 정리한 것이다.

표 17. 복구 횟수와 원래 학습 데이터와의 평균 차이

corruption	repair count	distance
1	8.18	8.39
2	8.44	8.86
4	8.84	9.65
6	9.49	10.64
8	9.90	11.46
10	10.12	12.25
20	11.98	15.27
30	13.04	17.62
40	14.13	19.02
50	14.16	20.86
60	14.56	21.51
64	14.73	21.95

오염 횟수가 증가할수록 복구 횟수가 점점 늘어나는데, 이는 랜덤하게 오염시킨 데이터가 Correalation PLM에 의해 학습된 라이브러리가 담고 있는 정보와 점점 더 멀어진다는 것을 의미한다. 즉, 이렇게 때문에 동일한 128회의 복구 시도 중에서 더 많은 복구 횟수가 이루어졌다. 또한 오염 횟수가 늘어남에 따라 원래 데이터와의 거리가 점점 늘어나는 것은 복구 과정을 거치면서 오염되었던 데이터가 원래의 데이터로 돌아가기 보다는, Correlational PLM에 의해 학습된 라이브러리에 들어있는 각 클래스의 공통적인 특징을 좀더 반영할 수 있도록 비트 조합이 달라진 것으로 해석할 수 있다.

3.2.5.2 2 비트 인코딩의 경우

2 비트 인코딩을 하고 학습 데이터에 대해 앞에서 언급한 조건으로 학습을 수행한 뒤, 오염 횟수를 증가시키면서 학습 데이터를 오염시키고, 총 8064회(클래스를 제외했을 때, 라이브러리 내의 DNA 개체 수)의 복구를 수행했을 때의 인식률과 엔트로피를 측정한 결과는 [표

18]와 같다.

[표 18]에서 corruption은 오염 횟수, training은 학습 데이터에 대한 인식률, corrupted는 오염된 데이터에 대한 인식률, c_entropy는 오염된 데이터의 엔트로피 그리고 repaired는 복구된 데이터에 대한 인식률, r_entropy는 복구된 데이터의 엔트로피를 나타낸다.

표 18. 2 비트 인코딩시, 오염 횟수에 따른 인식률와 엔트로피 변화

corruption	training	corrupted	c_entropy	repaired	r_entropy
1	83.57	82.55	15.71	100.00	3.98
5	84.54	76.79	21.98	100.00	4.00
10	84.07	69.00	26.36	100.00	4.01
15	83.78	59.61	28.82	100.00	4.14
20	83.41	51.34	30.40	100.00	4.38
30	83.52	37.30	31.92	100.00	4.13
40	83.25	25.55	32.48	100.00	4.13
50	83.96	20.06	32.64	100.00	4.23
64	84.01	14.83	32.50	100.00	4.26
average	83.79				

여기에서도 [표 15]에서 관찰했던 현상과 비슷한 경향을 확인할 수 있다. 즉, 학습 데이터에 대한 인식률은 오염 횟수에 상관없이 평균 83.79%의 인식률을 보였으며, 오염 횟수가 증가할수록, 오염된 데이터의 인식률은 점점 감소하며 이에 따라 오염된 데이터의 엔트로피도 점점 증가한다. 그리고 1 비트 인코딩에서는 총 128회의 복구 과정을 거친 것과는 달리, 여기에서는 총 8064회의 복구를 수행했는데, 이는 라이브러리 내의 모든 DNA 조합에 대해 복구 시도를 충분히 할 수 있도록 하기 위함이다. 이렇게 한 결과 총 100%의 인식률을 보였으며, 이를 반영하듯 복구된 데이터의 엔트로피가 상당히 낮게 나온 것을 확인할 수 있다.

1 비트 인코딩의 시뮬레이션 결과 대략 8정도의 엔트로피 감소가 100%에 약간 못미치는 정도의 인식률을 내도록 하는데 크게 기여했다는 점을 상기해보면, 복구된 데이터의 엔트로피가 약 4정도로 낮게 나온 것은, 복구 과정이 너무 많이 적용됨으로써 Correlational PLM의 라이브러리에 학습된 요소에 심하게 비트맵 데이터가 변형되었을 것이라는 예측을 가능하게 한다. 이 부분에서 착안하여, 뒷부분의 시뮬레이션에서는 엔트로피를 심하게 감소시키지 않는 방향에서 복구 시도를 멈추는 방법을 적용하였다.

[표 19]는 1비트 인코딩과 2비트 인코딩에 대해 오염된 데이터와 복구된 데이터의 엔트로피 차이를 정리한 것이다. [표 19]에서 corruption은 오염 횟수, diff_entropy2는 2 비트 인

코딩 시 오염된 데이터와 복구된 데이터의 엔트로피 차이 그리고 diff_entropy1은 1 비트 인코딩 시의 해당 데이터이다.

표 19. 1, 2 비트 인코딩 시, 오염된 데이터와 복구된 데이터의 엔트로피 차이

corruption	diff_entropy2	diff_entropy1
1	11.73	5.34
5	17.98	7.34
10	22.35	8.47
15	24.68	9.30
20	26.02	9.55
30	27.79	9.30
40	28.35	9.11
50	28.41	8.38
64	28.24	7.84

일단 2 비트 인코딩 시 더 많은 양의 엔트로피가 줄어든 것을 확인할 수 있다. 이 부분은 2 비트 인코딩의 시뮬레이션에서는 총 8046회의 복구 시도를 거쳤기 때문에, 128회 복구 시도를 한 1 비트 인코딩의 엔트로피 차이보다는 훨씬 더 큰 것을 알 수 있다. 따라서, 복구 시도를 더 많이 할수록 더 많은 양의 엔트로피가 줄어드는 것도 짐작할 수 있다. 그리고 이 부분은 복구 횟수를 적절이 조절하고자 할 때의 비교 데이터로 사용할 수도 있다.

