[UCI]I804:24011-200000359933

2021년 2월
석사학위 논문

딥러닝기술을 활용한 무선광통신시스템 신호생성에 관한 연구

조선대학교 대학원

전자공학과

황 용 운

딥러닝기술을 활용한 무선광통신시스템 신호생성에 관한 연구

A Study on Signal Format Generation for Optical Wireless Communication Link using Deep Learning

2021년 2월 25일

조선대학교 대학원

전자공학과

황 용 운

딥러닝기술을 활용한 무선광통신시스템 신호생성에 관한 연구

지도교수 이 충 규

이 논문을 전자공학석사학위신청 논문으로 제출함

2020년 10월

조선대학교 대학원

전자공학과

황 용 운

황용운의 석사학위논문을 인준함

위원장 조선대학교 교 수 고 낙 용 (인)

위 원 조선대학교 교 수 강 문 수 (인)

위 원 조선대학교 교 수 이 충 규 (인)

2020년 11월

조선대학교 대학원

목 차

도 목 차

표 목 차

ABSTRACT

A Study on Signal Format Generation for Optical Wireless Communication Link using Deep Learning

Hwang, Yong Woon
Advisor : Prof. Lee Chung Ghiu Ph.D.
Dept. of Electronic Engineering,
Graduate School of Chosun University

Many state-of-the-art technologies, such as Cloud Computing, AI(Artificial Intelligent), IoT(Internet of Things), and VR(Virtual Reality), have been developed rapidly with communication systems. Out of various communication systems, OWC(optical wireless communication) having advantages in viewpoints of high-frequency band, security of information, and free allocation frequency, which makes at a promising a next-generation communication. However, the performance of OWC suffered from deteriorating channel effects. Thus, appropriate signal formats for OWC channels are needed.

In this thesis, I propose a DL-based signal format generation/recommendation system for OWC links. The system incorporates a physical optical channel model and deep learning neural network for generating signal sets for binary OWC system. It is simulated by a computer with assumed physical conditions for OWC. In this study, I specifically focus on generating signal sets for a parallel LED array to decide binary symbols (HIGH and LOW) for binary communication by classifying the recovered image pattern at the charge-coupled device(CCD).

제1장 서론

지속적인 사회 기술의 발달을 통해서, 자율주행 차량 간 통신, 드론 간 통신, 사물인터넷의 데이터 수집 및 처리 제어, 클라우드 컴퓨팅 처리 통신, VR 가상현실, 스트리밍 서비스 그리고 군사보안 통신 등과 같은 수많은 데이터를 무선 통신으로 빠르게 처리하면서 보안을 중요시하는 기술의 수요는 폭발적으로 증가해왔다.

그러나 기존에 널리 상용화 되어진 RF 통신은 본질적인 주파수 대역폭의 한계로 인해서 처리해야 할 데이터가 많아지면 많아질수록 기술적 한계에 부딪칠 수밖에 없고, 주파수 간섭문제로 인한 주파수 할당 문제에 있어서 자유롭지 못하다. 또한 필요 이상의 범위에서 물리적으로 전파가 노출 되는 점에서 보안상의 한계가 존재한다.

이러한 문제를 보완하는 기술로서 무선광통신 기술은 차세대 통신기술로 기대 받으며 활발히 연구되어오고 있다. THz 대역의 주파수를 가져서 이론상 많은 데이터를 송신할 수 있고, 빛의 직진성이 강한 특성을 이용하여 통신 범위의 제어가 가능하여서 보안성이 강하며, 주파수의 할당 문제에 있어서 기존 상용화 되어진 무선 RF통신과의 전자파와 간섭이 없기에 법적으로도 자유롭다 [1].

이러한 장점들이 존재하지만, 통신 채널의 물리적인 특성의 영향을 다르게 받는, RF와 빛의 차이로 인해서 발생하는 문제들이 존재한다. 이러한 문제들을 해결하기 위해서 기존에는 연구자가 실험이나 시뮬레이션을 반복하며 빛의 신호에 영향을 미치는 채널특성을 확인하고 성능을 평가하였으며, 채널로부터 미치는 영향이 적은 신호를 연구자가 직접 고안해서 다시 검증하는 방향으로 발전해왔다.

그러나 기존의 신호 생성 접근법은 파라미터를 조정하고 신호를 분석하는데 있어서 연구자 개인의 선행지식 수준이나 경험에 지나치게 의존한다는 한계가 존재해왔다. 또한 무선광통신 시스템에서 여러 가지 물리적 환경 변수가 가정 될 수 있는데, 이러한 특성이 변할 때마다 채널특성에 최적화된 신호를 찾는 것은 많은 시간과 비용이 소요되었다.

본 논문은 이와 같은 문제들을 해결하기 위한 방법으로, 최근 혁신적인 방법으로 각종 분야들에 적용되어 놀라운 성과들을 보여주는 딥러닝을 무선광통신 시스템의 신호 생성에 적용하는 것을 제안한다.

딥러닝이란 머신러닝 기술들 중 하나이며 인간의 신경망의 원리를 일부 모방하여 나온 모델이다. 딥러닝의 기본 특성으로는, 전문가가 내부의 알고리즘의 파라미터를 전부 이해하고 하드코딩하여 설계 하는 기존 방식이 아닌, 입력데이터와 출력데이터를 이용하여 인공신경망이 스스로 파라미터를 갱신하여 입출력 함수를 만들어내는 것이다 [2].

이러한 딥러닝 모델 중에 AE(AutoEncoder) 방식이라는 End-to-End 생성모델을 무선광통신시스템에 응용하면 설정한 물리층에 적합한 신호를 생성하는 것이 가능하며, 더 나아가 생성한 신호들의 성능을 평가하여 통신 시스템의 성능을 개선시킬 수 있다 [3].

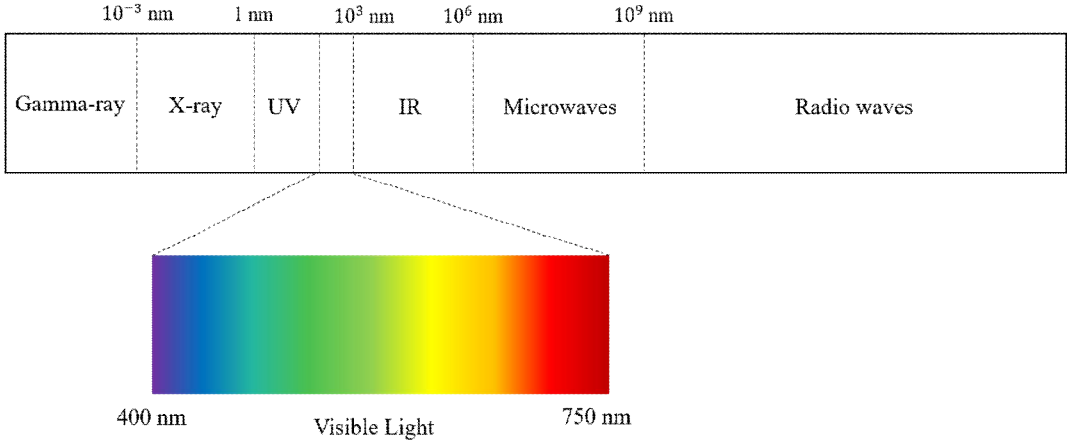
본 연구는 무선광통신 기술 분야 중에서도 기존에 아직 연구가 많이 부족한 무선 병렬 LED 통신에 초점을 맞춰서 신호 생성 시뮬레이션을 진행하였고 평가하였다.

제2장 무선광통신 시스템

제1절 빛의 분류

무선광통신 시스템은 빛의 물리적인 성질을 이용한 무선 통신 방식이다. 따라서 무선광통신 시스템을 이해하고 응용하기 위해서는 빛의 성질을 이해할 필요가 있다.

빛은 물리적으로 파동과 입자의 성질을 모두 갖고 있으며, 파동으로서의 빛은 전자기파의 일종이며, 입자로서의 빛은 광자로 나타난다. 그리고 빛이 가지고 있는 주파수별로 특성이 달라지므로 다음 그림 2.1.1과 같이 주파수에 따라 분류되어진다 [4].



[그림 2.1.1 전자기파의 스펙트럼]

인간의 눈을 이용하여 인식 가능한 빛을 가시광선이라고 하며 약 400nm에서 700nm 사이의 파장을 갖는 전자기파이다. 직진성이 강하고, 인체에 무해한 특성을 갖고 있다. 더 나아가 기존에 많이 설치되어진 LED를 이용하면 가시광선을 생성할 수 있다는 점에서 범용 통신 수단으로서의 장점이 있다. 이러한 장점을 바탕으로 본 연구에서 다루는 병렬 LED 무선광통신은 이 주파수 대역의 빛을 이용하게 된다.

상대적으로 긴 파장의 전자기파는 강한 회절 특성과 멀리 퍼져나가는 특성을 이용하여 무전통신, 라디오에 응용되어왔다. 하지만 통신수단으로서 주파수 대역 간 간섭현상이 통신 성능을 감소시키는 단점이 있다.

적외선은 열을 전파하는 특성이 있으며, 물체에 열이 있으면 분자에서 방출되어진다. 대표적인 응용사례로는 열을 감지하는 적외선 카메라와 적외선 리모컨 등이 있다.

가시광선보다 짧은 파장의 자외선, X-ray, 감마선은 상대적으로 높은 주파수의 에너지를 갖고 있다. 이러한 전자기파들은 인체의 세포의 DNA를 파괴하여 돌연변이를 발생시켜 발암확률을 높일 수 있어서 인체에 유해한 측면이 존재한다. 그러므로 통신을 위한 수단으로서는 부적절한 측면이 있다. 다른 한 편으로는 에너지와 투과율이 높은 특성을 이용하여 공학적인 측면에서의 에너지발전, NDT(Non-Destructive Testing) 검사와 분석, 의학적인 측면에서 검진 및 치료 등 여러 이로운 측면도 같이 존재한다.

제2절 빛의 성질

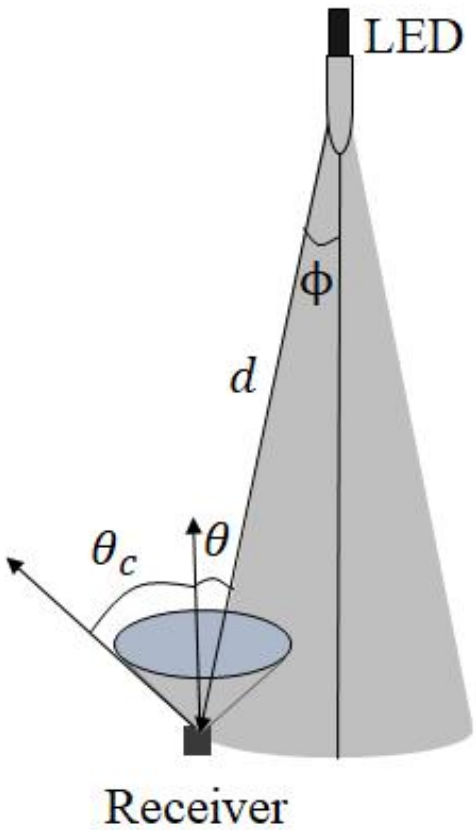
빛은 많은 파동 성질을 갖고 있다. 표 2.2.1과 같이 반사, 굴절, 산란, 회절 현상 등이 대표적으로 존재한다. 이러한 성질들은 무선광통신 시스템에서 빛의 강도에 대한 영향 및 기하학적인 영향을 주므로 고려되어져야 한다 [5].

