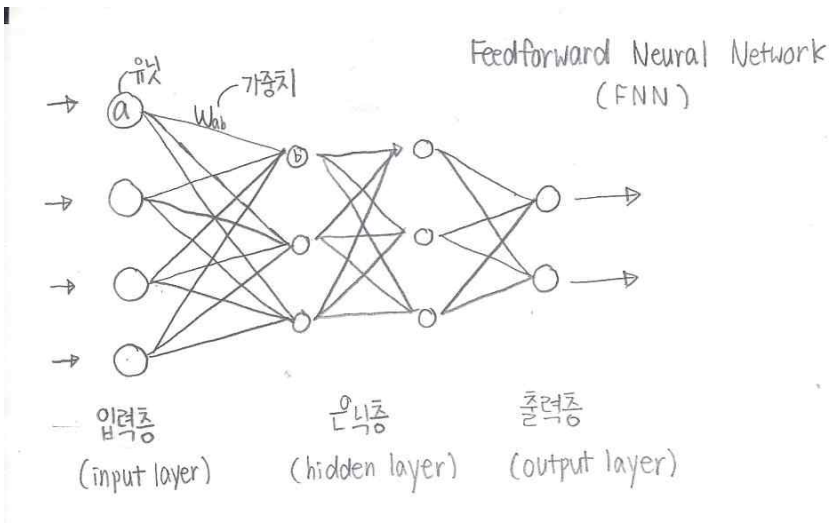
2017 강서학생탐구발표대회 탐구보고서

인공 신경망의 원리와 성능 개선 방법	출품번호
	미기재

출품 부문	공학 및 에너지



2017. 05. 07.

소 속 청	학 교 명	학 년	성 명
서울	염경중학교	2	강준서
서울	염경중학교	2	이유진

목 차

Ⅰ. 서론

ⅰ. 개요

바야흐로 인공지능의 시대이다. 인공지능은 우리가 채 알아채기도 전에 우리 생활에 녹아 들어왔다. 아침에 일어나서 새벽에 잠이 들기까지, 우리의 생활은 인공지능과 함께한다. 우리가 자주 사용하는 SNS인 페이스북이나 트위터의 경우에도 사용자의 정보를 기반으로 사용자가 관심을 가질 것이라 예상되는 글과 사진을 뉴스피드의 상단에 올려 추천하며, 휴대폰의 사용을 편하게 해주는 가상 비서 프로그램도 우리 생활에 녹아든 인공지능의 한 예시이다.

이런 인공지능의 특징 중 하나는, 우리가 생각하는 '계산 기계'로서의 컴퓨터, 한 치의 오차도 용납하지 않는 컴퓨터와는 다르게 '퍼지적으로' 사고하고 작동한다는 것이다. 1965년 미국의 L.A 자데 박사가 창시한 '퍼지(Fuzzy; 희미한)'의 개념은 0과 1로만 나누는 양비론적인 디지털 사고가 아닌 어느 것도 아닌, 애매한 정보를 처리하는 데에 중점을 둔다. 그렇다면, 0과 1로만 상황을 판단하는 컴퓨터가 어떻게 인간의 사고방식과 유사하게 작동하고 사고할 수 있는 것일까?

이를 가능케 하는 것이, 1950년대에 로센블라트 교수에 의해 제시된 퍼셉트론 이론이 발전된 형태인 '인공 신경망'이다. 이는 인간의 신경 구조를 본 따 만들어진 이론이었으나, 1900년대 중후반의 인공지능 연구의 침체기와 '연결주의론의 몰락'이라는 사건, 그리고 당시 컴퓨터 성능의 한계로 크게 발전하지 못하였다. 하지만, 현대 인공신경망 기술을 이끌고 있는 제프리 힌턴 교수가 발표한 논문을 시발점으로 인공 신경망은 크게 발전하기 시작했고, 21세기의 시작을 당시 세계 체스 챔피언 게리 카스파로프와 IBM의 인공지능 딥 블루의 체스 대결, 그리고 딥 블루의 승리로 장식하며 인공 신경망과 딥 러닝의 부흥을 세계에 알렸다.

그리고 현재, 실리콘밸리 뿐 아니라 전 세계의 핫 이슈는 바로 '딥 러닝'이다. 딥 러닝 또한 인공 신경망 기술에서 파생된 연구 분야로, 인공 신경망의 층(Layer)을 깊게(Deep) 쌓은 형태를 칭한다. 딥 러닝 또한 여러 문제를 이유로 한동안 발전하지 못하였으나, 21세기 초, 여러 성능 개선 및 문제 해결 이론의 개발과 컴퓨터 성능의 대폭적인 향상과 더불어 크게 발전하고 있다.

본 탐구에서는 위와 같은 인공 신경망-딥러닝의 문제 해결 이론을 탐구하고, 얀 르쿤 교수의 이미지 분류 학습용 숫자 데이터세트[1] MNIST를 사용하여 여러 인공 신경망과 그 성능을 개선하는 알고리즘을 평가하여 보겠다.

ⅱ. 탐구의 동기

알파고와 이세돌의 대국은 수많은 사람들을, 정치가들을, 기업가들을 **인공지능**이라는 가능성에 희망을 품게 만들었다. 〈원피스〉의 명대사에 비유하자면, 세상은, 대 인공지능 시대를 맞았다. '4차 산업 혁명' 이라는 키워드만 들어가도 서점은 대대적으로 홍보해주고, 기업들은 AI 관련 상품들을 마구 내놓기 시작했다. 우리는 단순히 인공지능의 사회적 파급효과만이 아닌 그 원리와 역사, 그리고 나아가 어떻게 하면 그 성능을 향상시킬 수 있을까에 의문점을 가졌고, 탐구해보고자 하였다. 본 탐구에서는 인공지능의 원리, 뇌 구조와의 유사성, 그리고 그 성능 향상 방안을 탐구하고, 실제로 구현하여 그 성능 개선 척도를 측정하도록 한다.

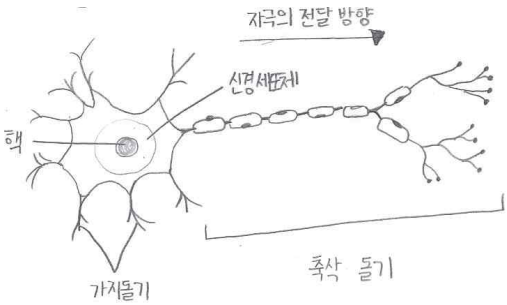
iii. 선행 연구 1 : 인공지능의 역사

개요에서도 이미 언급한 바가 있듯이 인공지능은 점점 인간의 능력을 뛰어넘는 수준으로 빠르게 발전하고 있다. 알파고와 이세돌 9단의 대국으로 인해 얼마 전 우리나라에서도 인공지능의 비약적인 발전이 증명되었다. 컴퓨터는 코딩에 따라 어떠한 작업을 처리하는 기계이지만 인공지능은 많은 데이터를 가지고 스스로 결과를 도출하는 시스템을 가지고 있다. 인공지능의 역사에 대해서 짧게 설명하자면 처음으로 인공지능(artificial intelligence)이라는 용어가 생겨난 때가 1956년이다. 하지만 1974년부터 1980년까지 인공지능은 빙하기를 맞이하게 되는데, 그 이유는 바로 스스로 생각할 수 있는 기계를 프로그래밍 하는 것이 생각보다 어려운 작업인 것으로 판단되어 지원마저 끊겼기 때문이다. 하지만 1980년대 신경망 이론이 등장하면서 인공지능이 다시 주목을 받게 되었다. 여기서 신경망 이론이란 인간의 사고를 담당하는 두뇌 작용을 모방하는 메커니즘 을 고안해낸다면 생각하는 기계를 만들 수 있다는 생각에서 출발한 이론이다. 하지만, 아직 뇌와 같이 많은 데이터를 관리하는 것이 불가능했기 때문에 다시 침체기를 맞았지만 1990년대에 인터넷이 빠르게 발달하면서 재조명되었다. 검색 엔진이 등장하면서 많은 데이터를 수집하고 관리하는 것이 가능해져 이는 기계학습, 즉 머신러닝(machine learning)으로 발전하게 되었다. 또한, 역전파법의 개발로 신경망의 효율적인 학습이 가능하게 되며, 데이터를 스스로 분석하고 학습하는 것이 가능해진 것이다. 1997년에는 IBM의 컴퓨터 딥 블루가 체스 세계챔피언을 상대로 승리하게 되고 2011년에는 IBM의 왓슨이 퀴즈 쇼 "Jeopardy!"에서 두 명의 경쟁자를 이기고 승리하게 된다. 더 나아가 데이터의 분류를 통해 예측을 가능하게 한 딥 러닝의 등장으로 구글 이나 유튜브 등 다양한 기업들이 인공지능을 적극적으로 활용하며 가없이 발전을 하고 있는 인공지능이다.

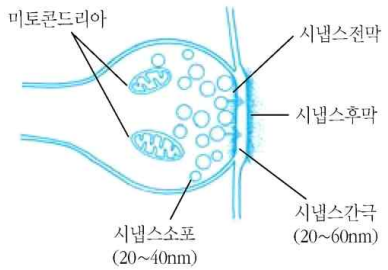
iv. 선행 연구 2 : 인공 신경망의 작동 메커니즘

1) Neural Network

사람의 뇌는 무수히 많은 뉴런과 이를 연결하는 시냅스로 이루어져 있다.