그리고 인식률 100%를 보였다는 것은 완전하게 Correlational PLM의 정보에 맞게 숫자 데이터가 복구 및 변형되었음을 의미하는데, 이때에는 그 복구되는 양이 오염 횟수가 증가할수록 대략 28 부근의 값으로 수렴하는 것을 알 수 있다. 그리고 이 부분은 거꾸로 생각하면 랜덤하게 64회 오염을 시도했을 때 늘어날 수 있는 엔트로피의 증가량의 한계치가 있다는 것으로 해석할 수도 있겠다. 이를 반영하듯, 대부분의 실험에서 오염 횟수를 증가시키면 엔트로피의 값이 특정 값으로 수렴하는 현상을 보였다. 또한 랜덤하게 오염시킨 데이터의 인식률이 특정 값 아래로 내려가지 않은 현상도 이와 비슷하게 설명할 수 있겠다.

[표 20]은 1 비트와 2 비트 인코딩 시, 복구를 진행했을 때, 복구 횟수와 원래 학습 데이터와의 거리를 정리한 것이다. 인식률 100%까지 복구가 된 2 비트 인코딩 시의 시뮬레이션에서는 원래 데이터와 복구된 데이터의 거리가 거의 10 비트 정도로 수렴하는 것을 확인할 수 있으며, 학습의 대상을 반대로 보았을 때 복구된 데이터가 Correlational PLM에 상당히 overfitting된 것으로 해석할 수 있다. 이와 대조적으로 복구된 데이터의 인식률이 100%까지는 올라가지 않은 1 비트 인코딩의 경우에는 오염 횟수가 증가함에 따라 복구 횟수와 거리가 조금씩 증가한다.

표 20. 복구 횟수와 원래 학습 데이터와의 평균 차이

corruption	repair count 2	distance 2	repair count 1	distance 1
1	233.48	10.59	8.18	8.39
5	232.07	10.50	9.02	10.05
10	236.60	10.55	10.12	12.25
15	240.54	10.66	11.16	13.77
20	249.87	10.63	11.98	15.27
30	247.73	10.58	13.04	17.62
40	242.77	10.57	14.13	19.02
50	254.13	10.55	14.16	20.86
64	253.72	10.57	14.73	21.95

3.2.5.3 복구 시도 횟수에 따른 변화

앞에서의 시뮬레이션에서는 오염 횟수를 변화시켜 가면서, 라이브러리 내의 모든 DNA들에 의해 복구를 시도하도록 충분히 복구 시도 횟수를 주어 시뮬레이션했다. [그림 8]에서 확인했듯이, 총 64개의 비트 중에서 랜덤하게 32회만 오염시켜도 충분히 엔트로피가 높아지며, 복구를 시도하기에 충분히 변형된 형태의 비트 조합을 얻어낼 수 있다. 따라서 앞으로의 시뮬레이션에서는 32회 오염을 시키고, 복구 회수를 조절함으로써 인식률과 엔트로피가 어떻게 변화하고 복구가 어떤 식으로 되는지 확인해보고자 한다.

우선 1 비트 인코딩을 한 경우에 대해 살펴보자. [표 21]은 1비트 인코딩 시, 복구 시도 횟수에 따른 데이터의 변화를 정리한 것이다. 즉, 32회 오염을 시키고, 복구 시도 횟수를 10회에서 350회까지 늘려가면서 시뮬레이션했다. [표 21]에서 count는 복구 시도 횟수, training은 학습 데이터의 인식률, t_entr은 학습 데이터의 엔트로피, corrupted는 오염된 데이터의 인식률, c_entr은 오염된 데이터의 엔트로피, repaired는 복구된 데이터의 인식률, r_entr은 복구된 데이터의 엔트로피, entr_diff는 오염된 데이터와 복구된 데이터의 엔트로피 차이, repair는 실제 복구가 일어난 평균 횟수, distance는 원본 비트맵 정보와의 Hamming Distance 차이의 평균이다.

표 21. 1비트 인코딩 시, 복구 시도 횟수에 따른 데이터

count	training	t_entr	corrupted	c_entr	repaired	r_entr	entr_diff	repair	distance
10	82.91	12.87	35.86	32.08	46.48	31.69	0.39	1.58	20.10
20	82.65	12.87	35.83	32.09	56.73	31.21	0.88	3.00	19.86
30	83.25	12.87	35.96	32.09	65.70	30.62	1.47	4.36	19.69

40	84.04	12.87	36.43	32.09	74.18	29.82	2.27	5.59	19.35
50	83.02	12.87	35.75	32.14	78.94	29.30	2.84	6.62	19.32
60	83.57	12.87	35.75	32.12	83.03	28.41	3.71	7.78	18.98
70	83.54	12.87	35.44	32.08	87.96	27.70	4.38	8.58	18.95
80	83.20	12.87	35.15	32.17	90.84	26.84	5.33	9.86	18.55
90	82.86	12.87	36.35	32.11	93.14	26.07	6.04	10.50	18.42
100	83.15	12.87	35.94	32.09	95.18	25.37	6.72	11.03	18.46
110	83.46	12.87	32.09	32.09	96.18	24.52	7.57	11.95	18.14
120	83.15	12.87	35.78	32.07	97.67	22.74	9.33	13.39	17.76
150	83.83	12.87	35.91	32.12	98.77	21.07	11.05	14.68	17.62
200	84.07	12.87	36.67	32.09	99.68	18.03	14.06	16.38	17.34
250	83.70	12.87	35.75	32.09	99.86	14.84	17.25	18.24	17.13
300	84.14	12.87	35.99	32.08	99.97	12.81	19.27	19.14	16.73
350	84.61	12.87	35.23	32.12	100.00	11.46	20.66	19.48	16.92