빛의 성질	현상	특징
반사	빛이 물체에 충돌 시 표면에서 다른 방향으로 꺾이는 현상	빛의 입사각과 반사각의 크기는 항상 같다는 반사의 법칙을 따름
굴절	서로 다른 성질을 갖는 두 매질의 경계면에서 빛이 방향의 각도가 꺾이는 현상	두 매질의 특성에 따라 굴절각이 달라지는 특성이 나타남
산란	빛과 일정 조건의 크기의 입자와 충돌 할 때 빛이 여러 방향을 흩어지는 현상	입자의 크기와 빛의 파장의 크기에 따라 서로 다른 현상이 나타남
회절	빛이 장애물에 충돌 할 때 빛의 파동이 장애물 뒤에까지 돌아서 들어가는 현상	주파수가 낮은 빛의 경우 더 두드러지게 나타남

[표 2.2.1 빛의 성질의 현상과 특징]

제3절 LOS 전파 모델

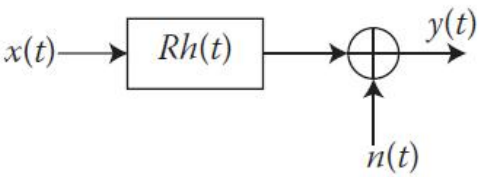
LOS(Line of sight)란 그림 2.3.1에서의 기하학 구조와 같이 가시거리의 빛의 직진 환경을 의미한다. 본 연구에서는 이러한 전파 모델을 적용한 병렬 LED 무선 광통신 시스템을 가정 한다 [6].



[그림 2.3.1 LOS 전파의 기하학 모델]

여기서 d는 LED의 빛이 수신기에 도달한 거리를 나타낸다. Φ는 LED의 빛의 강도가 퍼지는 각을 의미한다. 이 때 수신기의 직각과 이루는 각을 θ라고 한다. θ_c는 수신기가 렌즈를 통해서 빛을 받을 수 있는 최대 각을 의미한다.

그림 2.3.1을 통신시스템 블록 다이어그램 구성으로 표현하면, 다음의 그림 2.3.2로 표현할 수 있다.



[그림 2.3.2 기저대역의 IM/DD 모델을 적용한
무선광통신 시스템]

이러한 시스템의 입출력 특성은 NRZ-OOK IM/DD(Intensity modulation with direct detection) 시스템의 충격응답으로 설명된다.

$$y(t) = Rx(t) \otimes h(t) + n(t) = \int_{-\infty}^{\infty} Rx(\tau)h(t-\tau)d + n(t) \quad (2.1)$$

여기서 R은 광검출기의 응답율을 의미하며 h(t)는 기저대역채널의 임펄스 응답이다. n(t)는 신호 독립적인 샷노이즈를 의미한다. ⊗ 는 컨볼루션 연산을 의미한다.

그림 2.3.1의 광수신기의 FOV (θ_c)에 따라 LOS 광채널의 충격 응답 함수 h(t)는 다음과 같이 결정된다. (수식 3.2)

$$h(t) = f(x) = \begin{cases} \dfrac{2t_0}{t^3 \sin^2(FOV)} & t_0 \le t \le \dfrac{t_0}{\cos(FOV)} \\ 0 & elsewhere \end{cases} \quad (2.2)$$

SNR(Signal-to-noise ratio)과 H(0)는 수식 2.3과 2.4로 계산 할 수 있다.

$$\mathrm{SNR}=\frac{R^2H^2(0)P_r^2}{R_bN_0} \tag{2.3}$$

여기서 N_0는 노이즈의 스펙트럼 밀도이고 $H(0)$는 채널의 DC이득이다.

$$H(0)=\int_{-\infty}^{\infty}h(t)dt \tag{2.4}$$

그림 2.3.1의 모델에서 수신된 광신호의 파워는 수식 2.5 - 2.12를 통해 계산된다. 빛의 방사 강도 패턴의 각 분포 $R_0(\Phi)$는 수식 2.5를 통해 계산할 수 있다. 여기서 m_1은 Lambertian emission의 차수를 의미하며 이는 빛의 강도가 절반이 되는$\Phi_{1/2}$ 각과 관련이 있다.

$$R_0(\Phi)=\begin{cases}\dfrac{(m_1+1)}{2\pi}\cos^{m_1}(\Phi) & \text{for } \Phi\in[-\pi/2,\pi/2]\\ 0 & \text{for } \Phi \geq \pi/2\end{cases} \tag{2.5}$$

$$m_1=\frac{-\ln 2}{\ln(\cos\Phi_{1/2})} \tag{2.6}$$

방사 강도 $S(\phi)$는 다음 수식 2.7로 계산된다.

$$S(\Phi)=P_t\frac{m_1+1}{2\pi}\cos^{m_1}(\Phi) \tag{2.7}$$

활성영역 A_r로 모델링 되어진 effective collection area $A_{\mathrm{eff}}(\theta)$는 다음과 같다.

$$A_{\mathrm{eff}}(\theta)=\begin{cases}A_r\cos\theta & 0 \leq \theta \leq \pi/2\\ 0 & \theta > \pi/2\end{cases} \tag{2.8}$$

여기서 내부의 굴절율 n을 갖는 광이득 $g(\theta)$는 다음 수식 2.9와 같다.

$$g(\theta) = \begin{cases} \dfrac{n^2}{\sin^2\theta} & 0 \le \theta \le \theta_c \\ 0 & \theta > \theta_c \end{cases} \tag{2.9}$$

collection area of lens A_c는 다음수식 2.10의 관계를 갖는다.

$$A_c \sin\left(\frac{\theta_c}{2}\right) \le A_{\mathrm{r}} \tag{2.10}$$

위의 수식들을 바탕으로 수식 2.11을 통해 근사되어진 $H_{\mathrm{los}}(0)$를 계산 할 수 있다. 여기서 $T_s(\theta)$는 광 밴드패스 필터의 송신을 의미한다.

$$H_{\mathrm{los}}(0) = \begin{cases} \dfrac{A_{\mathrm{r}}(m_1+1)}{2\pi d^2}\cos^{m_1}(\Phi)\, T_s(\theta) g(\theta)\cos\theta & 0 \le \theta \le \theta_c \\ 0 & \text{elsewhere} \end{cases} \tag{2.11}$$

최종적으로 송신기의 전력 P_t와 $H_{\mathrm{los}}(0)$를 통해 수신기에서의 수신 전력$P_{\mathrm{r-los}}$ 계산 할 수 있다.

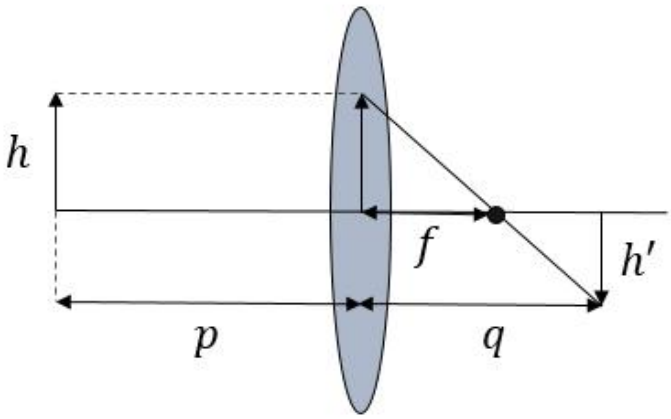
$$P_{\mathrm{r-los}} = H_{\mathrm{los}}(0) P_t \tag{2.12}$$

이러한 기하학적 구조를 바탕으로 배열형 광수신기가 광신호를 측정하게 되며, 광수신기를 구성하는 배열형 광검출기에서 측정된 배열형 영상을 기반으로 딥러닝을 적용하여 통신 성능을 만족하기 위한 이진 신호 심볼(M1, M2)의 집합을 제안한다.

제4절 렌즈 방정식과 배율

무선광통신시스템에서 수신기에서는 빛을 최대한 많이 모으거나 확대하기 위해서 렌즈를 사용한다. 따라서 수신기에 맺힌 이미지 상을 구현하기 위해 렌즈방정식을 적용해야한다.

렌즈방정식은 물체로부터 직접 나온 빛이나 반사된 빛이 렌즈를 통해 굴절되어 센서에 도달 하였을 때, 이미지 상의 길이와 방향이 어떻게 나타나는지에 대한 관계를 구하는 방정식이다 [7].



[그림 2.4.1 물체와 이미지, 렌즈 간 기하학 관계]

그림 2.4.1에서 p는 물체와 렌즈사이의 거리를 의미하며, q는 렌즈와 이미지의 거리를 나타낸다. f는 렌즈로부터의 초점거리이다. h는 물체의 길이, h'은 센서에 맺히는 물체의 크기와 방향을 나타낸다. 다음 (2.13) 수식은 이런 길이의 관계에 대한 렌즈방정식이다. f와 p를 알고 있을 때 q를 계산하는 것이 가능하다.

$$\frac{1}{p}+\frac{1}{q}=\frac{1}{f} \tag{2.13}$$

다음 수식 (2.14) 는 배율 M에 대한 공식이다.

$$M=\frac{h'}{h}=-\frac{q}{p} \tag{2.14}$$

h의 물체의 길이와 h'은 센서에 맺히는 이미지의 비가 배율 M이며, h'은 p와 q 및 h를 통해서 계산할 수 있다.

제3장 머신러닝과 딥러닝의 개념과 원리

제1절 머신러닝의 학습방법 분류

무선광통신 시스템 시뮬레이션의 변복조 과정에 적용하고자하는 머신러닝인 AE를 이해하기 위해서는 우선 머신러닝의 학습방법을 이해할 필요가 있다.

머신러닝 학습방법은 크게 지도학습, 비지도학습, 강화학습으로 분류 할 수 있다 [8]. 이 때 풀고자하는 문제에 따라 학습 방법은 달라질 수 있으며 어떤 데이터를 수집 및 생성해야 하는지 어떤 환경을 구축해야하는지 또한 이 학습 분류에 따라 결정되어진다.

지도학습은 입력 데이터와 출력 데이터 쌍을 기계에게 제시하여 학습 시키는 방법이다. 이 학습 방법은 학습하고자 하는 입출력 쌍이 반드시 있어야만 학습이 가능하다는 단점이 존재하며, 다른 여러 분야의 문제 적용하기 쉽다는 장점을 갖고 있다. AE에서 Decoder 네트워크가 지도학습에 해당하는 학습을 진행한다.