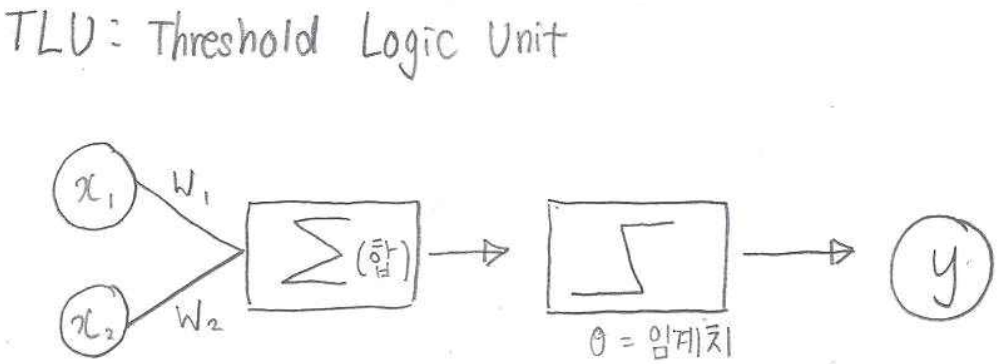


뉴런은 가지 돌기와 핵을 포함하는 신경 세포체, 축삭 돌기로 이루어져 있는데, 축삭 돌기 말단에서 전기적인 신호를 보내고, 이가 시냅스를 통해 다른 신경세포의 가지 돌기를 자극하는 형식으로 신경끼리의 연결이 이루어진다. 여기서 시냅스는 신경 세포 사이를 잇는 부분을 칭한다.

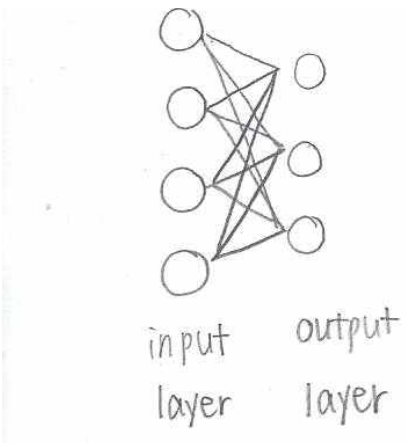


보통 태아의 뇌는 완성이 안 된, 마치 백지의 회로 같은 모습이다. 이 시냅스는 외부의 자극을 통해 신경 간 연결을 강화하거나 약화시켜 뇌의 구조를 변경할 수 있게 하여 인간이 유연한 학습을 할 수 있게 한다.

인공 신경망은 이런 뇌의 구조를 본 따 만들어진 알고리즘[2]이다. 그 기원은 퍼셉트론 이론, 외부 자극을 받아 가중치를 계산하고 출력하는 알고리즘에서 출발한다. 아래는 초기 퍼셉트론 알고리즘의 산물인 TLU[3]의 도식이다.



이 TLU를 여러 겹 쌓은 구조에서 인공 신경망은 유래되었다. 즉, 인간의 뇌 구조를 본 딴 유닛들의 연결 구조를 통해 사람에 가까운 사고를 할 수 있게끔 한 알고리즘이다.

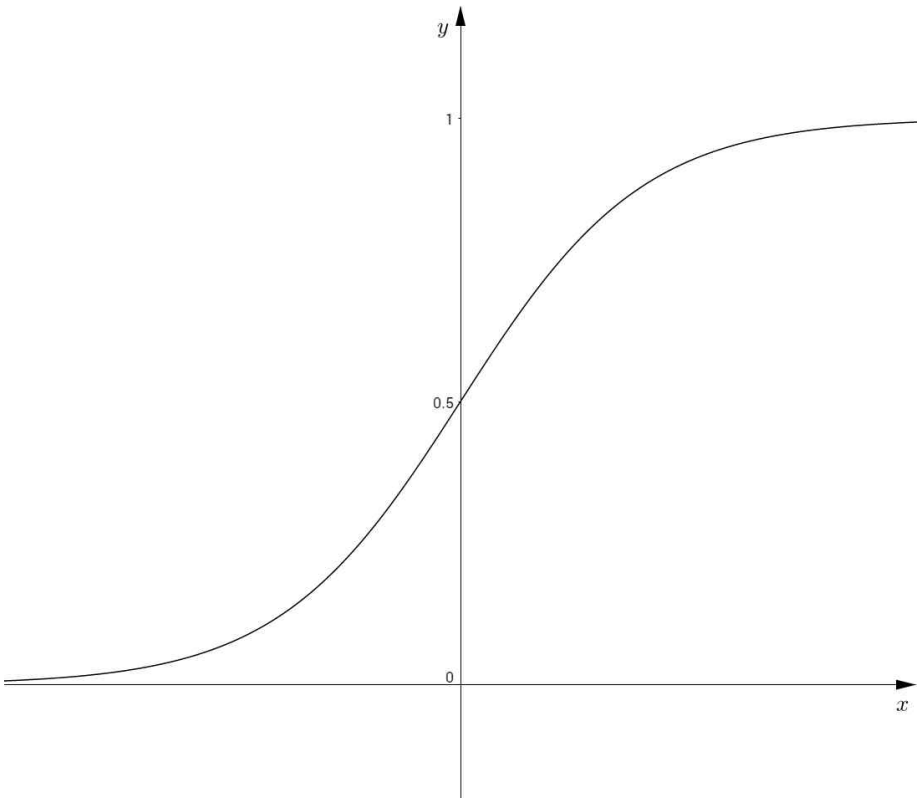


위의 그림에서, 신경세포 하나에 대응되는 유닛(Unit)들은 다음 층(Layer)[4]의 유닛들과 연결되며 각 연결마다 각각의 가중치를 갖는다. 또한, 축삭 돌기에서 발산하는 신호는 일정 수준 이상이 되어야 한다는 점을 본 따, 신경망에 들어오는 입력에도 **활성화 함수(Activate Function)**이라 불리는 함수를 적용한다. 이는 입력이 일정 수준 이상일 때 출력으로 값을 넘기게 하는 역할을 한다. 활성화 함수에 대해서는 아래에서 더 자세히 다루도록 하겠다. 유닛으로 들어온 입력이 활성화 함수를 거치면, 각 연결의 가중치[5]와 곱해져서 다음 계층으로 넘어가게 된다(Bias라 불리는 편향을 추가하는 경우도 있지만, 본 탐구에서는 논외로 두도록 한다). 다음 계층에서는, 각 유닛으로 들어오는 각각의 신호들을 전부 합하고, 이에 다시 활성화 함수를 적용하고 신호를 출력한다. 이것이 간단한 단층 신경망(Single Layered Neural Network)이다. 이 계층을 늘릴 수 있는데, 이때 처음 신호가 들어오는 입력층(Input Layer)에는 관례적으로 활성화 함수를 적용하지 않는다. 계층을 늘린 단층 신경망을 MLP 혹은 단순히 다층 신경망이라고 부른다. 입력층과 은닉층 사이에 있는 계층들은 은닉층(Hidden Layer)이라고 칭하며, **순전파**라고 불리는 유닛으로 들어온 입력이 활성화 함수를 거쳐 다음 계층으로 값을 출력하는 위의 과정은 MLP에서도 동일하게 적용된다(=Feed-Foward)

2) Activate Function

활성화 함수는 여러 가지가 존재하고, 그 역할에 따라 세분화되어 있다. 어떠한 레이어에 어떠한 활성화 함수를 적용하느냐 또한 신경망을 구축할 때 중요한 피쳐(Feature)이다.

- Logistic Sigmoid



$$f(x) = \frac{1}{1+e^{-x}}$$

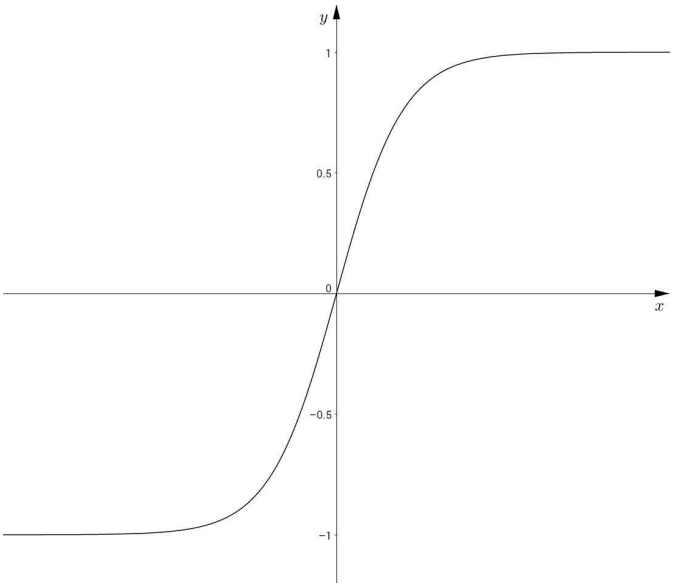
로지스틱 시그모이드, 주로 시그모이드 함수로 불리는 이 함수는 가장 많이 사용되는 활성화 함수이다. 0~1의 치역을 가지며, x=0일 때 y=0.5의 값을 갖는다. 0에서 멀어질수록 기울기가 0에 가까워지며, -1<x<1의 구간에서 매끄럽게 상승하는 곡선 그래프를 보인다.