학습 데이터의 인식률, 학습 데이터의 엔트로피, 오염된 데이터의 인식률, 오염된 데이터의 엔트로피는 대조군 데이터에 해당한다. 즉, 각각의 시뮬레이션에서 복구된 데이터와 비교를 수행함에 있어 데이터 자체에 문제가 없음을 의미한다.

먼저 복구 시도 횟수가 증가함에 따라 복구된 데이터의 인식률은 증가하고, 복구된 데이터의 엔트로피는 감소한다. 이러한 경향은 앞에서도 이미 확인한 바 있다. 100%의 인식률을 보인 350회의 복구 시도(실제 평균 복구 횟수는 19.48) 시뮬레이션을 보면, 엔트로피가 11.46으로 원래 학습 데이터의 엔트로피인 12.87과 많이 비슷해진 것을 알 수 있다. 또한 300회의 복구 시도(실제 평균 복구 횟수는 19.14)에서도 99.97%의 인식률을 보였으며, 그 엔트로피 값도 원래 학습 데이터의 엔트로피와 거의 비슷하다.

350회의 복구 시도 조건이 인식률을 높이고, 엔트로피도 심하게 감소시키지 않은 조건이라는 판단 하에, 실제 비트맵 이미지가 어떻게 변화하는지를 살펴보았다. [그림 9], [그림 10]은 10 종류의 숫자 중에서 0과 2 중에서 랜덤하게 1개씩 뽑아서 이미지를 출력한 것이다. 눈으로 보기에도 복구를 많이 시도할수록, 원래의 숫자로 판별할 수 있는 형태로 변형되어 가는 것을 확인할 수 있다.