비지도학습은 입력 데이터만을 가지고 기계가 패턴을 학습하는 방법이다. 장점으로는 출력 값을 미리 사용자가 제시하지 않아도 된다는 점이며, 지도학습과 비교했을 때 한정된 문제 해결에만 응용 할 수 있다는 단점이 있다. AE에서 Encoder 네트워크는 비지도 학습을 통해서 학습한다.

강화학습은 기계가 어떤 환경으로부터 상호작용하여 어떤 목표를 달성 했을 때 점수를 높이는 식으로 반복 학습시키는 방법이다. 장점으로는 기계가 어떠한 환경 속에서 목표를 이루기 위해 새로운 강화된 전략을 계속해서 만들어 낸다는 점이며, 단점으로는 기계가 상호작용하는 실제 환경에 대한 데이터가 충분하지 않으면 실제 적용 단계에서는 실패할 가능성이 높다는 점이다.

제2절 머신러닝이 해결하고자 하는 문제의 유형

머신러닝은 또한 표 3.2.1과 같이 분류문제, 회귀문제, 생성모델 등의 문제들을 다룬다 [9].

문제 유형	특징	예시
분류	어떤 특징들로부터 연속되지 않은 것들을 예측하는 문제	숫자 인식과 사물 분류
회귀	어떤 특징들로부터 연속된 값을 예측하는 함수를 만들어 내는 문제	날씨 온도 예측, 자산 가치 예측
생성	어떤 임의의 값을 받아 특징을 생성하는 모델	그림, 음악 생성, 소설 창작

[표 3.2.1 머신러닝 문제의 특징과 예시]

무선광통신에서의 AE의 Encoder 네트워크는 생성모델로서 메시지에 따른 신호 패턴을 생성하고 Decoder 네트워크에서는 Encoder 네트워크로부터 받은 신호를 바탕으로 메시지의 분류 문제를 수행한다.

제3절 머신러닝 알고리즘

머신러닝의 몇 가지 알고리즘으로는 SVM(Support Vector Machine), 유전알고리즘, 인공신경망 등이 있다.

SVM은 서포트 벡터를 이용한 최대마진의 최적분류 패턴을 학습시키는 방법인데, 딥러닝이 나오기 이전까지 분류 문제에 널리 연구되어지고 사용되어왔다 [10].

유전 알고리즘은 유전자의 진화의 원리를 이용한 것이며 개체를 선택하고 교배하거나 일부 돌연변이를 추가해서 각각의 개채들을 적합도 함수로 평가해서 개체를 최적화하는 방식이다 [11].

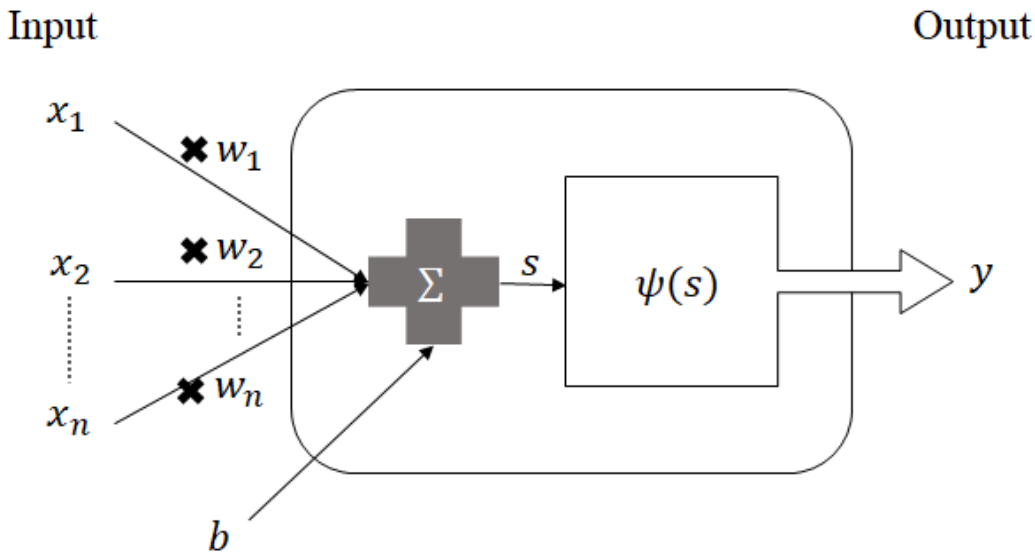
인공신경망의 정의는 인간의 뉴런의 동작 원리의 일부를 수학적으로 모방하여 모델링한 것이다. 이러한 인공신경망을 깊게 쌓아 올린 것이 딥러닝이다. 본 연구에서 사용한 AE는 인공신경망의 딥러닝에 속한다 [12].

제4절 딥러닝의 원리

1. 인공 뉴런 모델

인공신경망에서 기본단위는 인공 뉴런이다. 따라서 인공신경망을 이해하기 위해서는 인공 뉴런 모델을 이해할 필요가 있다.

인공 뉴런 모델은 그림 3.4.1에서 볼 수 있듯이 외부로부터 데이터들을 입력 받아 가중치를 곱하고 전부 더한다. 그리고 이 값에 바이어스를 더해서 활성화 함수에 입력으로 넣으면, 활성화 함수의 조건에 따라 활성화되는 값을 출력하는 구조이다 [13].



[그림 3.4.1 인공 뉴런 모델 구성도]

여기서 $x_1, x_2, \ldots x_n$은 인공 뉴런 모델에 들어오는 입력 값이다. $w_1, w_2, \ldots w_n$은 입력 값에 곱하는 가중치 파라미터들을 의미한다. b는 항상 활성화되는 바이어스 파라미터를 의미한다. 가중치와 입력 값들의 곱과 합 및 바이어스와 합 연산 값은 s를 의미한다. 이 s값을 넣는 함수 ψ를 활성화 함수라고 한다. 활성화 함수에서 출력되는 값 y가 최종 출력 값이다.

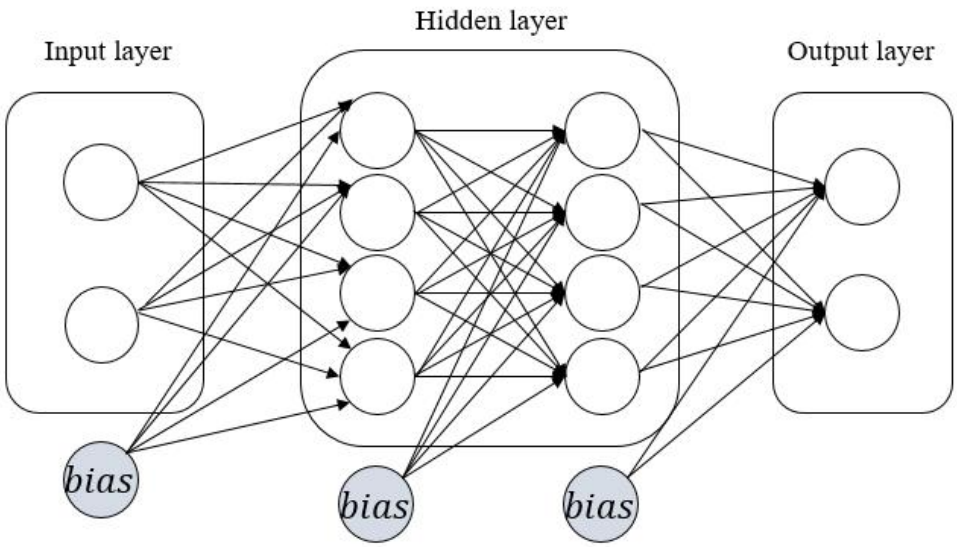
$$y = \psi(s) = \psi((x_1 \times w_1 + x_2 \times w_2 \ldots + x_n \times w_n) + b) \tag{3.1}$$

수식으로 나타내면 위의 (3.1) 공식과 같이 표현 할 수 있다.

이러한 단일 인공 뉴런 모델을 훈련시키면 선형분류문제를 해결하는 것은 가능하였지만 다차원 비선형 문제와 같은 복잡성이 높은 문제와 XOR 문제는 해결이 불가능하다는 단점이 있다.

2. 딥러닝의 구조

딥러닝은 단일 인공뉴런모델의 단점을 보완하기 위해 서로 연결하여 확장한 구조이다. 딥러닝의 구성요소로는 그림 3.4.2와 같이 외부의 입력 데이터를 입력 받는 입력층과, 외부에 드러나지 않는 은닉층 그리고 외부로 어떤 값들을 출력하는 출력층으로 구성된다 [2]. 추가로 바이어스 노드를 외부에 구현해놓은 형태이다. 이러한 딥러닝 모델은 단일 인공 뉴런 모델에서 할 수 없었던 다차원 특징의 비선형 문제들을 해결 할 수 있다.



[그림 3.4.2 딥러닝의 구조]

3. 딥러닝의 학습방법

이러한 신경망들을 학습시키는 원리는, 신경망이 예측한 출력과 정답으로 제시한 출력 데이터간의 손실함수를 최소화 하도록 내부의 층들 간 연결 가중치를 최적화하는 것이다.

이 때 사용되는 학습법의 기본적인 틀이 역전파 알고리즘이다 [14].

역전파 알고리즘은 크게 2가지 단계로 나뉘는데 전향 단계와 후향 단계가 있다.

먼저, 전향 단계에서는, 신경망의 입력 층에 입력을 넣어서 나온 행렬 연산 출력 값과 실제 출력 하고자하는 데이터를 비교한다. 이 때 발생하는 오차를 손실함수를 통해서 확인하는 것이 전향단계이다.

후향 단계에서는 이때의 출력 값들의 손실함수 값을 통해 출력층에서 은닉층과 입력층 방향으로 손실함수의 미분을 연쇄 계산한다. 이후 미분 방향의 크기와 학습률에 따라 신경망의 가중치 및 바이어스 파라미터를 반복적으로 갱신한다. 이후 다시 전향단계로 돌아간다.

이를 통해 최종적으로 손실 함수가 최소가 되는 신경망 파라미터를 찾는 알고리즘이다.

4. 딥러닝의 세부 구성요소

딥러닝에서 활성화 함수란 인공 뉴런의 선형 연산 이후에 비선형 효과를 주기 위한 함수를 의미한다. 활성화 함수에서 대표적으로 사용되는 함수로는 sigmoid, ReLU, Softmax 등이 있다. 과거에는 sigmoid 함수를 주로 사용해오다가 신경망의 학습 중에 손실함수의 기울기가 점점 소실되는 문제가 발생하였고 이를 해결하기 위해 ReLU 함수를 많이 사용하고 있다.

최적화 알고리즘이란 역전파 알고리즘을 사용하면서 가중치를 갱신하는 규칙을 결정하는 알고리즘이다. SGD, Adadelta, Adam 알고리즘 등이 있다 [15]. 본 연구에서는 성능이 비교적 일반적으로 높다고 알려진 Adam 알고리즘을 사용하였다.