- Softmax

$$y_k \equiv \frac{sigm(y_k)}{\sum_{i=1}^{K} sigm(y_i)}$$

소프트맥스는 주로 분류 문제의 출력층에 사용되는 활성화 함수이다. 이 함수는 특이하게 인자를 하나 더 받는데, 해당 층의 모든 가중합 벡터[6]를 받는다. 이를 바탕으로 비례 배분하여 출력을 내게 되는데, 이 때문에 출력의 총합이 언제나 1이라는 특징을 갖는다.

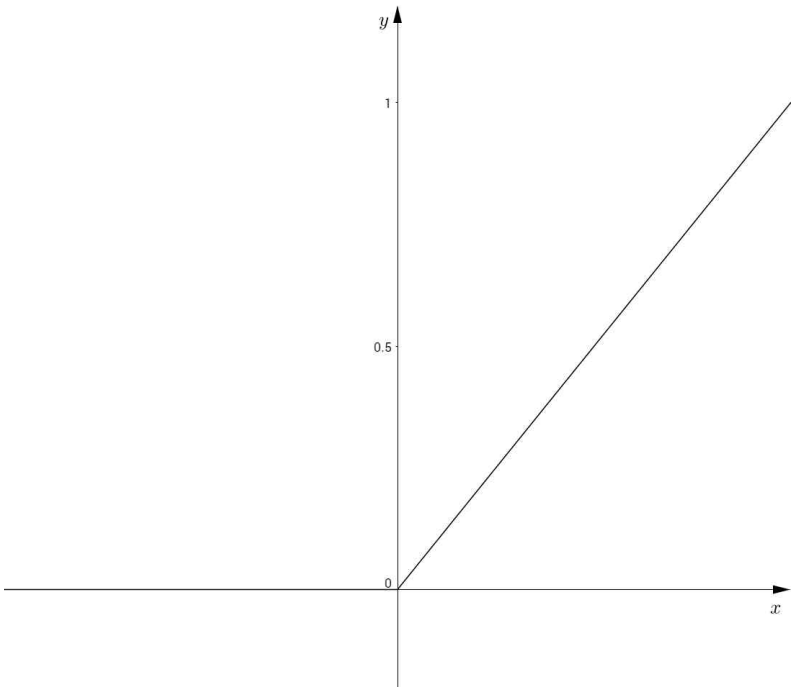
- tanh



$$f(x) = \tanh(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x})$$

tanh 함수는 시그모이드 함수와 같이 0에서 멀어질수록 기울기가 0에 가까워지고, 특정 구간에서 매끄럽게 상승하는 곡선 그래프를 보인다. 다만 치역이 {y|−1=〈y=〈1}이라는 점에서 차이가 난다.

- ReLU(Rectifier Linear Unit)



$$f(x) = \max(x, 0)$$

ReLu 함수는 최근 학계에서 가장 뜨거운 관심을 받고 있는 활성화 함수이다. 시그모이드 함수는 0에서 멀어질수록 기울기가 0에 가까워지는 단점을 갖는다, 이 때문에 신경망의 학습이 제대로 되지 않는 기울기 소실(Gradient Vanishing)의 문제가 발생하는데, 이를 해결할 수 있고, 또한 층의 개수가 많은 신경망에서 계산이 효율적이라는 결과가 발표되면서 최근 동향은 ReLU를 주로 사용하는 쪽으로 넘어가고 있다.

3) Gradient Descent, Backpropagation

뇌의 각 뉴런을 연결하는 시냅스는 외부 자극에 의해서 그 연결을 강화하거나 약화시킨다. 즉, 우리는 살아가면서 외부의 자극을 통해 우리 뇌를 학습시키는 것이다. 인공 신경망도 각각의 유닛(뉴런)을 연결하는 가중치의 크기를 조절함으로서 원하는 기능을 수행하게끔 자신을 학습시킨다.

가중치를 학습시키려면 신경망이 수행한 연산 결과와 실제 예상 값과의 오차가 가장 작아야 한다. 따라서, 이 오차를 최소화시키는 가중치 행렬을 구하는 것이 신경망의 목표이고, 이 과정을 '학습' 또는 '훈련' 이라고 한다.

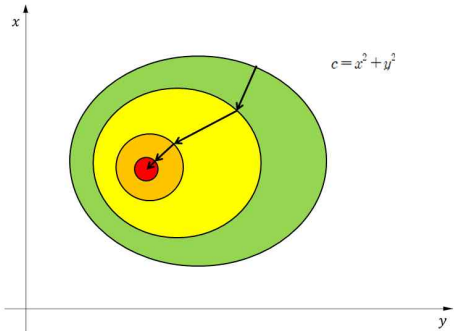
이 때 오차를 구하는 방법은 여러 방법이 있는데, 주로 최소제곱 오차나 교차 엔트로피 오차를 사용한다. 본 탐구에서는 최소제곱 오차만을 통해 신경망을 학습시키기에, 앞으로의 설명에서는 최소제곱 오차만을 예로 설명하겠다. 최소제곱법에 대한 오차함수는 다음과 같다.

$$E(u) = (t_u - o_u)^2 ... u\text{층의오차}$$

즉, 목표값과 실제 출력값의 차를 제곱한 값을 오차로 설정한다. 이를 최소화하는 것이 학습의 주된 목표이다. 목표값은 상수이고, 실제 출력값은 가중치에 의해 결정되기에, 오차함수는 수많은 가중치들에 의해 결정되고, 결국 오차함수의 그래프를 그리면 n차원 이차곡면이 될 것이다. 결론적으로, 이 n차원 그래프의 전역 최솟값의 파라미터를 찾는다면, 신경망의 오차를 최소화하는 가중치 행렬을 구할 수 있다. 즉, 오차함수의 전역 최솟값(의 좌표, 파라미터)을 찾는 것이 신경망 학습의 과정이다.

현실적으로, 이를 미분해서 전역 최솟값을 찾는 방법은 불가능하다(혹은 매우 어렵다) 우선, 컴퓨터는 미분과 같은 계산엔 구조상 능하지 않다. 또한, 이 방법 또한 파라미터가 매우 많아지면-즉 유닛 간 연결이 많아지면 불가능해진다. 가중치를 줄이면 신경망의 자유도 또한 덩달아 낮아지기에, 미분을 하기 위해 가중치를 줄이는 것은 매우 좋지 않은 선택이다. 다시 본론으로 돌아와서, 그렇다면 최적의 가중치 행렬을 구하기 위해선 어떠한 방법을 선택해야 할까, 이에 대해선 몇 가지의 방법이 제시되어 있지만(뉴턴법 등) 본 탐구에서는 이 중에서 가장 대표적이고, 주류를 차지하는 경사감소법(Gradient Descent Method)에 대해서만 설명하고, 사용하도록 하겠다.

경사감소법은 최솟값으로 '점진적으로' 다가가는 최적화 기법으로, 수렴 과정이 경사(기울기)를 감소시키는 방향으로 수렴하기 때문에 붙여진 명칭이다. 가장 기본적인 예시를 들면, 이차함수 $y = x^2$의 최솟값을 찾는 과정은 아래 그림과 같이 일어난다.



골짜기 위에 공이 있다고 생각을 해보자, 이 때 이 골짜기의 가장 낮은 곳으로 향하는 방법은 간단하다. 내리막길 방향으로만 움직이면 언젠간 가장 낮은 곳에 도달할 수 있다. 경사감소법 또한 이와 같은 방법을 사용하여 최솟값을 찾아낸다. 이를 식으로 증명하는 과정은 다음과 같다.

$c = a^2 + b^2$일 때, c의 변화량은 아래와 같다.

$$\triangle c = \frac{\partial c}{\partial a}\triangle a + \frac{\partial c}{\partial b}\triangle b..(1)$$ 또한 이 식은 아래와 같이 나타낼 수 있다.

$$\triangle c = (\frac{\partial c}{\partial a}, \frac{\partial c}{\partial b})^T.(\triangle a, \triangle b)..(2)$$

$\nabla c = (\frac{\partial c}{\partial a}, \frac{\partial c}{\partial b})^T, \triangle v = (\triangle a, \triangle b)$이라고 한다면

$\triangle c = \nabla c.\triangle v$이 된다.

우리가 구해야 할 것은 c의 변화량을 언제나 음수로 만드는 v의 변화량이다. 이는 아래와 같다.

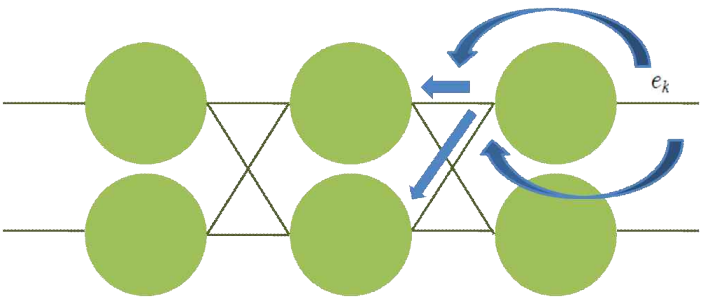
$$\triangle v = -\alpha \nabla c..(3)$$

$$\because \triangle c = -\alpha \|\nabla c\|^2, \|\nabla c\|^2 > 0$$

이로서 c의 변화량을 언제나 음수로 만드는 v의 변화량을 찾을 수 있었다. 즉, 이를 오차함수에 대해 적용하면 오차함수를 감소시키는 파라미터(가중치)의 변화량을 구할 수 있다

신경망에 대해서는 단순히 경사감소법만을 적용할 수 없다. 신경망의 순전파를 거치는 과정이 매우 많이 일어나기 때문에 식을 구하기 어렵고, 이를 미분하기도 힘들기 때문이다. 따라서, **역전파**(Backpropagation)이라 불리는 기법을 사용한다. 역전파는 유닛에 연결된 가중치에 비례해서 오차를 배분하고, 이 오차를 최소화하게끔 경사감소법을 적용하는 기법으로, 역전파법의 개발은 인공지능의 발전을 촉진하는 계기가 되었다(인공지능의 역사 참조)

역전파법은 유닛에 연결된 노드에 따라 오차를 가중치에 비례해서 배분한다. 이는 노드의 가중치에 따라 유닛의 출력에 기여하는 정도가 결정되기 때문이다



(그림 : 오차의 역전파)

배분되는 오차는 아래와 같이 나타낼 수 있다(유닛 j에 연결된 노드)

$$\frac{w_{ij}}{\sum_{i=0}^{M} w_{ij}} * e_j$$

또한 뒤 계층의 유닛에 연결된 노드가 한 개 이상일 경우 역전파된 오차를 재조합하여 그 유닛의 오차를 계산한다. 즉, (k-1)레이어에 있는 유닛 m이 (k)레이어에 존재하는 유닛 j에 연결된 노드와 유닛 k에 연결된 노드가 있는 경우, 각각을 역전파하여 계산한 오차를 다시 합하여 유닛 m의 오차를 구할 수 있다.