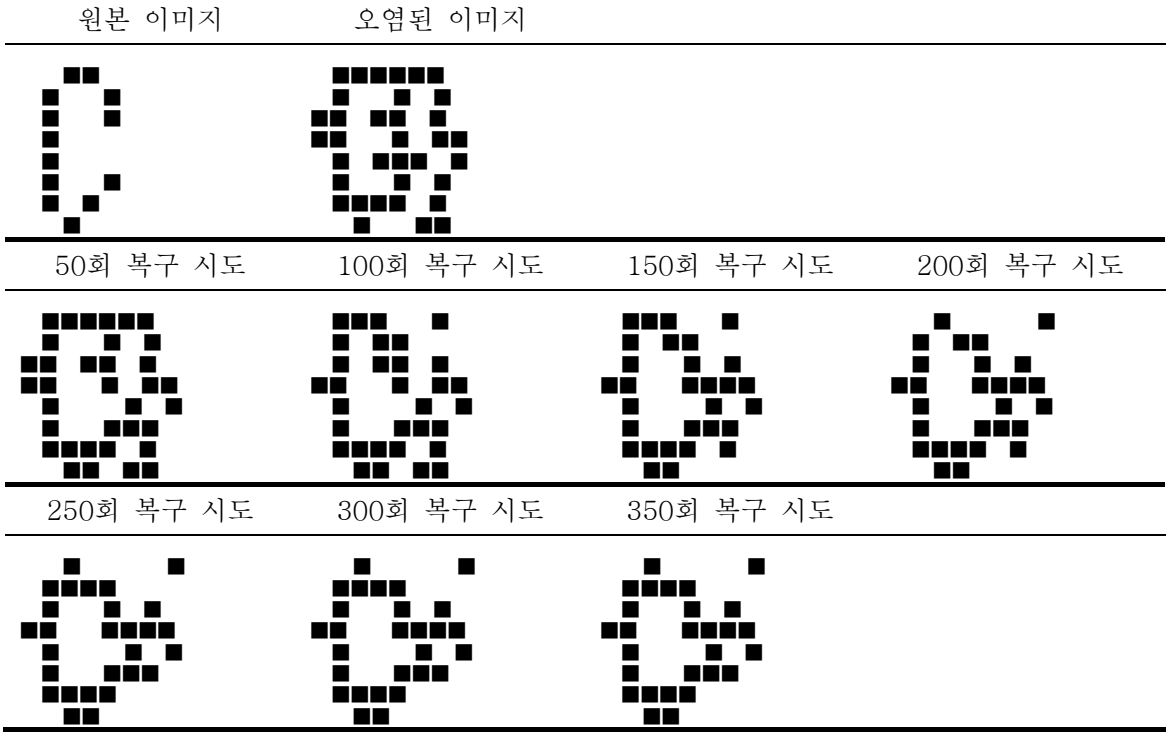


그림 9. 숫자 0의 비트맵 이미지 변화

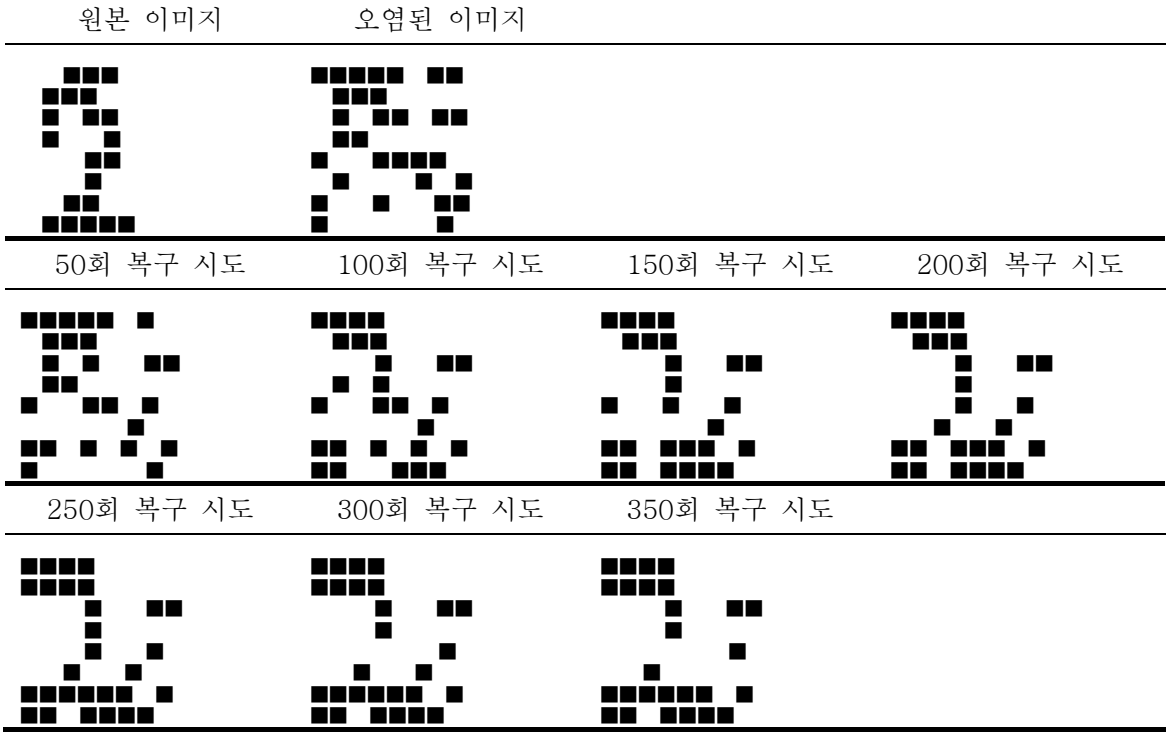


그림 10. 숫자 2의 비트맵 이미지 변화

2 비트 인코딩을 하고, 64회 오염을 한 데이터에 대해 8064회 복구 시도를 한 데이터를 살펴보면, 100%의 인식률을 보이도록 오염된 데이터가 변경된 것을 확인할 수 있다. 인식률

면에서는 상당히 좋은 성능을 보이지만, 이와 함께 고려해야 할 점으로 원래 데이터에서 평균 10 비트 이상 차이가 나도록 데이터가 복구되었다는 점이다. 그리고 이 데이터의 엔트로피는 4 정도로 매우 낮았다.

앞에서 데이터를 분석하면서 100%의 인식률과 4 정도의 낮은 엔트로피는 Correlational PLM에 오염된 데이터가 복구되면서 너무 많이 Overfitting된 것이 아닐까 하는 의문을 던졌었다. 그리고 오염되기 전의 데이터가 12.87의 엔트로피를 갖는 집합이었다는 점을 감안하면, 데이터를 복구하면서 이 수치를 반영하는 성능 평가 방법을 고려하기 위해서, 복구 횟수를 늘려가면서 엔트로피가 오염되기 전의 데이터의 엔트로피인 12.87과 비슷하면서 인식률이 높아지는 조건을 찾기 위해 시뮬레이션을 수행했다.

[표 22]은 2비트 인코딩 시, 복구 시도 횟수에 따른 데이터의 변화를 정리한 것이다. 즉, 32회 오염을 시키고, 복구 시도 횟수를 50회에서 7000회까지 늘려가면서 시뮬레이션했다. [표 22]에서 count는 복구 시도 횟수, training은 학습 데이터의 인식률, t_entr은 학습 데이터의 엔트로피, corrupted는 오염된 데이터의 인식률, c_entr은 오염된 데이터의 엔트로피, repaired는 복구된 데이터의 인식률, r_entr은 복구된 데이터의 엔트로피, entr_diff는 오염된 데이터와 복구된 데이터의 엔트로피 차이, repair는 실제 복구가 일어난 평균 횟수, distance는 원본 비트맵 정보와의 Hamming Distance 차이의 평균이다.

표 22. 2비트 인코딩 시, 복구 시도 횟수에 따른 데이터

count	training	t_entr	corrupted	c_entr	repaired	r_entr	entr_diff	repair	distance
50	84.27	12.87	33.92	32.06	76.3	28.95	3.11	6.61	17.33
100	84.38	12.87	33.5	32.1	92.88	24.91	7.19	11.68	15.25
150	83.88	12.87	33.79	32.06	97.85	20.81	11.25	15.91	13.87
200	83.96	12.87	34.44	32.05	99.68	17.35	14.70	19.12	12.87
250	83.96	12.87	34.24	32.05	99.68	14.25	17.80	21.74	12.13
300	83.96	12.87	34.86	32.11	99.76	12.23	19.88	24.23	11.81
350	83.31	12.87	34.21	32.11	99.92	10.22	21.89	26.43	11.46
400	83.54	12.87	32.98	32.08	100	8.54	23.54	27.98	11.15
500	83.41	12.87	32.98	32.1	99.97	6.71	25.39	31.52	10.83
600	83.39	12.87	33.95	32.04	100	5.4	26.64	34.43	10.84
700	83.75	12.87	34.05	32.11	100	4.94	27.17	38.57	10.75

우선 200회의 복구 시도에서 이미 99% 이상의 인식률을 보였다. 실제 복구 횟수를 살펴보면 19.12인데, 이 값은 1 비트 인코딩에서 99% 이상의 인식률을 보인 300회 혹은 350회 복구 시도 조건에서 보인 실제 복구 횟수인 19.14, 19.48과 비슷한 값이다. 이것으로부터

32회의 오염 시도 결과 충분히 오염된 데이터에서 대략 20 비트 정도를 바꾸면 인식 가능한 패턴의 데이터로 복구된다는 짐작을 할 수 있다.