손실함수란 출력오차를 계산하는 함수이며 종류로는 Poisson, BinaryCrossentropy, categorical-cross entropy 등이 있다.

추가로 딥러닝의 세부 연산 층에 대해서 설명하면 아래와 같다.

완전연결층이란 기본적인 딥러닝 신경망을 의미한다.

컨볼루션층란 입력 특징 벡터들에 필터의 개수만큼 컨볼루션 연산을 수행하여 특징을 추출해내는 층이다. 완전연결층에서는 공간정보가 학습 중에 일부 소실되는 문제가 발생하였는데 이를 해결하기 위한 층이다.

풀링층은 특징 벡터들의 차원을 축소하는 층이다. 종류로는 Max pooling layer와 Average pooling layer, Min pooling layer가 있다. 각각 강한 특징이이나 평균적인 특징, 약한 특징들을 추출해내는 특성이 있다.

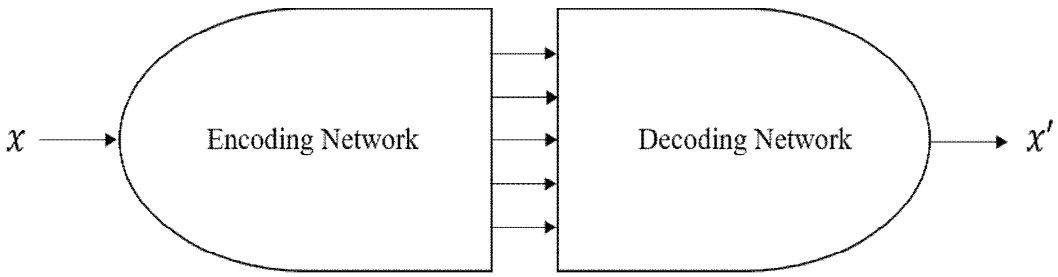
5. AutoEncoder 모델

본 연구에서 다루는 AE는 신경망 모델중 하나이며 지도학습과 비지도학습의 경계에 있는 혼합형 모델이다. AE는 크게 EN(Encoding Network)와 DN(Decoding Network)로 나누어져 있으며, EN는 비지도 학습으로, DN는 지도학습으로 학습된다 [3].

EN가 어떤 입력 값을 받아서, DN로 출력한다. DN는 다시 EN로부터 원래의 입력 값을 예측하는 훈련을 한다.

이 AE의 굉장히 흥미로운 점은 EN를 생성모델로서 사용할 수 있다는 점이다. 이를 이용해서 무선광통신에서의 적합한 새로운 광신호의 특성을 생성해낼 수 있다.

더 나아가 EN와 DN 사이에 통신 채널의 물리층을 넣으면 이 채널에 적합한 신호를 생성하는 학습을 할 수 있다.



[그림 3.4.3 AutoEncoder 구조 개략도]

제4장 광통신 분야 딥러닝 적용 연구

무선 광통신 분야에서 딥러닝 적용 연구가 활발히 진행되어오고 있다. 대표적으로 OOK(On-Off keying) 변조 문제, 컬러 변조 문제, 채널 분류 문제 등에 적용되어 오고 있다 [16].

OOK란 가장 기본적인 디지털 변조 방식중 하나이며, ON과 OFF를 통해서 신호를 보내는 방식이다. 이 변조의 성능을 높이기 위한 방법 중 하나로 딥러닝 연구가 진행되고 있다.

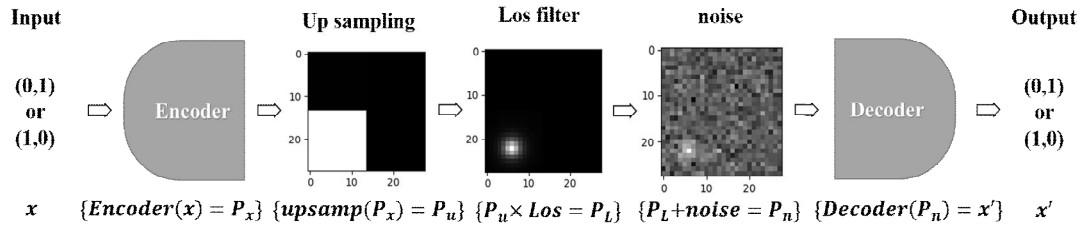
컬러 변조 문제는 필터를 통해서 여러 파장의 빛을 따로 분할하여 인식 할 수 있는 점에 착안한 것인데 이를 딥러닝을 통해서 파장별로 성능을 높이는 방법도 연구되고 있다.

채널 분류 문제의 경우 무선광통신채널의 특성을 딥러닝을 통해서 파악 및 분류하는 연구가 활발히 진행되고 있다.

제5장 무선광통신 시스템의 딥러닝 기반 신호 제안

제1절 무선광통신 신호 제안 시스템 구성

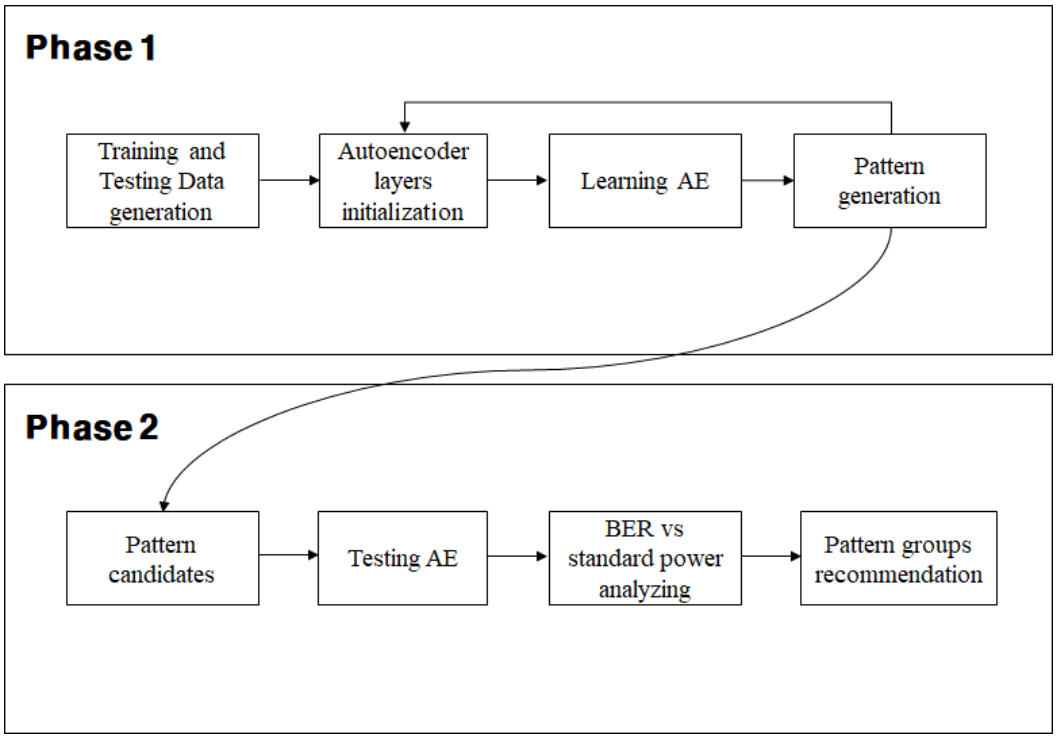
본 연구는 무선 광통신 시뮬레이션에 딥러닝 방식을 적용하여 채널 특성에 맞는 신호를 찾아내어 제안하는 방식을 제안한다. 기본 시스템 구성은 그림 5.1.1와 같이 구성된다.



[그림 5.1.1 딥러닝 기반 무선광통신 신호제안 시스템 기본 구성]

여기서 x는 메시지 데이터의 입력 벡터를 의미한다. 본 연구에서는 기본적인 2진 메시지를 다루었으므로 One-Hot encoding 되어진 $(0,1)$ 과 $(1,0)$ 벡터를 입력으로 넣었다. 훈련이 되지 않은 Encoder에 2진 벡터 x를 입력하면 그 벡터에 해당하는 빛의 신호 강도를 병렬 LED 배열의 개수에 해당하는 2×2의 임의의 행렬 값을 생성한다. 그 생성한 값을 Up sampling 연산 및 LOS 연산을 통해서 수신기에 도달한 빛의 강도를 계산하고 채널 효과인 노이즈를 추가하여 행렬 이미지 데이터가 출력된다. 이후에 이 값들을 입력받은 Decoder가 원래 메시지를 예측하기 위해 x'을 출력한다. 그리고 x와 x'간의 손실함수를 계산하여 역전파 알고리즘을 기반으로 하는 최적화 알고리즘을 통해서 AE를 학습시킨다. 이러한 과정을 반복하면 무선광통신의 신호와 채널효과에 맞게 손실 함수가 최소화되는 병렬 LED 광신호를 Encoder가 생성할 수 있게 되며, Decoder는 원래 메시지를 더욱더 잘 예측하는 가중치 파라미터를 학습하게 된다.

앞선 그림 5.1.1의 기본 구조를 바탕으로 전체적인 구성은 그림 5.1.2와 같다.

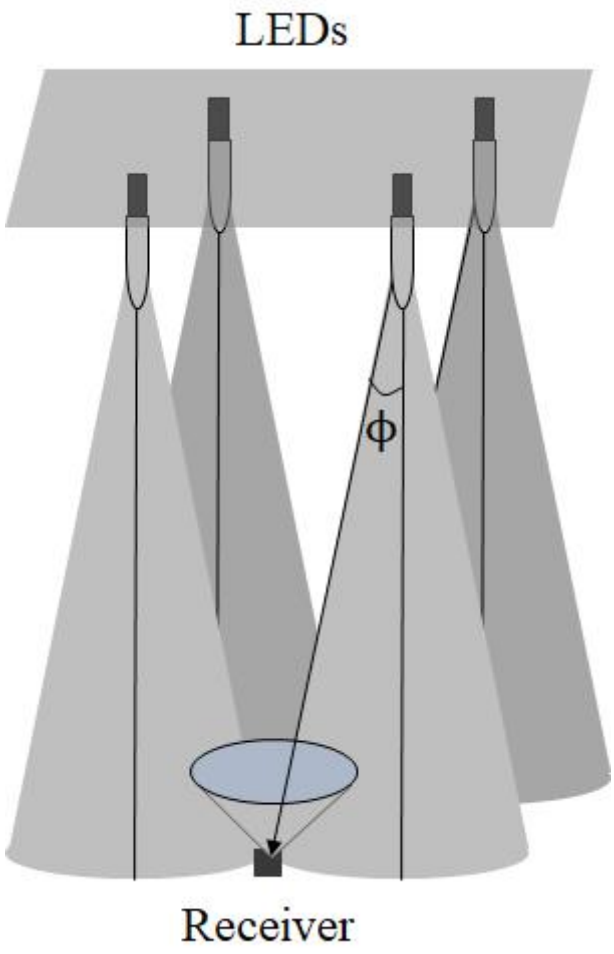


[그림 5.1.2 딥러닝 기반 무선광통신 신호제안 시스템 전체 구성]

우선 훈련 메시지 데이터 및 테스트 데이터를 생성한다. AE의 가중치 값들을 초기화하고 생성한 데이터 기반으로 학습시킨다. 충분히 학습되어 좋은 성능의 LED 신호패턴을 생성할 경우, AE들의 가중치 및 패턴들을 저장하며 다시 AE들의 가중치를 초기화 하고 새로운 AE를 학습시킨다. 이 과정을 계속해서 반복하며 패턴들의 후보군들을 모은다. 이후에 충분히 패턴들의 후보를 모으면 LED의 전력의 강도를 조절하며 빛의 강도에 따른 통신성능을 분석한다. 마지막으로 최종적인 패턴 군들을 추천해준다.