컴퓨터 자원을 효율적으로 활용하기 위하여 이를 벡터화하면[7], 아래와 같이 나타낼 수 있을 것이다.

$$error_k = \begin{bmatrix} \frac{w_{11}}{w_{11}+w_{12}} & \frac{w_{12}}{w_{12}+w_{22}} \\ \frac{w_{21}}{w_{21}+w_{11}} & \frac{w_{22}}{w_{22}+w_{12}} \end{bmatrix} \bullet \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$$

(은닉층, 출력층의 유닛이 각각 2개 있는 경우)

하지만 이런 연산은 너무 복잡하다. 만약 여기서 정규화 인자 역할을 하는 분모 부분을 제외해도 오차는 일정할 것이다.

$$e_k = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \bullet \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$$

이 연산의 가중치 벡터 부분을 보면 순전파시 곱해지는 가중치 벡터의 전치행렬임을 알 수 있다.

$$e_k = W^T \bullet e_o$$

따라서 k층의 오차 e_k는 다음 계층의 오차 e_o에 의해 역전파되어 위와 같이 나타낼 수 있다.

역전파를 통해 구한 오차 e는 경사 하강법을 통해 가중치 업데이트의 재료로서 사용된다. 출력층부터 한 단계씩 오차 역전파를 통한 가중치 업데이트를 알아본다.

$$e_{output} = (t_k - o_k)^2$$

최소제곱 오차를 통한 출력층의 오차는 다음과 같다. 이를 미분하여 얻은 Nabla 값은 아래와 같다.
(Logistic Sigmoid 함수의 미분은 sigm(x)(1-sigm(x))로 이루어진다)

$$\frac{\partial e}{\partial w_{jk}} = \frac{\partial e}{\partial o_k} \bullet \frac{\partial o_k}{\partial w_{jk}} = -2(t_k - o_k) \bullet sigm(\sum_{j=0}^{M} w_{jk} \bullet o_j)(1 - sigm(\sum_{j=0}^{M} w_{jk} \bullet o_j) \bullet o_j$$

여기서 2는 그저 상수일 뿐이므로, 소거하고 앞의 과정해서 유도한 가중치 변화량의 식에 대입하면 아래와 같이 가중치의 변화량을 구할 수 있다.

$$\triangle w_{jk} = \alpha \times (e_k * sigm(o_k)(1 - sigm(o_k)) \bullet o_j^T)$$

이를 통해 점진적으로 신경망을 최적화하는 가중치 행렬에 접근할 수 있다.

v. 선행 연구 3 : 두뇌의 시각 피질의 구조와 작동 메커니즘

시각피질이란 후두엽에서 시각적인 데이터 처리에 관여하는 영역이다. 우리의 시각과 관련된 뇌의 영역은 크게 배측계와 복측계로 나누어진다. 배측계는 두정엽에 정보를 보내고, 보고 있는 것이 공간적으로 어디에 위치하는지를 파악한다. 복측계는 하측두엽에 정보를 보내고, 보고 있는 물체가 정확히 무엇인지를 파악한다.

현재 뇌과학자들은 뇌의 각 영역에 그 영역의 기능과 관련된 데이터를 표시함으로써 뇌에서 일어나는 활동을 일종의 지도로 표현하고 있다. 이러한 뇌지도의 작성 이유는 현재까지의 연구결과를 간단하고 보기 쉽게 시각적으로 정리할 수 있기 때문이다.

뇌지도가 그려지기 시작한 것은 20세기부터였다. 1957년경에 작성된 뇌지도를 보면 뇌의 각 영역에 어떤 활동에 관여하는지, 그리고 후두엽이 손상된 사람들에게 시각적으로 문제가 생기는 것을 발견하고는 그 영역에서 발병가능한 병은 무엇인지 등을 기록해놓았다. 본격적으로 뇌에 지도를 그리게 된 것은 CT(computed tomography), PET(positron emission tomography)와 MRI(magnetic resonance imaging)이 개발된 뒤이다. 이러한 검사를 통해 사람들은 뇌 내부의 피질이 서로에게 어떻게, 그리고 어떠한 영향을 끼치는지를 연구했고, 따라서 현재 인지신경과학자들의 뜨거운 관심을 받고 있으며 새로운 정보가 계속해서 업데이트 되고 있는 분야가 바로 시각 연구이다.

PET와 MRI를 통한 시각 인지 연구에서 실험 대상자들의 멈춘 물건과 움직이는 물건을 봤을 때의 뇌 혈류량을 찍어보니 모든 대상자들이 움직이는 물건을 보았을 때 후두엽의 뇌 피질이 활성화됨을 알 수 있었다. 이런 식으로 어떠한 시각적 정보가 입력이 되었을 때 뇌의 어느 부분이 활성화되는지를 알아보면, 뇌지도가 부분부분 채워지는 것이다. 최근에는 한국인의 표준 뇌지도도 완성되어서 여러 가지 질병의 조기진단도 가능해졌다. 정확하고 세부적인 뇌지도가 완성된다면 인공 뇌를 만들 수 있게 되고 최고의 인공지능도 개발할 수 있게된다.

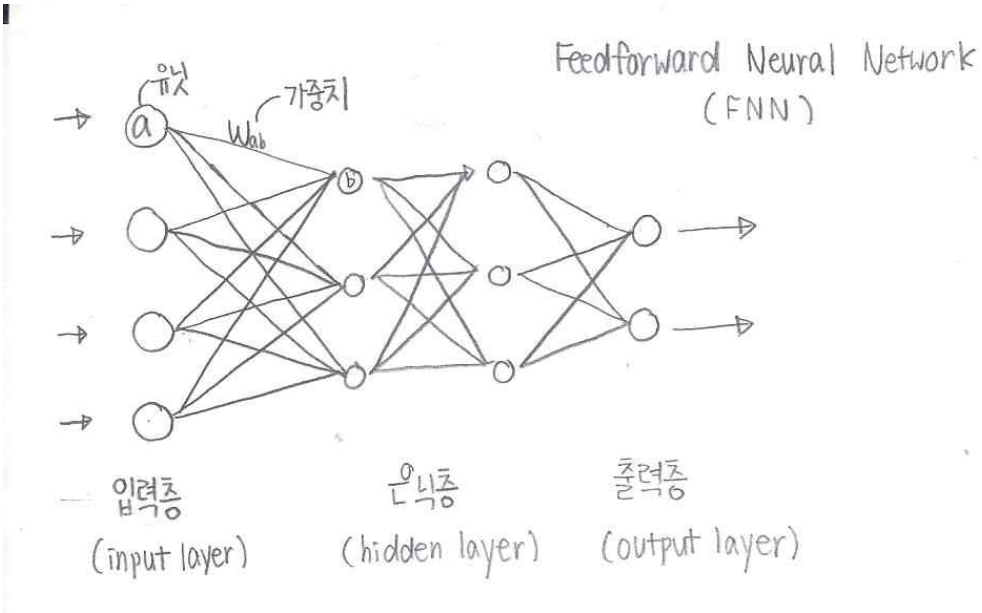
vi. 탐구 계획 및 목적

본 탐구의 목적은 여러 인공 신경망 **성능 개선 방안을 비교하고 실제로 구현**하는 데에 있다.

그러하기 위해, 우선 인공 신경망의 여러 메소드들을 탐구하고 그 원리를 파악하고, Python을 통해 실제로 인공 신경망을 구현한 다음, 탐구한 메소드들을 실제 신경망에 적용하여 그 성능을 비교한다.

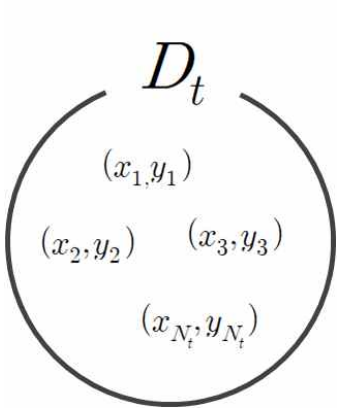
Ⅱ. 본론 : 인공 신경망의 성능 개선 방안

ⅰ. Feedfoward Neural Network의 성능 개선 방안



피드포워드 뉴럴넷은 앞먹임 신경망이라고도 불리며, 가장 기본적인 형태의 신경망이다, 기본적으로 모든 유닛이 연결된 형태를 가지나, 상황에 따라서 일부 연결을 끊는 경우 또한 있다.