그리고 400회 복구 시도 횟수 부근에서 100%의 인식률을 보이기 시작하는데, 1비트 인코딩에서 보인 300~350회의 복구 시도 횟수보다는 약간 많지만 거의 비슷한 값을 보인다. 하지만 실제 비트맵의 움직임을 눈으로 살펴보면 동일한 100%의 인식률일지라도 2비트 인코딩의 경우가 더 원본 이미지와 비슷한 패턴으로 복구가 이루어진다. 이는 비슷한 조건에서 2 비트 인코딩의 경우가 원래 학습 데이터에 숨어있는 각 비트들의 상관 관계를 더 잘 학습한다고 생각할 수 있다. 즉, 여기에서도 확인할 수 있듯이 숫자 비트맵의 각 비트들은 서로 독립이 아니다.

[그림 11], [그림 12]는 은 10 종류의 숫자 중에서 0과 2 중에서 [그림 9], [그림 10]과 동일한 데이터에 대해 이미지를 출력한 것이다.

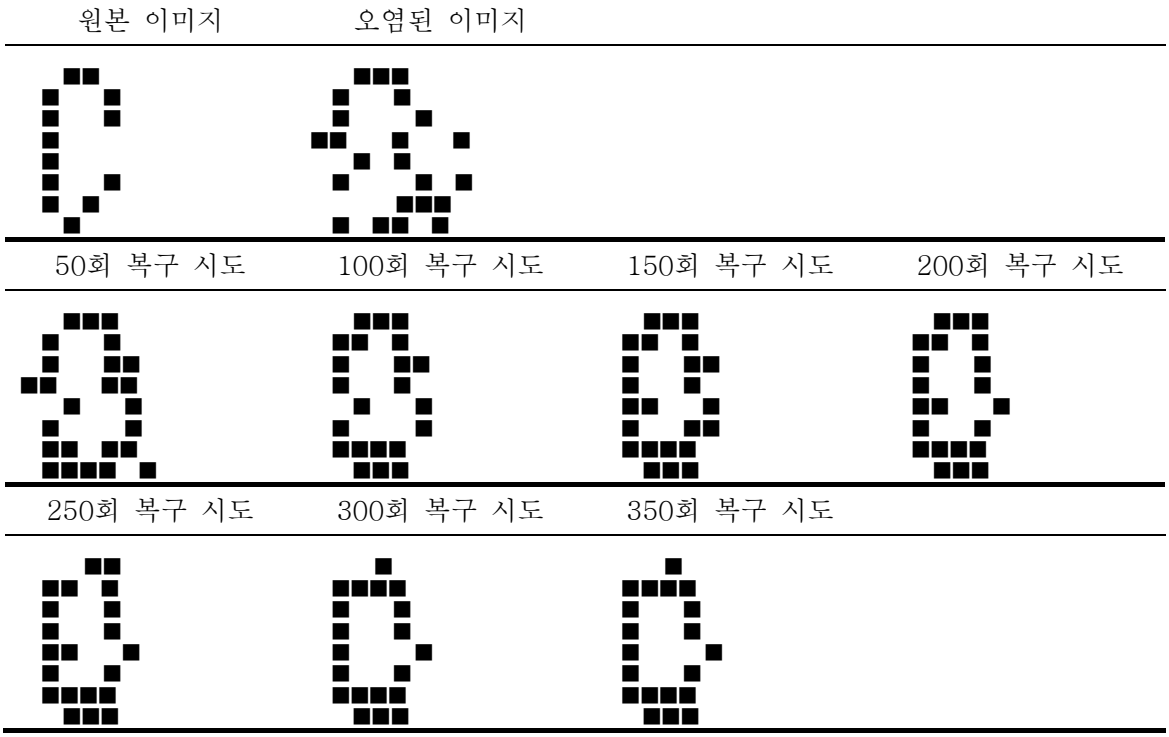


그림 11. 숫자 0의 비트맵 이미지 변화

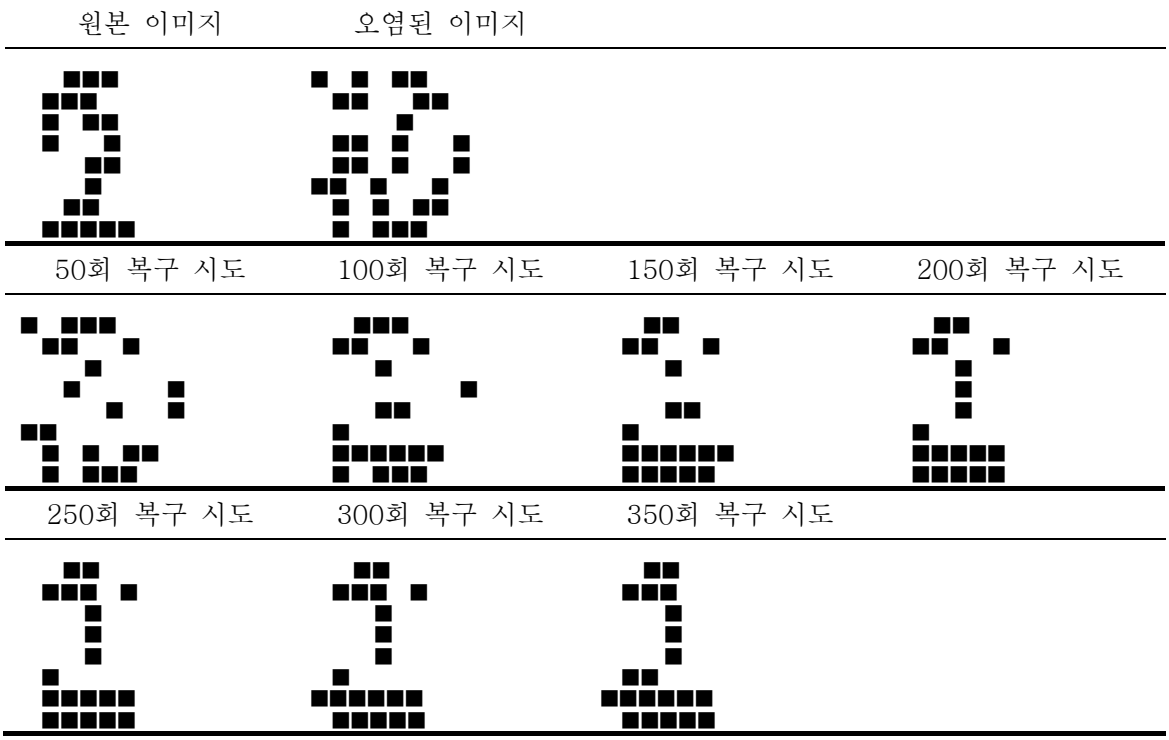


그림 12. 숫자 2의 비트맵 이미지 변화

[그림 11], [그림 12]와 [그림 9], [그림 10]을 비교해보면, 오른쪽 상단부의 불특정한 점들이 2비트 인코딩에서는 거의 사라진 것을 확인할 수 있다. 이는 역시 인코딩에 들어간 비트 수가 늘어나면서, 각 비트들의 상관 관계에 대한 정보가 Correlational PLM의 학습에 많이 반영된 것으로 해석할 수 있다.

4 결론 및 토의 사항

4.1 요약

본 논문에서는 **오류! 참조 원본을 찾을 수 없습니다.**장에서 설명한 숫자 인식 데이터를 이용하여, 3장에서는 PLM으로 패턴 분류 문제에 접근해 보았으며, **오류! 참조 원본을 찾을 수 없습니다.**장에서는 패턴 완성 문제에 접근해 보았다.

4.1.1 패턴 분류 문제