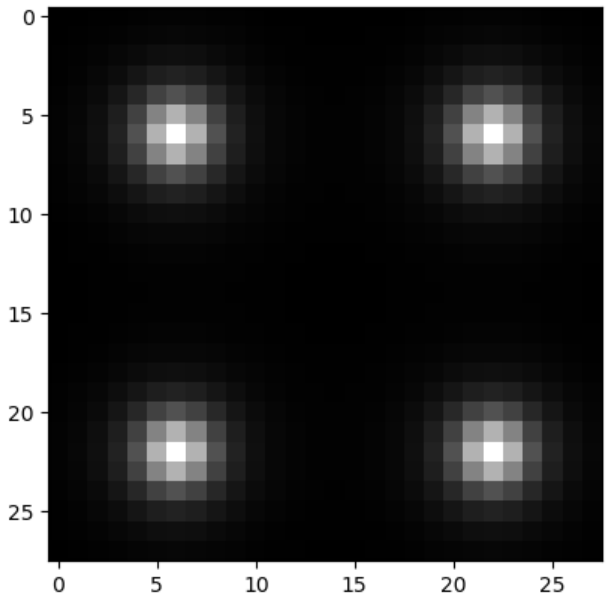
제2절 무선광통신의 물리적인 시뮬레이션 조건 구성

무선 광통신 시뮬레이션의 물리적인 구조는 다음과 같이 구성된다. 2×2의 LED 배열의 병렬 방식의 무선광통신이다. CCD로 구성된 수신기는 28×28(pixels)이다.

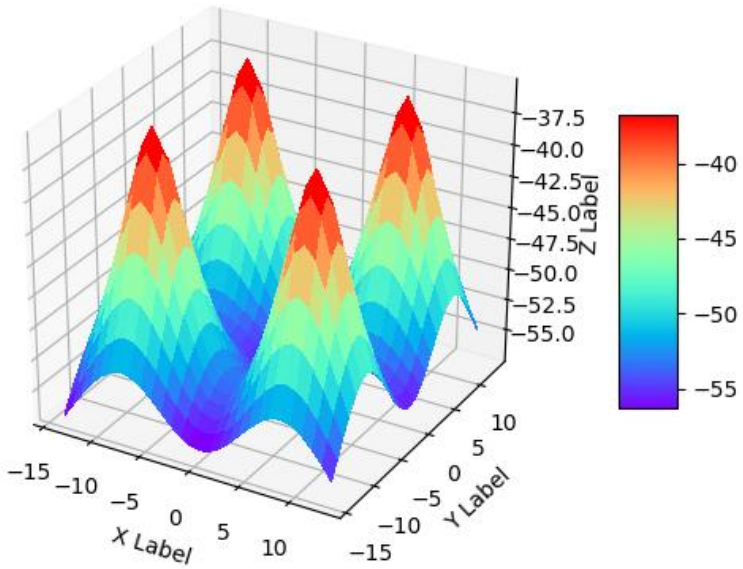


[그림 5.2.1 병렬 LED 배열 무선광통신
시스템의 기하학적 구조]

수신기에 감지되어진 빛의 강도 분포는 LOS모델의 연산을 따라서 그림 5.2.2와 그림 5.2.3의 형태와 같이 나타난다. LED와 수직이 되는 부분에서 빛의 강도가 제일 높고 그 주변으로 빛의 강도가 약해지는 것을 확인 할 수 있다.



[그림 5.2.2 수신기에서 감지된 빛의 강도 분포 1]



[그림 5.2.3 수신기에서 감지된 빛의 강도 분포 2]

제3절 AutoEncoder와 물리채널층의 신경망 구성

본 연구에서 수행되어진 딥러닝의 구조는 다음과 같이 설계되어진다.

	Encoding network		
	Layer	**Output shape**	**Activation**
1st layer	Fully-connected	M × 1	ReLU
2nd layer	Fully-connected	$16L^2$ × 1	ReLU
3th layer	Convolutional (M filters, 3-by-3)	4L × 4L	ReLU
4th layer	Max-pooling (2-by-2)	2L × 2L	-
5th layer	Convolutional (2M filters, 3-by-3)	2L × 2L	ReLU
6th layer	Max-pooling (2-by-2)	L × L	-
7th layer	Convolutional (M filters, 3-by-3)	L × L	ReLU

[표 5.3.1 인코딩 네트워크 구조]

크게 AE와 채널 모델링된 물리 채널층 기반으로 동작한다. EN과 같은 경우 완전 연결층과 컨벌루션층 Max-pooling 층으로 이루어져있으며 입력으로는 One-Hot 인코딩 되어진 M진 메시지의 M차원 벡터를 입력받는다. 본 연구에서는 기본적인 2진 통신을 다루었으므로 2차원 벡터를 입력받는다. 여기서 L이 의미하는 것은 최종 출력하고자하는 LED의 개수를 의미하며 본 연구에서는 2×2 개수의 LED를 사용하였으므로 L은 2가 된다. 차원축소 및 특징추출 시키는 역할만 수행하고 활성화 함수가 필요 없는 Max-pooling 층을 제외하고 나머지 층은 기울기 소멸이 적은 ReLU 함수를 활성화 함수로 선택하였다.

	Physical channel layers		
	Layer	**Output shape**	**Activation**
8th layer	LOS propagation	T × T	-
9th layer	Additive white Gaussian noise	T × T	-

[표 5.3.2 물리 채널층의 구조]

	Decoding network		
	Layer	**Output shape**	**Activation**
10th layer	Convolutional (2M filters, 5-by-5)	T × T	ReLU
11th layer	Max-pooling (2-by-2)	T/2 × T/2	-
12th layer	Convolutional (2M filters, 3-by-3)	T/2 × T/2	ReLU
13th layer	Max-pooling (2-by-2)	T/4 × T/4	-
14th layer	Fully-connected	M × 1	ReLU
15th layer	Fully-connected	M × 1	Softmax

[표 5.3.3 디코딩 네트워크 구조]

여기서 T는 수신기의 CCD의 개수를 의미하며 28×28의 배열의 CCD를 본 연구에서 사용하였으므로 T는 28이 된다.

LED의 출력 2×2 값에 Up-sampling을 하여 28×28로 변환하고 물리 채널층에서는 LED의 빛의 강도에 대한 LOS 곱 연산 및 노이즈 합 연산을 추가로 진행한다.

DN에서는 렌즈방정식을 따르는 배율의 이미지 값을 수신기로 입력받아서 최종적으로 원래 메시지를 예측한다.

제6장 딥러닝 기반 무선광통신 시스템 구현

제1절 실행환경 및 라이브러리 버전

제안한 딥러닝 기반 무선광통신 시스템 시뮬레이션을 실행하기 위해 다음과 같은 환경에서 구동하였다.

컴퓨터 OS 환경은 Window 10에서 구동하였다. 프로그래밍 언어로는 Python 3.6을 사용하였다. 딥러닝을 위한 라이브러리로는 Keras를 사용하였으며, cpu 기반에서 학습시켰다 [17].

구체적인 라이브러리의 버전은 그림 6.1.1과 같다.

Package	Version
Keras	2.1.5
Keras-Applications	1.0.8
Keras-Preprocessing	1.1.0
numpy	1.16.5
openpyxl	3.0.2
matplotlib	3.0.2

[그림 6.1.1 실행 라이브러리 버전]

딥러닝 및 수치연산, 이미지 출력과 엑셀 데이터 정리에 관한 오픈소스 라이브러리 코드는 그림 6.1.2와 그림 6.1.3을 통해 확인 할 수 있다.

```
# keras 라이브러리
from keras.layers import Input, Dense, Reshape, Flatten, Activation, ₩
    BatchNormalization, Dropout, concatenate, Conv2D, MaxPooling2D, UpSampling2D
from keras.models import Model
from keras import backend as K
from keras import optimizers
from keras.layers import Layer, InputSpec
from keras.utils import to_categorical
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
from keras.utils import plot_model
```

[그림 6.1.2 Keras 라이브러리 코드]

```
# 수치연산 라이브러리
import numpy as np
from numpy import array
from numpy import pi
import random

# 그래프 라이브러리
import matplotlib.pyplot as plt
import matplotlib.cm as cm

# 엑셀 라이브러리
import openpyxl
from openpyxl.chart import LineChart,Reference
from openpyxl.drawing.image import Image
```

[그림 6.1.3 기타 라이브러리 코드]

제2절 메인 클래스 코드

딥러닝 기반 무선광통신 시스템 시뮬레이션의 메인 클래스인 owcAuto 클래스에 관한 구조는 그림 6.1.4에서 확인 할 수 있다.

```
128  class owcAuto:
129      def __init__(self,M = 2, gen_number = 2*10**5, training_num = 10**5):...
140
141      def generating_data(self,Pt, no_plist, dataname):...
193
194      def merging_data(self,input_name_list, output_name):...
229
230      def los(self, P_total=1, L=28):...
294
295      # chains neural network layers
296      def encoder(self):...
328
329      def channel(self,x):...
342
343      def decoder(self,x):...
359
360      def fit(self,autoencoder,epoches,batchsize):...
378
379      def fit_history(self):...
391
392      def testing(self,autoencoder):...
399
400      def training_sample(self,autoencoder,Total_epochs,mini_batch_size):...
426
427      def testing_sample(self,autoencoder):...
446
447      def one_input_layer_output_test(self,autoencoder,M,input1,startlayer,endlayer):...
473
474      def two_input_layer_output_test(self,autoencoder,T,input1,input2,startlayer,endlayer):...
502
503      def validate(self,autoencoder):...
```

[그림 6.2.1 메인 클래스 코드 축약]

그림 6.2.2는 메인 클래스의 초기화 메소드를 확인 할 수 있으며 훈련 파라미터를 설정한다.

```
128 class owcAuto:
129     def __init__(self,M = 2, gen_number = 2*10**5, training_num = 10**5):
130         # generates data parameter
131         self.M = M
132         self.gen_number=gen_number
133 
134         self.L = 2
135         self.T = 28
136 
137         self.training_num = training_num
138         self.train_rate = 50
139         self.test_rate = 50
```

[그림 6.2.2 메인 클래스 초기화 메소드]

다음 그림 6.2.3과 그림 6.2.4는 훈련 데이터 생성 및 저장 메소드 코드이다. 또한 물리 채널층에 적용할 LOS 연산 메소드 호출 및 노이즈 데이터를 생성한다.