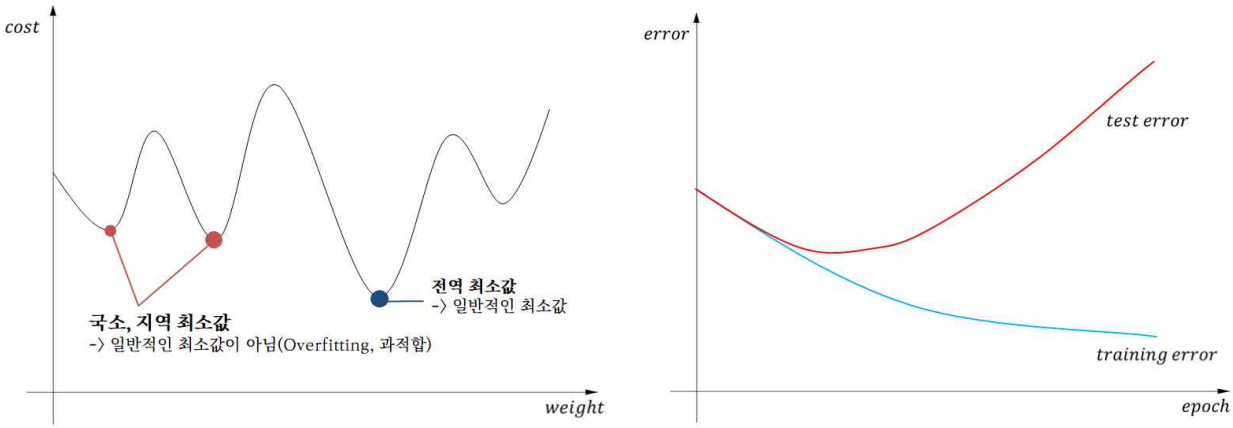
1) 속도 개선 : Minibatch



$$E_t(w) = \frac{1}{N_t} \sum_{n \in D_t} E_n(w)$$

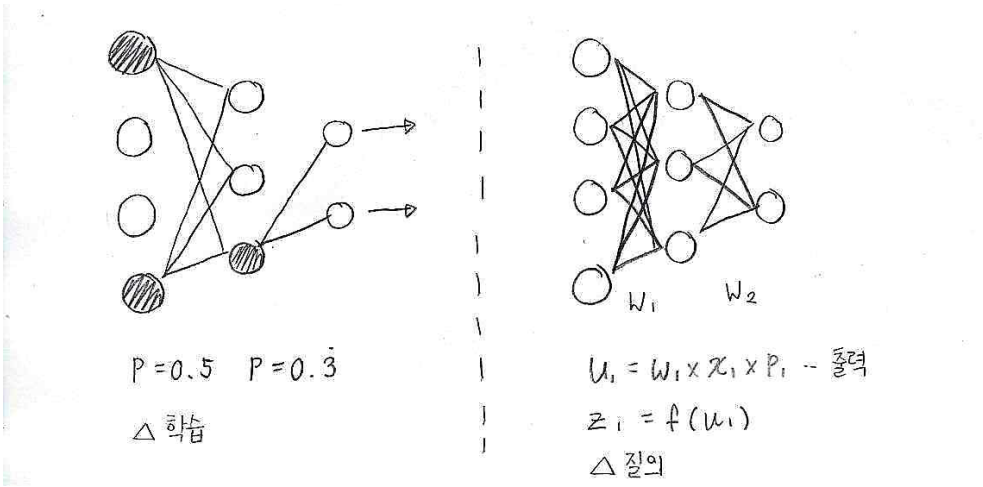
Minibatch는 SGD[8]의 학습 과정에서 오차 계산을 위해 몇 개의 샘플을 하나의 작은 집합으로 묶어 학습을 진행한다. 이 때 이 작은 집합을 Minibatch라 부르고, 이 Minibatch를 통해 가중치를 업데이트하게 된다. Minibatch를 통한 SGD의 학습을 수행하면 컴퓨터의 병렬 연산을 잘 활용할 수 있다. 그 여파로, 학습 속도 또한 빨라지게 된다. 이 때, Minibatch에 들어가는 샘플 수 n을 결정하는 것이 효율적인 학습의 열쇠이다(주로 10~100 사이를 사용한다. 이보다 작으면 행렬 연산의 이점을 활용하기 힘들고, 너무 높으면 SGD의 이점을 활용하기 힘들어지기 때문이다)

2) 과적합 완화



과적합은 학습용 데이터세트에 대해 계산한 결과의 오차에 비해 학습용 데이터세트가 아닌 테스트용 데이터세트, 즉, 일반화된 데이터에 대한 오차가 낮게 나오는 현상을 칭하며 이는 파라미터(예를 들어, 신경망의 가중치나 선형 회귀분석에서의 계수와 바이어스)가 경사 감소의 과정에서 지역 최솟값에 갇히는 문제로 인해 일어난다. 과적합은 주로 신경망의 자유도가 높은-파라미터가 많을 때 자주 일어나며, 주기-epoch를 높게 설정할 때 자주 일어난다. 이를 완화하기 위해 자유도를 낮추는 방법을 사용하나, 신경망의 문제 해결 능력 또한 자유도가 높을수록 높아지기에 과다한 자유도를 낮추거나 이를 제한하는 방법을 사용하여 과적합을 완화한다.

2-1) Dropout



Dropout은 과적합을 해결하기 위해 발명된 기법 중 하나로, 신경망 학습 시의 자유도를 강제적으로 낮춰 과적합을 완화시킨다. Dropout은 학습과 질의의 과정에 영향을 줌으로서 신경망의 자유도를 낮추는데, 학습 과정에서 각 레이어마다 p의 비율을 갖게 일부 유닛을 무효화하고 신경망을 학습시킨다(1>p>0). 이는 한 유닛이 선택될 확률 또한 p가 되는 것임을 의미한다. 다시 질의의 과정에서는 학습 시 유닛의 개수가 p배가 된 것을 보상하기 위해 각 레이어의 출력값에 p를 곱해 출력한다. 즉, 학습 시 유닛의 개수와 질의 시의 유닛의 개수비를 맞추기 위한 과정이다. 이러한 드롭아웃의 과정-유닛을 무효화시키는-을 통해 신경망의 파라미터는 p의 비율로 줄어들게 되고, 자연스레 신경망의 자유도가 낮아져 과적합 또한 완화된다.

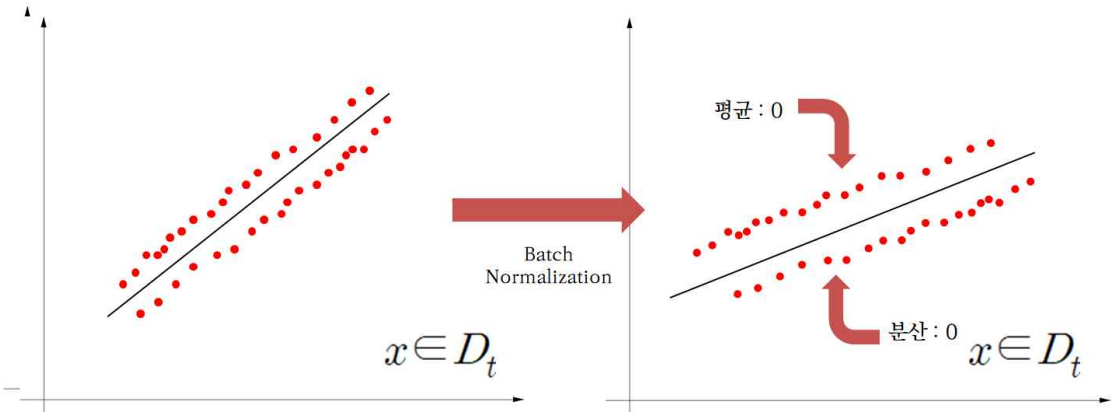
2-2) 가중치 감쇠

Dropout이 과도한 가중치를 낮추는 방식을 채택하였다면, 가중치 감쇠는 가중치에 제약을-매우 커지게 하지 않는-방식을 취하여 과적합을 완화한다.

$$E_{감쇠}(w) = E(w) + \frac{\gamma}{2}\|w\|^2$$

즉, 가중치 감쇠는 원래의 오차함수에 가중치의 제곱합과 $\gamma/2$의 곱을-감쇠의 정도를 결정한다, 일반적으로 10^-2~10^-6의 범위에서 선택된다- 더해 이를 통해 학습시키는 방법이다. 즉, 가중치는 그 크기가 커질수록 줄어드는 정도도 커지고, 결론적으로 가중치가 지속적으로 감쇠하게 되고, 이를 통해 신경망의 자유도를 제약할 수 있다.

3) 배치 정규화 : Batch Normalization



활성화 함수의 결과값을 활성화값이라 한다(u). Sigmoid 함수의 경우 입력값이 0에서 멀어질수록 기울기가 0에 수렴하기 때문에 학습 능력이 제한된다(Gradient Vanishment). 또한, 활성화 값이 특정 값에 몰려 있는 경우, 신경망이 표현할 수 있는 범위가 제한된다. 이를 예방하기 위해 강제로 활성화값을 분포시키는데, 이를 배치 정규화라 한다.