직접 구현한 PLM 시뮬레이션 애플리케이션은 각 숫자 이미지에 대한 비트맵 정보에서 1 비트와 2 비트 그리고 3에서 20 비트 사이의 비트 수를 하나의 DNA에 인코딩한 경우에 대해 학습과 테스트를 수행했다. 3 비트 이상의 경우에는 성능 상의 문제로 인코딩된 DNA가 표현할 수 있는 모든 공간에 대해 실질적인 테스트를 수행할 수 없기 때문에, 3 비트 이상에 대해서는 샘플링을 부분적으로 랜덤하게 수행하는 방식을 적용했다.

시뮬레이션 결과 비트 1 인코딩에서는 83.19%, 비트 2 인코딩에서는 84.06%의 평균적인 인식률을 보였다. 또한 비트 수를 늘려갈수록 라이브러리에 들어가는 DNA 종류 수 보다는 늘어가는 비트 수가 인식률에 기여하는 점이 크다는 것을 확인할 수 있었다. 이는 서로 연관 관계를 갖는 여러 비트들에 대한 정보를 포함할 수 있는 가능성을 높여주는 것이, 랜덤하게 초기 DNA의 종류 수를 늘려주는 것보다 더 중요하다는 예측을 가능하게 했다.

여러 조건에서 실험을 반복해서 수행한 결과, 93.53%의 인식률을 보이는 조건을 찾아낼 수 있었다. 학습 데이터의 분포를 고려한 샘플링, 업데이트 비율/Threshold/반복 횟수의 튜닝, 랜덤 샘플링 방식의 개선을 통해서 이러한 인식률을 얻을 수 있었다.

4.1.2 패턴 완성 문제

20비트 정도의 고 차원의 인코딩의 경우에 대해 학습을 효율적으로 시키는 시뮬레이션은 이미 패턴 분류 문제에서 수행했기 때문에, 패턴 완성 문제에서는 1 비트와 2 비트 인코딩을 한 경우에 Correlational PLM으로 오염된 데이터를 복구하는 것에 대해 테스트를 수행했다. 데이터를 오염시키는 방법은 랜덤하게 선택한 비트의 값을 뒤집는 것으로 했으며, 중복을 허용하도록 했다. 그리고 오염된 데이터를 복구하는 알고리즘은 학습된 라이브러리 내에 클래스 판별에 크게 기여하는 DNA의 정보를 바탕으로 오염된 데이터의 비트 정보를 수정하도록 했으며, 원래 데이터와 복구된 데이터의 거리는 Hamming Distance와, 데이터 집

합간의 엔트로피를 측정하는 것으로 대신했다.