그림 6.2.5의 코드는 LOS 연산을 수행하는 메소드이다.

```
141     def generating_data(self,Pt, no_plist, dataname):
142         data_list=[]
143         noise_list=[]
144 
145         gen_number = self.gen_number
146         M = self.M
147         T = self.T
148         L = self.L
149         training_num= self.training_num
150         train_rate = self.train_rate
151 
152         ##############################
153         # los
154         P_rec_list = []
155         self.P_rec_total = self.los(Pt, T)
156         ##############################
157 
158         for i in range(0, gen_number):
159             data = random.randint(0,M-1)
160             data_list.append(data)
161 
162             P_rec_list.append(self.P_rec_total)
163 
164             noise = np.random.normal(no_plist[0], no_plist[1], T*T)
165             noise_matrix = noise.reshape([T,T,1])
166             noise_list.append(noise_matrix)
```

[그림 6.2.3 훈련 데이터 생성 메소드1]

```
one_hot = to_categorical(data_list)
total_traing_data = one_hot[0:training_num]
rate = round(training_num*train_rate/100)

self.x_train = total_traing_data[0:rate]
self.x_validation = total_traing_data[rate:training_num]
self.x_test = one_hot[training_num:gen_number]

los_traing_data = P_rec_list[0:training_num]

self.los_train = los_traing_data[0:rate]
self.los_validation = los_traing_data[rate:training_num]
self.los_test = P_rec_list[training_num:gen_number]

#노이즈 샘플 이미지
noise_traing_data = noise_list[0:training_num]

self.noise_train = noise_traing_data[0:rate]
self.noise_validation = noise_traing_data[rate:training_num]
self.noise_test = noise_list[training_num:gen_number]

np.savez(dataname, x_train = self.x_train, x_validation = self.x_validation, x_test = self.x_test,
                   los_train = self.los_train, los_validation = self.los_validation, los_test = self.los_test,
                   noise_train = self.noise_train, noise_validation = self.noise_validation, noise_test = self.noise_test)
```

[그림 6.2.4 훈련 데이터 생성 메소드2]

```
195     def los(self, P_total=1, T=28):
196         theta = 70
197         m1 = -np.log10(2)
198         m2 = np.log10(np.cos(theta * np.pi / 180.))
199         m = m1 / m2
200 
201         Adet = np.array(1e-4)
202         Ts = 1
203         index = 1.5
204         FOV = 60 * pi / 180
205         G_Con = np.square(index) / np.sin(FOV)
206 
207         lx = 28
208         ly = 28
209         lz = 2.15
210         h = 2.15
211 
212         XT = [-8, 8]
213         YT = [-8, 8]
214 
215         Nx = lx * 1
216         Ny = ly * 1
217 
218         x = np.arange(-lx / 2, lx / 2, lx / Nx)
219         y = np.arange(-ly / 2, ly / 2, ly / Ny)
220         XR, YR = np.meshgrid(x, y, sparse=True)
221 
222         P_rec_total = np.zeros((Nx, Ny))
223         Nh = np.full((Nx, Ny), h)
224 
225         for i in XT:
226             for j in YT:
227                 DX = XR - np.full((Nx, Ny), i)
228                 DY = YR - np.full((Nx, Ny), j)
229                 D1 = np.sqrt(DX ** 2 + DY ** 2 + Nh ** 2)
230                 cosphi_A1 = Nh / D1
231                 H_A1 = (m + 1) * Adet * (cosphi_A1 ** (m + 1)) / (2 * pi * (D1 ** 2))
232                 P_rec = P_total * Ts * G_Con * H_A1
233                 P_rec_total = P_rec_total + P_rec
234 
235         P_rec_total = P_rec_total.reshape([T, T, 1])
236         return P_rec_total
```

[그림 6.2.5 LOS 연산 메소드]

그림 6.2.6 코드는 AE의 구성요소인 Encoder 메소드이다.

```
# chains neural network layers
def encoder(self):
    M=self.M
    L=self.L
    T=self.T

    self.input_img = Input(shape=(M,), name = 'orignal_input')  # 0 layer
    x = Dense(M)(self.input_img)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Dropout(0.1)(x)

    x = Dense(16*L*L)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Dropout(0.1)(x)
    x = Reshape((4*L, 4*L, 1))(x)

    x = Conv2D(M, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = MaxPooling2D((2, 2), padding='same')(x)

    x = Conv2D(2*M, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(1, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    self.x = Activation('relu')(x) # 20 layer
```

[그림 6.2.6 Encoder 메소드]

그림 6.2.7은 LOS연산과 노이즈를 추가하는 물리채널층 메소드이며, 그림 6.2.8 코드는 AE의 구성요소인 Decoder 메소드이다.

```
def channel(self,x):
    L = self.L
    T = self.T

    x = UpSampling2D((int(T/L),int(T/L)))(self.x) # 21 layer

    self.los_img = Input(shape=(T,T,1), name ='los_input') # 22 layer
    x = concatenate([x, self.los_img])
    x = mymulLayer((T,T,2))(x)  # 24 layer

    self.noise_img = Input(shape=(T,T,1), name = 'channel_input') # 25 layer
    x = concatenate([x, self.noise_img])
    self.x = mysumLayer((T,T,2))(x) # 27 layer
```

[그림 6.2.7 물리채널층 메소드]

```
def decoder(self,x):
    M = self.M

    x = Conv2D(2*M, (5, 5), padding='same')(x) # 28 layer
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = MaxPooling2D((2, 2))(x)

    x = Conv2D(M, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = MaxPooling2D((2, 2))(x)
    x = Flatten()(x)
    x = Dense(M, activation='relu')(x)
    self.decoded = Dense(M, activation='softmax')(x) # 38 layer
```

[그림 6.2.8 Decoder 메소드]

그림 6.2.9는 딥러닝 네트워크의 전체의 반복 훈련과 테스트 메소드이다.

```
340     def fit(self,autoencoder,epoches,batchsize):
341
342         self.hist = autoencoder.fit({'orignal_input': array(self.x_train),
343                                      'los_input': array(self.los_train),
344                                      'channel_input': array(self.noise_train)},
345                                     array(self.x_train),
346                                     epochs=epoches,
347                                     batch_size=batchsize,
348                                     shuffle=True,
349                                     validation_data=([array(self.x_validation),
350                                                       array(self.los_validation),
351                                                       array(self.noise_validation)],
352                                                      array(self.x_validation)))
353
354     def fit_history(self):
355         plt.subplot(211)
356         plt.plot(self.hist.history['loss'], label="loss")
357         plt.legend()
358         plt.title("loss")
359         plt.subplot(212)
360         plt.plot(self.hist.history['acc'], label="training accuracy")
361         plt.plot(self.hist.history['val_acc'], label="validating accuracy")
362         plt.legend()
363         plt.title("accuracy")
364         plt.tight_layout()
365         plt.show()
366
367     def testing(self,autoencoder):
368         # evaluates network
369         loss, self.acc = autoencoder.evaluate([array(self.x_test),
370                                                array(self.los_test),
371                                                array(self.noise_test)],
372                                               array(self.x_test))
373         print("모델의 정확도: {:5.2f}%".format(100*self.acc))
```

[그림 6.2.9 딥러닝 네트워크 전체의 반복 훈련 및 테스트 메소드]

그림 6.2.10은 딥러닝 전체 네트워크를 통해 어떤 임의의 입력데이터 하나에 대한 출력 데이터를 예측하여 출력하는 메소드이다.

```
402     def testing_sample(self,autoencoder):
403
404         # input, output sample data
405         idx = np.random.random_integers(0, self.gen_number-self.training_num)
406         xtest = array(self.x_test[idx])
407         xtest = xtest.reshape([1, self.M])
408         lostest = array(self.los_test[idx])
409         lostest = np.expand_dims(lostest, 0)
410         noisetest=array(self.noise_test[idx])
411         noisetest=np.expand_dims(noisetest,0)
412
413         y = autoencoder.predict({'orignal_input': xtest,
414                                  'los_input': lostest,
415                                  'channel_input': noisetest})
416         list=[]
417         for i in y[0]:
418             list.append(int(i*100))
419
420         return xtest, lostest, noisetest, list
```

[그림 6.2.10 전체 네트워크 단일 테스트 메소드]

그림 6.2.11은 딥러닝의 은닉층에 해당하는 출력 값을 테스트 할 때 사용하는 메소드이다.

```
def one_input_layer_output_test(self,autoencoder,M,input1,startlayer,endlayer):
    endlayer+=1

    # Before channel effect
    if isinstance(M,int):
        input_img1 = Input(shape=(M,))
    elif isinstance(M,list):
        input_img1 = Input(shape=(M[0],M[1],M[2]))
    else:
        raise print("type error: one_input_layer_output_test function requires type int or list")

    A_layer = input_img1
    for layer in autoencoder.layers[startlayer:endlayer]:
        A_layer = layer(A_layer)

    smatrix = Model(input_img1, A_layer)
    y = smatrix.predict(input1)
    #smatrix.summary()

    return y

def two_input_layer_output_test(self,autoencoder,K,input1,input2,startlayer,endlayer):
    endlayer+=1

    inputC = Input(shape=(K,K,1))
    inputD = Input(shape=(K,K,1))

    f = concatenate([inputC, inputD])
    flayer = f
    for layer in autoencoder.layers[startlayer:endlayer]:
        flayer = layer(flayer)
        #print(flayer)

    fmodel = Model([inputC, inputD], flayer)
    f_data = fmodel.predict([input1, input2])
    #fmodel.summary()

    return f_data
```

[그림 6.2.11 은닉층의 단일 입력 및 이중 입력 테스트 메소드]

제3절 서브 클래스 코드

그림 6.3.1과 그림 6.3.2은 각각 채널 물리층의 연산을 돕기 위해 커스터 마이징한 이중 입력의 합과 곱의 연산층이다.

```
510  class mysumLayer(Layer):
511
512      def __init__(self, units,
513                   axis = -1,
514                   **kwargs):
515          if 'input_shape' not in kwargs and 'input_dim' in kwargs:
516              kwargs['input_shape'] = (kwargs.pop('input_dim'),)
517          super(mysumLayer, self).__init__(**kwargs)
518
519          self.units = units
520          self.axis = axis
521          self.input_spec = InputSpec(min_ndim=2)
522          self.supports_masking = True
523
524      def build(self, input_shape):
525          dim = input_shape[self.axis]
526          if dim is None:
527              raise ValueError('Axis ' + str(self.axis) + ' of '
528                               'input tensor should have a defined dimension '
529                               'but the layer received an input with shape ' +
530                               str(input_shape) + '.')
531
532          self.input_spec = InputSpec(min_ndim=2, axes={-1: dim})
533
534          self.built = True
535          super(mysumLayer, self).build(input_shape)  # 끝에서 꼭 이 함수를 호출하십시오
536
537      def call(self, inputs, **kwargs):
538          input_shape = K.int_shape(inputs)
539          output=inputs[:,:,:,0]+inputs[:,:,:,1]
540          output=K.expand_dims(output,3)
541          return output
542
543      def compute_output_shape(self, input_shape):
544          output_shape=input_shape[:3]+(1,)
545          return output_shape
```