배치 정규화는 출력층 이전에(주로 ReLu 활성화함수를 사용) 배치 정규화층, 어파인 층을 추가하여 이루어지는데, 배치 정규화층 에서는 데이터세트의 미니배치 단위로 정규화가 이루어진다.

$$avg_D = \frac{1}{n}\sum_{i=0}^{n} x_i \ldots 미니배치 D의 평균$$

$$var_B^2 = \frac{1}{n}\sum_{i=0}^{n}(x_i - avg_D)^2 \ldots 미니배치 D의 제곱합$$

$$x_{i_n} = \frac{x_i - avg_D}{\sqrt{var_D^2 + \epsilon}} \ldots 정규화식$$

(엡실론 : Zero Division Error을 방지하기 위해 더하는 아주 작은 값)

즉, 미니배치 하나마다 평균을 0으로 갖고, 분산이 1인 정규분포를 갖게끔 데이터를 정규화[9]시킴으로서, 활성화값이 몰리는 것을 방지하여 과적합, 기울기 소실과 포화를 방지한다. 또한 정규화를 수행한 후에 확대, 이동의 선형 변환을 수행하고, 이 수치는 학습을 통해 적절히 수정한다.

4) 데이터 확장 : Data Augmentation

학습 데이터가 부족하면 과적합에 빠지기 쉽고, 정확도 또한 떨어진다. 그러나 많은 양의 학습 데이터를 취하기는 여러 문제가 있고, 이를 해결하기 위해 데이터를 가공해 새로운 레코드[10]를 만들어내는 데이터 확장을 수행한다. 이 가공에는 노이즈, 회전, 블러링 등이 있다.

ii. 성능 개선 평가 : 벤치마크

https://github.com/InvalidId404/Neural_Network/blob/master/Neural_Network.py : Code

탐구한 성능 개선 메서드를 평가하기 위해, Python을 이용하여 메서드들을 실제로 구현하고, 개선되는 정도를 평가한다. 하드웨어 조건은 같게 한다. 성능 개선 방안 중, 학습 속도와 과적합 완화에 기여하는 대표적인 3가지 방안만을 테스트한다.

0) MNIST DataSet 준비

MNIST는 Yann Lcun 교수의 인공 신경망 학습용 Public Dataset으로, 27*27 픽셀의 0부터 9까지의 그레이스케일 이미지와[11] 레이블로 이루어진 CSV 파일이다. Training용과 Test용으로 나누어져 있고, Training은 6만 개의 레코드를 포함하고 있으며, Test는 1만 장의 레코드를 포함하고 있다. 트레이닝용은 신경망의 학습에 사용되고, 테스트용은 신경망의 평가에 활용된다. 테스트 데이터세트에 대한 오차를 일반화 오차라 부른다.

MNIST는 한 픽셀의 alpha 값을 0~255 사이의 정수로 나타내는데, 우리가 기억해야 할 점은 시그모이드 함수의 치역이 0<y<1이라는 것이다. 시그모이드 함수는 지나치게 큰 입력을 받으면 포화해버리기에, 이 데이터를 먼저 신경망에 알맞게 가공해야 한다.

① 입력값을 신경망에 알맞게 조정하기

레이블은 그 이미지의 값을 정수로 나타낸다. 신경망의 출력은 정수가 아닌 출력층 유닛 개수만큼의 길이를 가지는 열벡터[12]이므로, 이와 같게끔 레이블을 가공해 줘야 한다. 아래의 코드는 레이블을 n번째 원소만 0.99의 값을 가지고 나머지는 0.01의 값을 가지게 가공한다. 이는 시그모이드 함수의 치역이 1보다 작고 0보다 큰 실수이기 때문이다. 이와 같은 맥락으로, 입력되는 값인 이미지의 픽셀도 27*27의 길이를 갖는 열벡터로, 또 시그모이드 함수와 동일한 치역을 갖게끔 보정한다.

```
def process_data(data):
            result = []
            for record in data:
              record = record.split(',')
              result.append([int(record[0]), [int(i)/256+0.01for i in record[1:]]])
              result[-1][0] = [int(result[-1][0] is i)*0.98+0.01for i inrange(10)]
            return result

def max_index(iterable):
              max_d =max(iterable)
              return iterable.index(max_d)
```

② 데이터세트 불러오고 가공하기

```
import neuralnetwork
import os
import matplotlib.pyplot

      def process_data(data):
        result = []
        for record in data:
          record = record.split(',')
          result.append([int(record[0]), [int(i)/256+0.01for i in record[1:]]])
```

```
            result[-1][0] = [int(result[-1][0] is i)*0.98+0.01for i inrange(10)]
        return result
    def max_index(iterable):
      max_d =max(iterable)
      return iterable.index(max_d)

    os.chdir(os.getcwd()+'\Data_set\MNIST')
    train_file =open('mnist_train.csv')
    train_data = train_file.readlines()
    train_file.close()
    test_file =open('mnist_test.csv')
    test_data = test_file.readlines()
    test_file.close()
    train_data = process_data(train_data)
    test_data = process_data(test_data)
```

이로서 MNIST 데이터세트를 불러오고, 신경망을 테스트할 준비가 끝났다. 이제 신경망의 코드를 작성하고, 그 성능을 테스트하도록 한다.

1) Feedfoward Neural Network 클래스 작성

https://github.com/InvalidId404/InvalidId-AI/blob/master/neuralnetwork.py : Code

탐구한 피드포워드 뉴럴넷의 성능 개선 메서드를 적용하기 위해 먼저 가장 기본적인 피드포워드 뉴럴넷을 구현하는 클래스를 작성한다.

① 부모 클래스 정의

```
class NeuralNetwork:
        activate_function = {
          'Logistic_Sigmoid': lambda x: 1/(1+np.e**(-x))
        }
```

앞으로 작성할 모든 신경망의 부모 클래스가 되는 클래스를 정의한다. 이 클래스는 공통적으로 사용하는 활성화 함수 등을 작성하는 클래스이다. 현재는 로지스틱 시그모이드 함수만 작성하였다.

```
class FeedFowardNeuralNetwork(NeuralNetwork):
                def __init__(self, nodes):
                                                    .
                                                    .
                                                    .
```

지금은 위에서 정의한 클래스를 상속받는 자식 클래스인 FeedfowardNeuralNetwork를 작성하겠다.

② 생성자 작성

```
def __init__(self, nodes):
    '''
     :keyword 생성자
     :param nodes: 노드 개수(List)
    '''
    self.nodes = nodes # 계층별 노드 개수 - List
    self.classes =len(nodes) # 계층 수
```

```
        self.weight = [
          np.random.normal(
            0.0, self.nodes[i]**(-1/2), (self.nodes[i+1], self.nodes[i])
         ) for i inrange(self.classes-1)
        ] # 가중치 초기화
        self.dataset = [] # 학습 데이터세트, (0 : 목표 신호, 1 : 입력 신호)
```

생성자는 클래스의 객체가 생성될 때 호출되는 함수이다. 이 클래스의 생성자는 각 계층의 노드(유닛) 개수를 입력받아 신경망의 뼈대를 생성하고 초기 가중치를 설정한다. 이 가중치는 Xavier Initialization[13]을 사용하여 초기화된다.

③ 질의 메서드 작성

```
def query(self, input):
    '''
     :keyword 순전파 메서드
     :param input: 입력 신호
     :return: 결과값 리스트(0~n)
    '''
    result = [] # 출력값
    input_data = np.array(input) # 입력 데이터 - 열벡터
    for i inrange(self.classes):
    if i is 0: # 초기값
      result.append(input_data)
    else: # 순전파
      result.append(
        self.activate_function['Logistic_Sigmoid'](self.weight[i-1]@result[-1])
      )
    return result
```

신경망에게 입력 신호를 입력하고 각 계층의 결과값을 반환하는 메서드를 정의한다. 인자로 입력한 데이터는 Numpy의 행렬 형식으로 변환되고, 행벡터에서 열벡터로 전치된다. 이는 계산의 편의성을 위한 전처리 과정이다. 그리고, 신경망의 값이 (p-1) 레이어에서 (p)레이어로 이동하는 것을 '순전파'라고 하는데, 이 순전파의 과정은 아래 식을 따른다(u는 해당 레이어에서 최종적으로 출력되는 값)

$$u_{jk} = w_{jk}^{T} \bullet o_{j} ...(1)$$

$$o_{k} = f(u_{jk}) ..(2)$$

④ 확률적 경사감소 메서드 작성(SGD)

```
def descent(self, input, target, lr, weight_defined=None):
    '''
     :keyword 확률적 경사감소 메서드
     :param target: 목표 신호
     :param lr 학습률
     :param weight_defined 사용자 지정 가중치
     :return: 새로운 가중치 행렬
    '''
    input_data = np.array([input]).T
    outputs =self.query(input_data)
    target_data = np.array([target]).T
```

```
    weight = weight_defined if weight_defined elseself.weight[:]
    for i in range(self.classes-1):
    if i is 0:
      error = target_data-outputs[-1]
    else:
      error = weight[-i].T@prev_error
    weight[-i-1] += lr * (
      error*outputs[-i-1]*(1.0-outputs[-i-1])@outputs[-i-2].T
    )
    prev_error = error
    return weight
```

확률적 경사감소법은 레코드(데이터세트의 샘플 하나) 하나만을 이용해 가중치를 업데이트 하는 경사감소법의 아종으로, 과적합을 예방하기 위해 사용된다. 이 때, 최소값에 수렴하는 경로는 지그재그 모양을 띈다, 이 덕분에 파라미터가 국소 최소값에 다가가더라도, 탈출할 수 있게 된다.