시뮬레이션 결과 1 비트 그리고 2 비트 인코딩 모두, 오염 횟수를 시킴에 따라 오염된 데이터의 인식률은 떨어지고, 엔트로피는 점점 증가했다. 그리고 라이브러리 내의 모든 DNA에 대해 오염된 데이터를 복구하도록 시도한 시뮬레이션 결과에서는 1 비트 인코딩의 경우에는 97~99%의 인식률 그리고 2 비트 인코딩의 경우에는 100%의 인식률을 보였다. 그리고 1 비트 인코딩의 경우에도 복구 시도 횟수를 늘리면 약 350회 정도에서 100%의 인식률을 보였다. 하지만 이 부분은 2 비트 인코딩의 경우 약 63배나 더 많은 복구 시도를 하였지만, 비슷한 엔트로피 차이와 인식률을 보이는 조건에서의 평균 복구 횟수는 19회 정도로 비슷하였으며, 복구 시도를 많이 할수록 인식률이 더 높아지는 경향도 비슷함을 확인할 수 있었다. 또한 실제 복구가 되어가는 데이터를 실제 비트맵의 형태로 출력해본 결과, 2비트 인코딩의 경우가 훨씬 더 원본 이미지에 가까우며 눈으로 식별하기에도 좋은 형태로 복구가 이루어졌다.

4.2 고찰

4.2.1 엔트로피만을 이용한 분류

[표 2]에서 볼 수 있듯이, 각 숫자는 차이가 작지만 엔트로피에 차이가 있다는 것을 확인할 수 있다. 물론 본 연구에서는 PLM을 이용해서 숫자 인식 문제를 해결하는 것이지만, 단순히 엔트로피만을 이용해서도 어느 정도 숫자 인식을 할 수 있을 것으로 보인다.

4.2.2 학습 순서를 달리한 경우에 대한 고찰

신경망(Neural Network)의 경우에는 클래스가 다른 학습 데이터를 학습하는 순서를 달리함으로써 성능을 향상시킬 수 있었다. 가령 한 클래스의 학습 데이터를 다 학습하고 다른 클래스의 학습 데이터를 학습하는 것보다, 각 클래스의 학습 데이터를 번갈아 가면서 학습하는 것이 더 좋은 성능을 보인다.

하지만 PLM을 구현한 이번 애플리케이션의 경우에는 학습 순서를 달리해도 결과가 크게 달라지지 않았다. 이는 아마도 실험에서 PCR 증폭에 해당하는 DNA 개체 수 증가 과정 이후, 전체 DNA 개수를 일정하게 맞추기 위해서 각 DNA 종류의 개수를 조정해주는 희석(Dilution) 과정이 있었기 때문으로 생각해볼 수 있다.

4.2.3 시간 복잡도와 공간 복잡도에 대한 고찰

PLM을 통한 실험을 진행하면서 한가지 드는 생각은, 시리얼로 계산을 수행하는 컴퓨터가 겪는 시간 복잡도의 문제를 아보가드로 수에 비례하는 매우 넓은 공간 상으로 전환한 것으로 해석할 수 있겠다는 점이었다.

즉, 비관적으로 생각을 하자면 아보가드로의 수에 비례하는 DNA 분자의 수가 매우 큰 숫자이기는 하지만, 이는 문제의 복잡도가 증가하다 보면 어느 순간에는 소진될 가능성이 있는 제약을 갖는 표현 공간이라는 점이다.

그럼에도 불구하고 라이브러리에 들어가는 DNA의 종류가 10,000,000을 넘어가는 경우에 수 시간씩 걸리는 문제 해결 시간이, DNA 분자의 병렬적인 Hybridization 과정을 통해 순식간에 이루어질 수 있다는 특징은 동일한 문제를 매우 빠른 시간 안에 풀 수 있는 가능성을 제시했다는 점에서 상당히 긍정적으로 평가할 수 있겠다.

4.3 향후 연구 과제

4.3.1 다른 분류 방법과의 비교

본 연구에서는 단순히 PLM을 이용해서 분류(Classification) 문제를 접근했다. 하지만 분류에 사용할 수 있는 방법은 많다. 대표적인 예로, *k*-nn 방법이나 신경망(Neural Network)을 들 수 있다. 다양한 분류 방법을 이 문제에 적용해보고, PLM으로 접근한 방법과 어떤 차이와 장단점이 있는지 분석해보는 것도 의미가 있을 것 같다.

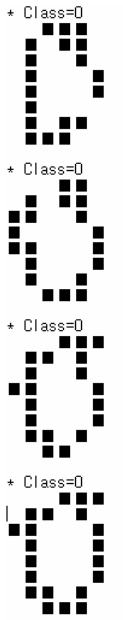
k-nn(*k*=3) 방법으로 숫자 인식 문제를 접근했을 때, 93.10%의 정확도를 보였다고 한다. 본 연구에서의 시뮬레이션 수행 결과로는 20 비트로 인코딩을 하고 DNA 종류 수를 10,000,000으로 한 경우에 93.53%의 인식률을 보였다. *k*-nn(*k*=3)으로 학습하는 경우에 드는 공간/시간 상의 복잡도가 어느 정도 되는지는 모르겠으나, PLM으로도 충분히 훌륭한 정도의 학습을 시킬 수 있다는 것을 어느 정도 확인할 수 있었다.