[그림 6.3.1 커스텀 마이징된 합연산 층 클래스]

```
class mymulLayer(Layer):

    def __init__(self, units,
                 axis = -1,
                 **kwargs):
        if 'input_shape' not in kwargs and 'input_dim' in kwargs:
            kwargs['input_shape'] = (kwargs.pop('input_dim'),)
        super(mymulLayer, self).__init__(**kwargs)

        self.units = units
        self.axis = axis
        self.input_spec = InputSpec(min_ndim=2)
        self.supports_masking = True

    def build(self, input_shape):
        dim = input_shape[self.axis]
        if dim is None:
            raise ValueError('Axis ' + str(self.axis) + ' of '
                             'input tensor should have a defined dimension '
                             'but the layer received an input with shape ' +
                             str(input_shape) + '.')

        self.input_spec = InputSpec(min_ndim=2, axes={-1: dim})

        self.built = True
        super(mymulLayer, self).build(input_shape)  # 끝에서 꼭 이 함수를 호출하십시오

    def call(self, inputs, **kwargs):
        input_shape = K.int_shape(inputs)
        output=inputs[:,:,:,0]*inputs[:,:,:,1]
        output=K.expand_dims(output,3)
        return output

    def compute_output_shape(self, input_shape):
        output_shape=input_shape[:3]+(1,)
        return output_shape
```

[그림 6.3.2 커스텀 마이징된 곱연산 층 클래스]

그림 6.3.3과 그림 6.3.4는 데이터를 엑셀 형태로 다루거나 저장하기 위한 클래스이다.

```
class ExcelControl:

    def __init__(self):
        self.wb = openpyxl.Workbook()
        self.sheet = self.wb.active

    def excel_read(self,reading_path,name):
        if os.path.exists(reading_path + name):
            listA=[]
            listB=[]
            self.wb2 = openpyxl.load_workbook(reading_path+name)
            self.sheet2 = self.wb2.active
            number_row = self.sheet2.max_row
            number_column = self.sheet2.max_column
            for row in range(1, number_row+1):
                for col in range(1, number_column + 1):
                    listA.append(self.sheet2.cell(column=col, row=row).value)
                listB.append(listA)
                listA=[]
            return listB
        else:
            raise print("Error: excel read")
```

[그림 6.3.3 엑셀 제어 클래스1]

```
def excel_save(self,name,lists,genrating_path='',n=1,linkcolnumber=0,linktext='same',blanknum=0,delcolnumber=None)
    # n을 추가로 입력하면 n번째 엑셀파일을 만듬, 아니면 중복되지 않게 n+1 생성
    # linkcolnumber은 hyperlink 엑셀열, 디폴트는 하이퍼링크 없음
    self.blanknum=blanknum+1
    self.lists=lists

    for list in self.lists:
        self.sheet.append(list)

        for i in range(1,self.blanknum):
            self.sheet.append([])

    self.linkcolnumber = linkcolnumber - 1
    if self.linkcolnumber == -1:
        pass
    else:
        if linktext == 'same':
            self.excel_hyperlink(self.linkcolnumber,self.linkcolnumber)
        if linktext == 'diff':
            self.excel_hyperlink(self.linkcolnumber,self.linkcolnumber+1)
    if not delcolnumber == None:
        for i in delcolnumber:
            self.sheet.delete_cols(i)

    if n == 1:
        while 1:
            if not os.path.exists(genrating_path + name + '_(%d).xlsx' % n):
                self.wb.save(genrating_path + name + '_(%d).xlsx' % n)
                break
            else:
                n += 1
    else:
        self.wb.save(genrating_path+name+'_(%d).xlsx'%n)
```

[그림 6.3.4 엑셀 제어 클래스2]

제4절 메인 함수 코드

그림 6.4.1은 전체 클래스 코드를 아우르는 메인 함수의 파라미터 설정 부분이다. 이 부분에서 훈련 및 연산 데이터 생성 및 저장과 불러오기, 합치기 메소드를 제어한다.

또한 딥러닝 신경망의 가중치 불러오기, 딥러닝 신경망 훈련시키기 및 테스트하기, 중간 은닉층의 값의 시각화 이미지 출력하기, 패턴 데이터 저장하기 또한 제어한다.

```
def main_auto_encoder(train_dataname):

    load_layername = 'save_layer_80_n.h5'
    save_layername = 'save_layer_80_n.h5'

    genrating_traindata = False
    loading_traindata = True
    merging = False

    weight_load = True
    fit = False
    only_testing = True

    sample_train = False
    sample_output_test = True

    data_save = True

    M_list = [2]
    power_list = [20, 40, 60, 80, 100, 120]
    #power_list = [80]

    #no_plists = [[0.5, 0.05],[0.6, 0.05],[0.7, 0.05],[0.8, 0.05],[0.9, 0.05]]
    no_plists=[[0.9, 0.05]]
```

[그림 6.4.1 메인 함수 코드 파라미터 설정 부분]

그림 6.4.2는 데이터 생성 및 합치기, 불러오기 메소드를 호출하여 연산을 수행한다.

```
total_data_lists = []
for M in M_list:
    oa = owcAuto(M, gen_number=8*10**4, training_num=4*10**4)
    for power in power_list:
        for no_plist in no_plists:

            dataname = 'savez_' + str(no_plist[0]) +'_'+ str(power) + '.npz'
            #layername = 'save_layer_' + str(no_plist[0]) + '.h5' #노이즈 별로 신경망 생성

            if genrating_traindata == True:
                oa.generating_data(power, no_plist, dataname)
            else:
                pass

    if merging == True:
        load_data_name_list=[]
        for power in power_list:
            for no_plist in no_plists:
                load_dataname = 'savez_' + str(no_plist[0]) + '_' + str(power) + '.npz'
                load_data_name_list.append(load_dataname)
        print(load_data_name_list)
        oa.merging_data(load_data_name_list,train_dataname)
    else:
        pass


    if loading_traindata == True:
        load_traindata = np.load(train_dataname)
        oa.x_train = load_traindata['x_train']
        oa.x_validation = load_traindata['x_validation']
        oa.x_test = load_traindata['x_test']
        oa.los_train = load_traindata['los_train']
        oa.los_validation = load_traindata['los_validation']
        oa.los_test = load_traindata['los_test']
        oa.noise_train = load_traindata['noise_train']
        oa.noise_validation = load_traindata['noise_validation']
        oa.noise_test = load_traindata['noise_test']
        load_traindata.close()
```

[그림 6.4.2 훈련데이터 생성 저장, 합치기 및 불러오기 코드]

그림 6.4.3은 무선광통신 시스템 시뮬레이션의 AE 딥러닝 적용에 대한 메소드를 모두 호출하여 동작시키는 역할을 수행한다.

```
oa.encoder()
oa.channel(oa.x)
oa.decoder(oa.x)

autoencoder = Model(inputs=[oa.input_img, oa.los_img, oa.noise_img], outputs=oa.decoded)
plot_model(autoencoder, to_file='model.png', show_shapes=True)

# sets neural network
adam = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
autoencoder.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])

# load weights
if weight_load == True:
    autoencoder.load_weights(load_layername)

autoencoder.summary()
if fit == True:
    oa.fit(autoencoder,epoches=1000, batchsize=1000)
    # oa.fit_history()

    autoencoder.save(save_layername)
    oa.testing(autoencoder)

if only_testing == True:
    oa.testing(autoencoder)
```

[그림 6.4.3 딥러닝 신경망 선언 및 훈련과 테스트 코드]

다음 그림 6.4.4와 그림 6.4.5는 패턴생성부에 해당하는 데이터를 시각화 및 저장하는 부분이다.

```
if sample_train == True:
    oa.training_sample(autoencoder,Total_epochs=2, mini_batch_size=10)
    # oa.validate()

if sample_output_test == True:
    count = 0
    data_lists = []
    while count <= M-1:
        [xtest, lostest, noisetest, result] = oa.testing_sample(autoencoder)
        y = oa.one_input_layer_output_test(autoencoder, M, input1=xtest, startlayer=1, endlayer=20)  # encoding output
        y2 = oa.one_input_layer_output_test(autoencoder, [2,2,1], input1=y, startlayer=21, endlayer=21)
        f_data = oa.two_input_layer_output_test(autoencoder, K = 28, input1 = y2, input2 = lostest, startlayer = 24, endlayer = 24) # los output
        f_data2 = oa.two_input_layer_output_test(autoencoder, K = 28, input1 = f_data, input2 = noisetest, startlayer = 27, endlayer = 27) # output
        p = oa.one_input_layer_output_test(autoencoder, [28,28,1], input1=f_data2, startlayer=28, endlayer=38)  # Decoding output
```

[그림 6.4.4 개별 은닉층의 출력 이미지 시각화 및 테스트와 데이터 저장1]

```
            ydata = y[0, :, :, 0].reshape([1, 4])

            plt.subplot(3, 1, 1)
            ydata2 = y2[0, :, :, 0].reshape([28, 28])
            plt.imshow(ydata2, cmap='gray', interpolation='nearest')

            plt.subplot(3, 1, 2)
            f_data = f_data[0, :, :, 0].reshape([28, 28])
            plt.imshow(f_data, cmap='gray', interpolation='nearest')

            plt.subplot(3, 1, 3)
            f_data2 = f_data2[0, :, :, 0].reshape([28, 28])
            plt.imshow(f_data2, cmap='gray', interpolation='nearest')

            plt.show()


            data1 = np.array([no_plist[0], no_plist[1],
                              xtest[0, 0], xtest[0, 1],
                              ydata[0, 0], ydata[0, 1], ydata[0, 2], ydata[0, 3], oa.acc])
            data1 = list(data1)

            if not data1 in data_lists:
                count += 1
                data_lists.append(data1)

        for data_list in data_lists:
            total_data_lists.append(data_list)

    if data_save == True:
        genrating_path = "C:/Users/chosun/Desktop/ALL_ONE/1_SystemStrategy/2_method/Autoencoder/result2/"
        EC1 = ExcelControl()
        EC1.excel_save('data_save', total_data_lists, genrating_path)

    acc = oa.acc
    return acc
```

[그림 6.4.5 개별 은닉층의 출력 이미지 시각화 및 테스트와 데이터 저장2]