위의 메서드는 입력 신호와 목표 신호, 학습률을 이용해 가중치를 업데이트한다. 레코드 하나에 업데이트 되는 가중치의 변화량은 아래와 같이 구할 수 있다. (시그모이드 함수의 미분으로)

$$\triangle w_{jk} = \alpha(e_{jk} \times o_{jk} \times (1 - o_{jk}) \bullet o_{lj}^{T})$$

⑤ 훈련 메서드 작성

```
def train(self, epoch, learning_rate, weight_defined=None, dataset_defined=None):
      '''
       :keyword 학습 메서드
       :param epoch: 주기
       :param learning_rate: 학습률
       :param weight_defined 사용자 지정 가중치
       :param dataset_defined 사용자 지정 데이터세트
       :return: 학습된 가중치 행렬
      '''
      weight = weight_defined if weight_defined else self.weight[:]
      dataset = dataset_defined if dataset_defined else self.dataset[:]
      for _ in range(epoch):
          for record in dataset:
           weight = self.descent(
              input=record[1],
              target=record[0],
              lr=learning_rate,
              weight_defined=weight
           )
      return weight
```

작성한 확률적 경사감소법(SGD) 메서드를 실제 신경망의 학습에 사용하는 메서드를 작성한다. SGD는 기본적으로 레코드 하나에 대한 가중치 변화만을 계산하기 때문에, 따로 학습 메서드를 작성해서 데이터세트의 모든 데이터에 대해 SGD를 수행한다. 이 훈련 메서드를 통해 오차함수가 전역 최솟값으로 수렴하는 과정은 상술했듯 지그재그 모양을 띄며 일어난다.

⑥ 클래스 전문

```
class FeedFowardNeuralNetwork(NeuralNetwork):
      def__init__(self, nodes):
      '''
       :keyword 생성자
       :param nodes: 노드 개수(List)
      '''
      self.nodes = nodes # 계층별 노드 개수 - List
      self.classes =len(nodes) # 계층 수
      self.weight = [
      np.random.normal(
      0.0, self.nodes[i]**(-1/2), (self.nodes[i+1], self.nodes[i])
       ) for i inrange(self.classes-1)
       ] # 가중치 초기화
      self.dataset = [] # 학습 데이터세트, (0 : 목표 신호, 1 : 입력 신호)

  def query(self, input):
      '''
       :keyword 순전파 메서드
       :param input: 입력 신호
       :return: 결과값 리스트(0~n)
      '''
      result = [] # 출력값
      input_data = np.array(input) # 입력 데이터 - 열벡터
      for i in range(self.classes):
        if i is 0: # 초기값
          result.append(input_data)
        else: # 순전파
          result.append(
            self.activate_function['Logistic_Sigmoid'](self.weight[i-1]@result[-1])
          )
      return result

  def descent(self, input, target, lr, weight_defined=None):
      '''
       :keyword 확률적 경사감소 메서드
       :param target: 목표 신호
       :param lr 학습률
       :param weight_defined 사용자 지정 가중치
       :return: 새로운 가중치 행렬
      '''
      input_data = np.array([input]).T
      outputs =self.query(input_data)
      target_data = np.array([target]).T
      weight = weight_defined if weight_defined elseself.weight[:]
      for i in range(self.classes-1):
        if i is0:
          error = target_data-outputs[-1]
        else:
          error = weight[-i].T@prev_error
```

```
        weight[-i-1] += lr * (
          error*outputs[-i-1]*(1.0-outputs[-i-1])@outputs[-i-2].T
        )
        prev_error = error
        return weight

    def train(self, epoch, learning_rate, weight_defined=None, dataset_defined=None):
        '''
         :keyword 학습 메서드
         :param epoch: 주기
         :param learning_rate: 학습률
         :param weight_defined 사용자 지정 가중치
         :param dataset_defined 사용자 지정 데이터세트
         :return: 학습된 가중치 행렬
        '''
        weight = weight_defined if weight_defined elseself.weight[:]
        dataset = dataset_defined if dataset_defined elseself.dataset[:]
        for _ in range(epoch):
          for record in dataset:
            weight =self.descent(
              input=record[1],
              target=record[0],
              lr=learning_rate,
              weight_defined=weight
            )
        return weight
```

⑦ 테스트

```
nodes = [
  len(train_data[0][1]),
  200,
  10
]

Wrath = neuralnetwork.FeedfowardNeuralNetwork(nodes)
Wrath.dataset = train_data
Wrath.weight = Wrath.train(
  epoch=5,
  learning_rate=0.1
)
```

작성한 신경망의 정확도를 테스트하기 위해 위에서 준비한 MNIST 데이터세트를 이용해 신경망을 학습시킨다. 그 후, 아래 코드를 이용해 신경망의 테스트 데이터에 대한 일반화 정확도를 측정한다.

```
score = sum([max_index(record[0]) is max_index(list(Wrath.query(record[1])[-1])) for record in test_data])/len(test_data)
```

결과는 아래와 같다.

학습 데이터 양	60,000
주기(epoch)	5회
학습률	0.1
학습 시간	약 23분
정확도	97.1%

2) 성능 개선 방안 적용

① Minibatch

미니배치는 학습 데이터세트의 레코드 몇 개를 '미니배치'라 불리는 하나의 집합으로 묶어 연산의 양을 줄이는 학습 기법이다. 이 때 미니배치 하나에 들어가는 레코드 양이 늘어날수록 연산의 양은 줄어들고, 따라서 최저값 수렴에 걸리는 시간 또한 줄어든다. 본래 신경망의 학습 메서드와 경사감소 메서드만 수정해서 작성한다(계승)

```
class FeedfowardNeuralNetwork_Minibatch(FeedFowardNeuralNetwork):
  def__init__(self, nodes, minibatch):
      super.__init__(nodes)
      self.minibatch = minibatch

  def descent(self, batch, lr, weight_defined=None):
      '''
       :keyword 확률적 경사감소 메서드
       :param batch 배치
       :param lr 학습률
       :param weight_defined 사용자 지정 가중치
       :return: 새로운 가중치 행렬
      '''
      weight = weight_defined if weight_defined elseself.weight[:]
      error = [0for i inrange(self.classes-1)]
      for input_data, target_data in batch:
        input_data = np.array([input]).T
        target_data = np.array([target]).T
        outputs =self.query(input_data)
      for i inrange(self.classes -1):
        if i is 0:
          error[-1-i] += target_data - outputs[-1]
        else:
          error[-1-i] += weight[-i].T @ prev_error
      for i, error inreversed(error):
        weight[-i -1] += lr * (
          error * outputs[-i -1] * (1.0- outputs[-i -1]) @ outputs[-i -2].T
        )
      return weight

def train(...):
      '''
       :keyword 학습 메서드
       :param epoch: 주기
       :param learning_rate 학습률
       :param weight_defined 사용자 지정 가중치
       :param dataset_defined 사용자 지정 데이터세트
       :param batch_defined 사용자 지정 배치(n(D))
       :return: 학습된 가중치 행렬
```

```
    '''
    weight = weight_defined if weight_defined elseself.weight[:]
    dataset = dataset_defined if dataset_defined elseself.dataset[:]
    batch = batch_defined if batch_defined else self.minibatch
    batch_set    =    [[dataset.pop(0)    for    i    in    batch]    for    l    in
    len(dataset)//batch]
    dataset =None;
    for _ in range(epoch):
      for minibatch in dataset:
        weight =self.descent(
          batch=minibatch,
          lr=learning_rate,
          weight_defined=weight
         )
    return weight
```

학습 데이터 양	60,000
주기(epoch)	5회
학습률	0.1
미니배치 수	10
학습 시간	약 6분
정확도	68.69%

② 가중치 감쇠

가중치 감쇠는 역전파시 계산하는 가중치 변화량에 전체 가중치의 제곱합을 더하는 기법으로, 이는 가중치가 자신에 비례하는 속도로 줄어들게 한다.

```
class FeedfowardNeuralNetwork_WeightAttenuation(FeedFowardNeuralNetwork):
    def__init__(self, attenuation_constant=10**(-3)):
    super.__init__()
    self.att_const = attenuation_constant

    defdescent(self, input, target, lr, weight_defined=None):
    '''
     :keyword 확률적 경사감소 메서드
     :param target: 목표 신호
     :param lr 학습률
     :param weight_defined 사용자 지정 가중치
     :return: 새로운 가중치 행렬
    '''
     input_data = np.array([input]).T
     outputs =self.query(input_data)
     target_data = np.array([target]).T
     weight = weight_defined if weight_defined elseself.weight[:]
    for i inrange(self.classes -1):
    if i is0:
     error = target_data - outputs[-1]
    else:
```

```
        error = weight[-i].T @ prev_error
        error+=(self.att_const/2)*self.sum_of_squares(weight[-i-1])
        weight[-i -1] += lr * (
        error * outputs[-i -1] * (1.0- outputs[-i -1]) @ outputs[-i -2].T
        )
        prev_error = error
    return weight

    defsum_of_squares(self, iterable):
        avg =sum(iterable)/len(iterable)
    returnsum((iterable-avg)**2)
```

학습 데이터 양	60,000
주기(epoch)	5회
학습률	0.1
감쇠상수	10^-8
학습 시간	약 24분
정확도	97,25%

Ⅲ. 결론

ⅰ. 데이터 해석

학습 데이터 양	60,000
주기(epoch)	5회
학습률	0.1
학습 시간	약 23분
정확도	**97.1%**

피드포워드 뉴럴넷에서의 학습 결과는 위와 같다. 이에 비해 미니배치를 사용한 신경망의 벤치마크 결과는 아래와 같다.