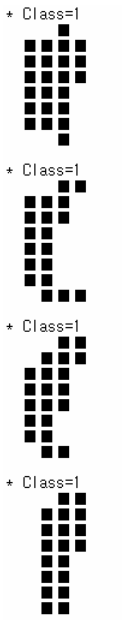
4.3.2 학습 데이터의 개선

[그림 13], [그림 14], [그림 15]은 클래스 0, 1, 9의 비트맵 데이터 중 일부분을 직접 그려서 눈으로 확인해 본 것들이다. 비교적 인식이 잘 되는 0은 눈으로 보기에도 0의 형태를 띄고 있지만, 1은 상당히 두꺼운 형태를 띈 것들이 많았으며, 인식률이 가장 안좋았던 클래스 9는 상당수의 비트맵들이 눈으로 보기에도 식별하기에 어려운 것들이 많았다.

만약 학습 데이터에서 각 비트를 0과 1로 구분하는 것이 아니라, 0에서 255 사이의 숫자로

명암을 구분할 수 있도록 한 뒤, 라이브러리에 학습 시킬 때는 각 비트의 숫자를 보고 업데이트 비율을 동적으로 조절할 수 있게 한다면 좀더 학습 효과가 좋았을 것 같다.





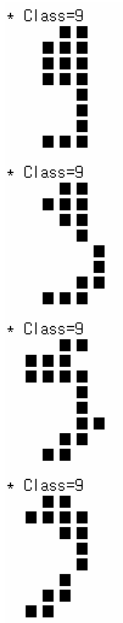


그림 13. 클래스 0의 데이터 **그림 14. 클래스 1의 데이터** **그림 15. 클래스 9의 데이터**

그리고 실험적인 부분을 고려하자면, 각 비트에 해당하는 값들을 참조해서 PCR 시간을 조절하는 것으로 어느 정도 구현이 가능할 것 같다는 생각도 든다.

4.3.3 실제 실험을 통한 구현

PLM의 강점은 실제 DNA의 Hybridization을 통한 병렬 연산에 있다. 따라서 이렇게 시뮬레이션만으로 계산 모델을 만드는 것만으로는 큰 의미가 없을지도 모른다. 따라서 PLM의 모델을 실제 실험으로 구현할 수 있도록 발전하는 것이 중요하다고 본다.

물론 시뮬레이션으로 구현을 해본 바로는 실제 실험 과정이 단순 반복 과정이 많을 것으로 보인다. 즉, 학습 데이터를 이용해서 학습을 진행하려면 튜브 내에 증폭하고자 하는 DNA를 증폭시킬 수 있도록 상보된 염기를 갖는 DNA를 넣어주고, 이를 PCR(Polymerase Chain Reaction)로 증폭시키고, 이 중 일부분을 원래의 튜브에 넣어주고, 다시 희석(Dilution)하는 과정을 반복해야 한다.

물론 PCR 증폭과 희석 과정이 약간의 실험상의 오차를 동반하기 때문에 과연 PLM 모델을 실제 실험으로 구현할 수 있을지 의문이 들기는 하지만, 대부분 이러한 기술적인 문제는 시간이 지나면 해결이 되는 경향을 보인다.

물론 이러한 사항 이외에도 고려해야 할 점들은 많다. 학습하려는 데이터를 실제 DNA 염기로 인코딩할 때, 어떤 염기 서열로 디자인하는지에 따라 실험 결과가 달라질 수 있다. 왜냐하면 C와 G 사이의 수소 결합이 3개로 A와 T 사이의 2개짜리 수소 결합에 비해 더 강력하기 때문에다. 이러한 특성 때문에, PCR을 이용해서 특정 유전자의 돌연변이를 만들 때도 Primer의 CG 비율 등의 특성에 따라 Primer의 길이를 달리 하거나, PCR을 할 때 온도를 조절해주기도 한다.

그리고 본 연구에서는 시뮬레이션 프로그램을 작성할 때, 정확하게 각각의 상보적인 염기서열이 결합한다고 가정을 하고 했다. 하지만, 실제 DNA의 Hybridization에는 상보적이지 않은 결합도 존재한다. 심지어 A, T, C, G의 각 염기들은 Keto 형태와 Enol 형태의 두 형태로 존재한다. 물론 좀더 안정한 형태가 95%로 우세이긴 하지만, 불안정한 형태도 당연히 존재하며, 불안정한 형태의 염기는 다른 염기와 결합하는 것이 가능하다. 어쨌든 이러한 불완전한 특성을 이용하면, PLM이 좀더 범용적인 학습을 하도록 구현하는 것이 가능해 보인다. 또한 PCR에 사용하는 Taq Polymerase도 100%의 Fidelity를 보이는 것이 아니기 때문에(특히 PCR을 15 사이클 이상 돌리게 되면, Mutation Rate가 높아진다), 이러한 특징은 PLM의 발전 방향에 반드시 포함되어야 한다고 본다.

참고 문헌

[장 03] 장병탁, “바이오분자 컴퓨터 기술”, 물리학과 첨단기술, 2003년 3월호, pp. 13-14, 2003

[A 94] Adleman, L., Molecular Computation of Solutions To Combinatorial Problem, Science 266, Oct. 1994

[B 82] Bennett, C.H., The Thermodynamics of Computation, Internat. J. Theoret. Phys. 21, Sep. 1982

[M 98] Maley, C. C., DNA Computation: Theory, Practice, and Prospects, Evolutionary Computation 6, Feb. 1998

[Z 02] Zhang, B.-T., Molecular Information Processing Technologies, The Magazine of IEEK 29(3), p. 58, 2002

[ZJ 04] Zhang, B.-T. and Jang H.-Y., Molecular Learning of wDNF Formulae, DNA11, 2004