제5절 실행 제어 코드

그림 6.5.1은 메인 함수의 실행 제어 및 AE의 초기화와 반복을 통해, 패턴을 생성하고 패턴들의 성능 결과를 저장하는 코드이다.

```
if __name__ == '__main__':
    #datalist=['savez_0.9_20.npz','savez_0.9_40.npz','savez_0.9_60.npz','savez_0.9_80.npz','savez_0.9_100.
    datalist = ['savez_0.9_80.npz']

    weight_list = []

    acclist=[]

    for data in datalist:
        acc = main_auto_encoder(data)

        word = data.split('_')
        power = int(word[2].replace(".npz",""))
        acclist.append(list([power, acc]))
        print(acclist)

    genrating_path = "C:/Users/chosun/Desktop/ALL_ONE/1_SystemStrategy/2_method/Autoencoder/"
    EC2 = ExcelControl()
    EC2.excel_save('result', acclist, genrating_path)
```

[그림 6.5.1 실행 제어 및 최종 결과 저장 코드]

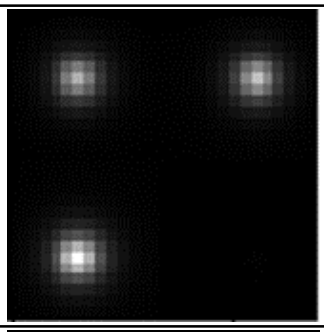
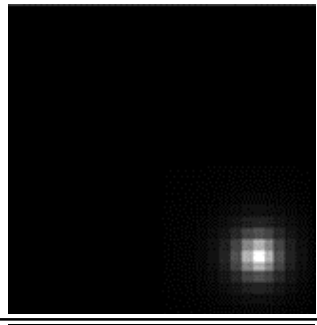
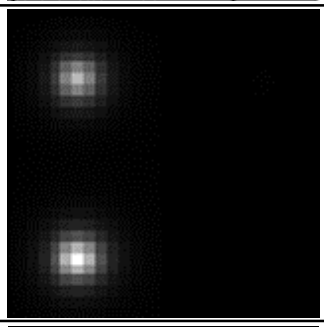
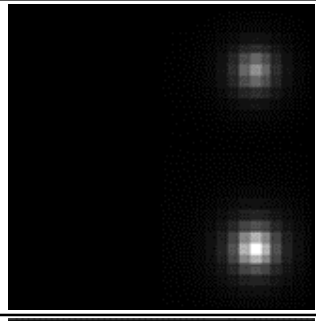
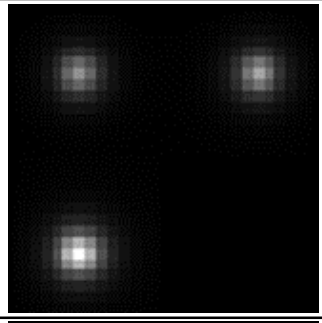
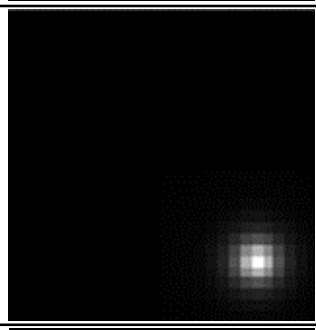
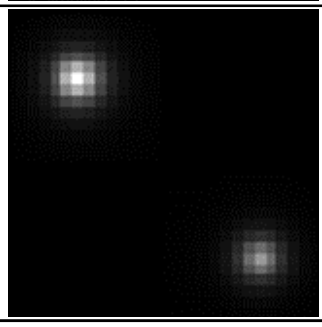
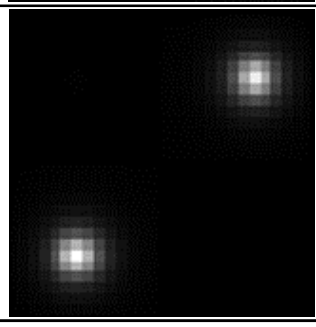
제7장 딥러닝 기반 무선광통신시스템의 통신성능계산

본 연구의 시뮬레이션 학습 결과 AE에서 EN은 표 7.1.1과 같은 메시지 별 패턴들을 생성하였다. 여기서 P1, P2, P3, P4는 각각의 다른 AE로부터 생성된 패턴을 의미한다. M1, M2는 이진 메시지 패턴 쌍을 의미한다.

	M1		M2	
P1	23.34	25.81	0	0
	33.55	0.48	0	47.12
P2	32.75	0.59	0.30	27.95
	45.19	0.21	0.23	49.54
P3	23.86	27.31	0	0
	41.95	0	0	44.29
P4	51.04	0.18	0.52	37.24
	0.10	30.78	39.82	0.15

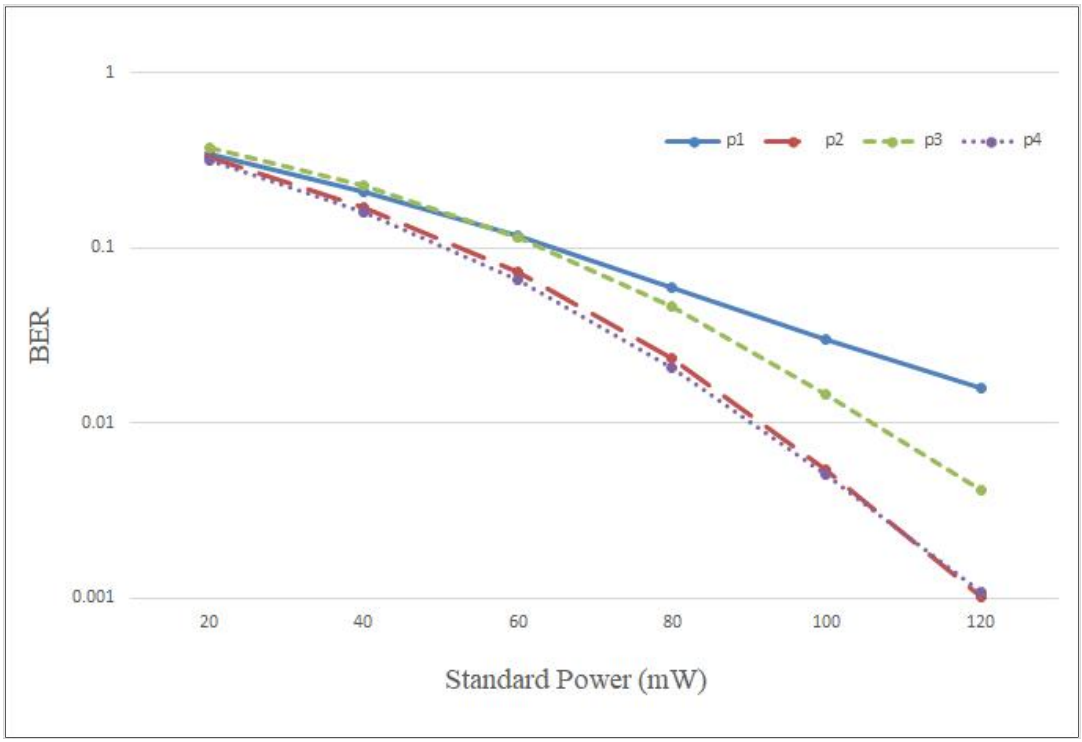
[표 7.1.1 Encoding Network에서 생성된 LED 빛의 강도 패턴표]

표 7.1.1에서 나타나는 수치의 단위는 없으며, LOS연산을 수행 할 때 설정해준 LED의 기준 전력과 곱의 연산을 수행한다. 연산이 수행되어진 패턴은 그림 7.1.1에서 확인할 수 있다.

	M1	M2
P1		
P2		
P3		
P4		

[그림 7.1.1 LOS 연산된 생성 패턴들]

이 패턴들에 대한 기준 전력에 따른 성능의 그래프는 다음 그림 7.1.2와 같이 나타났다. 결과를 분석하면 패턴 P2, P4가 상대적으로 다른 패턴들보다 좋은 성능을 보여주는 것을 확인 할 수 있었다.



[그림 7.1.2 AutoEncoder로 생성된 패턴들의 성능 그래프]

이러한 BER vs 기준 전력의 통신 성능 로그 그래프를 바탕으로 신호 패턴과 채널을 분석하여 실제 물리적인 통신 시스템 성능을 향상시킬 수 있을 것으로 기대되어진다.

제8장 결론

무선광통신 시스템의 성능 개선을 위한, 채널 특성에 적합한 무선광통신 신호를 생성하기 위한 문제 해결을 위한 방법으로 딥러닝 접근법을 적용하였다.

딥러닝 접근법 중에서 AE 모델을 이용하면 채널 특성을 고려한 성능이 좋은 LED 무선광신호 패턴을 생성 할 수 있었다.

또한 생성한 신호들 중에서 성능을 상대 평가하여 통신시스템의 성능을 개선하는데 기여하였다.

더 나아가 다른 물리 통신채널 특성을 넣어주거나, 광변조 특성을 새로이 추가하여 채널에 적합한 또 다른 패턴들을 생성하여 무선광통신시스템 성능에 기여할 것으로 보인다.

참고문헌

[1] H. Elgala, R. Mesleh, and H. Haas, “Indoor optical wireless communication: Potential a nd state-of-the-art,” IEEE Commun. Mag., vol. 49, no. 9, pp. 56–62, Sep. 2011.

[2] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MI T Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[3] A. Ng. (2010). Sparse autoencoder. [Online]. Available: https://web.stanford.edu/class/cs2 94a/sparseAutoencoder.pdf

[4] G. Duree., Optics for Dummies, Wiley, NJ: Hoboken, USA, pp. 36, 2011.

[5] G. Duree., Optics for Dummies, Wiley, NJ: Hoboken, USA, pp. 45-56, 2011.

[6] Z. Ghassemlooy, W. Popoola, and S. Rajbhandari, Optical Wireless Communications: Sy stem and Channel Modelling with MATLAB. Boca Raton, FL, USA: CRC Press, 2013.

[7] D. Singh, Fundamentals of Optics, PHI Learning, 2015.

[8] F. Chollet et al. (2015). Keras. [Online]. Available: https://keras.io

[9] 한학용, 패턴인식 개론, 한빛미디어 (주). pp. 31-32, 2005.

[10] N. Cristianini and J. Taylor. An Introduction to Support Vector Machines and Other K ernel-based Learning Methods. Cambridge UP, 2000.

[11] D. Whitley, “A genetic algorithm tutorial,” Statist. Comput., vol. 4, pp. 65–85, 1994.

[12] S. S. Haykin and S. Haykin, Neural Networks and Learning Machines. New York, N Y, USA: Prentice-Hall, 2009.

[13] 한학용, 패턴인식 개론, 한빛미디어 (주). pp. 318-330, 2005.

[14] R. Hecht-Nielsen, “Theory of the backpropagation neural network,” in Proc. Int. Joint Conf. IEEE Neural Netw., Washington, DC, USA, 1989, pp. 593 – 605.

[15] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” in Proc. I nt. Conf. Learn. Represent., 2015, pp. 1 – 41.

[16] H. Lee, S. H. Lee, T. Q. S. Quek, and I. Lee, “Deep learning framework for wireless systems: Applications to optical wireless communications,” IEEE Commun. Mag., vol. 5 7, no. 3, pp. 35 – 41, Mar. 201

[17] https://keras.io/getting_started/intro_to_keras_for_engineers/