학습 데이터 양	60,000
주기(epoch)	5회
학습률	0.1
미니배치 수	10
학습 시간	**약 6분**
정확도	**68.69%**

미니배치는 학습 속도를 올리기 위해 Na개의 레코드를 하나의 집합으로 생각하며(미니배치) 가중치를 업데이트하는 메서드이다. 이 때, 학습 속도는 Na에 비례해서 빨라지는 경향을 보인다. 벤치마크 결과를 보면 피드포워드 뉴럴넷에 비해 미니배치를 사용했을 때의 정확도는 오히려 낮은데, 이는 미니배치를 사용함에 따라 학습 레코드 수가 $\frac{1}{N_a}$배가 되었기 때문에 최솟값에 도달하지 못하고 학습이 종료되었기 때문이다. 위에서 설명한 데이터 확장과 연계해서 사용하면 높은 정확도와 빠른 학습 속도를 보일것이라 예상된다. 또한, 좋지 않은 하드웨어 조건에서도 6분이라는 빠른 학습 속도를 보인 바, 빠른 학습 속도가 필요한 서비스나 연구에 이용하면 좋은 성과를 낼 수 있을 것이라 예상된다.

아래는 가중치 감쇠를 사용했을 때의 벤치마크 결과이다.

학습 데이터 양	60,000
주기(epoch)	5회
학습률	0.1
감쇠상수	10^-8
학습 시간	**약 24분**
정확도	**97,25%**

가중치 감쇠는 과적합을 완화시키고자 오차에 가중치의 제곱합과 감쇠상수의 곱을 더해주는 기법으로, 이는 지나치게 커진 가중치를 제약하여, 국소 최솟값에 빠지는 일을 방지한다. 감쇠상수는 감쇠의 정도를 결정하며, 이는 실험적으로 최적의 값을 구해야 한다. 본 탐구에서는 10^{-8}을 사용했다. 정확도는 일반 피드포워드 뉴럴넷보다 0.15% 상승한 것을 알 수 있다. 이는 가중치 감쇠를 통해 전역 최솟값에 더 접근했다는 것을 보이는 지표이다. 계산의 양이 늘어나기에 학습 시간이 약간 증가했지만, 높은 정확도를 요하는 서비스나 실험, 혹은 연구에서는 가중치 감쇠를 적용하면 피드포워드 뉴럴넷에 비해 좋은 결과를 도출해낼 수 있을 것이다.

ii. 향후 이용 가능성 및 결론

위의 과정을 통해 인공 신경망의 작동 원리를 탐구하였고, 그 성능 개선 방안을 탐구하였다. 그리고, 그 방안의 실효성과 향후 이용 가능 여부를 탐구하고자 직접 Python(+Numpy, Scipy)를 통해 신경망의 기본 코드를 작성하고, 탐구한 성능 개선 방안들을 직접 적용해 보았으며, 이를 통해 각각 성능 개선 방안의 장단을 파악하고, 이를 어떻게 활용하면 좋은 성능을 낼 수 있을지 해석했다.

우선, MiniBatch의 경우는 학습 속도는 굉장히 빨라졌으나, **가중치 수정의 횟수가 부족**해 최솟값에 도달하지 못하게 되는 현상이 일어났다. 미니배치를 사용할 때, 데이터를 복사하고, 약간의 변동을 가하는 **데이터 확장을 같이 사용**하면 **높은 성능과 빠른 학습 속도**를 보일 것이라 예상한다.

다음으로, **가중치 감쇠**는 정확도를 증가시키는 효과를 보이며, 가중치가 과도하게 커지는 경우를 방지했다. 이는 2개의 레이어를 갖는 본 신경망에서도 효과를 보였지만, 물체 유형 인식이나 알파고의 정책망, 심층신뢰망 같은 많은 Hidden Layer을 갖는 신경망의 학습에 매우 중요하게 작용한다. 이는 신경망의 Layer가 많아질수록 가중치의 자유도가 높아지고, 과적합할 확률이 높아지기 때문이다. 가중치 감쇠는 그 연산에 컴퓨터의 행렬 자원을 활용하기에 연산에 추가로 소모되는 시간이 크지 않다. 이를 고려하면, 과적합을 피하기 위해 가중치 감쇠를 적용하는 것은 현명한 선택이라고 할 수 있다.

인공 신경망 기술이 발전함에 따라 인류의 미래도 특이점에 가까워지고 있다. 4차 산업혁명, 4차 산업, 알파고, 빅데이터 등의 키워드들은 어느새 우리에게 익숙해졌고, Apple의 Siri, Samsung의 Bixby 등 인공지능 비서는 없어지면 허전할 정도로 우리 생활의 일부분이 되었다. 이런 기술들 또한 인공 신경망이 발전하여 적용된 사례이다. 예를 들어, 자연어 처리는 단어들을 형태소로 분류하고, 이에 대한 최적의 대답을 구하는데, 이 때 각각 인공 신경망이 사용된다. 현재 세계 바둑 랭킹 1위를 굳건히 지키고 있는 Alphago 또한, 기본적으로 게임판의 가치를 판단하는 가치망, 착수 위치를 결정하는 정책망이라 불리는 두 가지의 신경망이 구성한다.

미래는 신경망에 의해 동작하는 AI들이 음식을 주문하고, 어떤 요리를 할 것인지 추천하며, 최단 경로를 찾아 자동차를 운전하고, 학생에게 맞는 학습 플랜을 제안할 것이다. 미래 같은가? 아니다. 신경망 기술이 대폭적으로 발전하며, 언급한 기술들은 연구가 거의 끝나가거나 시판되고 있다(삼성사의 삼성 스마트허브(냉장고), 테슬라 사의 자율운전 자동차 연구, 클래스팅 사의 러닝카드(인공지능 학습 플래닝 및 학습 자료 제공 서비스)) 이는 신경망 기술의 대폭적인 발전 없이는 불가능했을 것이다. 하지만 아직 인공 신경망에는 해결할 과제들이 많다. 딥 러닝-많은 은닉층을 가진 신경망에서의 과적합-이는 가중치 감쇠와 같은 본 탐구에서 탐구한 방법들로 완화시킬 수 있다-과 연산 속도 문제-미니배치를 사용하거나, 배치 정규화 혹은 데이터 정규화를 사용하여 초기 학습/학습 속도를 높일 수 있다. 또한, GPU를 통한 병렬 연산 자원을 활용하여 해결할 수 있다(이는 AlphaGo가 사용한 방법과 동일하다. 알파고는 1천개 이상의 GPU를 병렬적으로 연결하여 연산을 수행한다)-등이 있다. 하지만 본 탐구에서 탐구한 방법들을 통해 위와 같은 문제들을 완화시킬 수 있다. 필자는, 본 탐구를 통해 여러 인공지능 관련 서비스를 준비하는 프로그래머들, 인공 신경망 기술을 발전시킬 수 있는 방법과 그 척도를 찾는 연구자들에게, 여러 성능 개선 방법과 그 개선의 객관적인 척도를 제공하고자 한다.

Ⅳ. 참고 문헌 및 미주

ⅰ. 참고 서적

신경망 첫걸음, 타리크 리시드(한빛미디어)
텐서플로 첫걸음, 조르디 토레스(한빛미디어)
딥러닝 제대로 시작하기, 오카타니 타카유키(제이펍)
인공지능, 머신러닝, 딥러닝 입문, 김의중(위키북스)

ⅱ. 참고 논문

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,
- Sergey Ioffe, Christian Szegedy
Neural Network model for a mechanism of pattern recognition unaffected by shift in position - Neocognitron,
- Fukushima. K
A logical calculus of the ideas immanent in nervous activity, Bulletiin of mathematical biophysics,
- Warren S. McCulloch&Walter Pitts
Perceptrons : An Introduction to computational Geometry,
- Marvin Minsky&Seymour Parpet
Improving the convergence of back-propagation learning with second-order methods, - S. Becker&Yann Lecun

ⅲ. 참고 웹사이트

http://umbum.tistory.com/ : UMBUM(Tistory 블로그)
http://invalidid.tistory.com/ : C3H8(Tistory 블로그 : 강준서)

ⅲ. 미주

1) 데이터세트 : '레코드'라 불리는 종속변수와 그 레이블의 쌍(x, y, z)의 모임
2) 알고리즘 : 어떤 문제를 해결하기 위한 절차, 방법, 명령어들의 집합
3) TLU : Threshold Logic Unit, 뉴런의 구조를 본따 만들어진 '논리 게이트' 알고리즘, 들어오는 신호를 합하고 가중치를 곱해 신호를 발산한다.
4) 층 : 같은 유닛들과 결합하면서, 서로 결합되지 않은 유닛들의 모임
5) 가중치 : 신경망에서 발산하는 신호에 곱해지는 값
6) 가중합 벡터 : 가중치의 합 벡터
7) 벡터화하면 : 행렬로 나타내면
8) SGD : 확률적 경사감소법, 레코드 하나만을 이용해 가중치를 수정한다
9) 정규화 : 프로그램이 처리하기 편하게 데이터를 가공하는 행위 또는 그 과정
10) 레코드 : 데이터세트의 한 원소, 종속변수와 레이블의 쌍
11) 그레이스케일 이미지 : 픽셀이 모두 검은색으로만 이루어진 이미지
12) 열벡터 : 열 1개만을 갖는 벡터