중앙고등학교

ML과 LLM의 작동 원리

인공지능 기초 심화탐구 보고서

2025-10-12

황태준
중앙고등학교
2025
2학년 7반 31번
지도교사 인기초 김화해 선생님

목차

I. 인공 신경망 4
I.1. 인공 신경망의 수학적 구조 4
I.2. 인공 신경망의 학습 방법 8
지도학습 8
비용함수 9
경사하강법 개요 10
경사하강법 수식 전개 11
경사하강법 계산 15
II. 인공 신경망 직접 구현해보기 16
III. 1부 참고자료 및 출처 27
IV. LLM과 트랜스포머 28

이 보고서를 보여달라는 친구들이 좀 있어서 글이 설명문 형식으로 되어 진행이 부진할 수 있는 점 양해 부탁드립니다.

Abstract

인공지능은 말그대로 인간이 컴퓨터로 만든 지능이다. 그런데, 지능이란 무엇인가? 지능이 있다는 것은 생각할 수 있다는 것인가? AI는 생각하는가? 겉으로 보기에는 윤리적 판단도 어느 정도 하고, 수학 문제도 풀어내고, 각종 사소한 건에 대해 물어보면 나름 합리적인 판단도 내려주므로 생각하는 것처럼 보이기도 한다. 하지만 중학교에서 알고리즘을 처음 배울 때, 컴퓨터는 일정한 형식의 명령으로 계산(computation)만 빠르게 할 수 있는 바보라고 하지 않았는가? 구체적이고 특정적이지 않은, 모호한 인간의 자연어를 인공지능은 어떻게 처리하고 이해하여 그로부터 양질의 출력을 제공하는가?

지능(intelligence)의 정의는 아직도 제대로 내려지지 못하고 있다. 하지만 여러 정의에서 공통적으로 언급되는 부분은, 정보나 기술을 획득하여 목표 성취에 적용하고 새로운 것을 창조한다는 부분이다. 이 과정에서는 학습이 필수적이다. 정보나 기술을 획득하여 저장하고 이해(?)하는 과정이 학습이기 때문이다. 지능을 어떻게 정의 내리냐에 따라 컴퓨터가 지능을 가질 수 있는지 없는지는 철학적인 문제가 되겠지만, 적어도 지능을 모방할 수는 있을 것이다.

인공 신경망은 이러한 발상에서 비롯한 기술이자 자료구조로, 일종의 그래프이다. 정점은 뉴런의 신경세포체[1]를, 간선은 축삭을 본따 그 역할을 대신한다고 하는데, 생명과학을 배운 입장에서 이게 정말 맞는 말은 아닌 것 같고, 일반인들의 이해를 돕기 위한 비유적 설명으로 받아들일 수 있을 것이다.

정점들은 계층별로 모여 있는데, 직접적인 입력이 들어오는 입력 계층, 실질적인 처리가 일어나는 은닉 계층, 출력이 일어나는 출력 계층으로 나뉜다. 입력 계층과 출력 계층은 하나이며, 은닉 계층은 개수 제한은 없으나 그 개수와 해당 계층에서의 정점 개수를 어떻게 하느냐에 따라 학습 결과 성적이 좌우된다. 각 계층 간 정점은 수많은 간선들이 연결하고 있다. 최적화 알고리즘에서 모식적 모형으로 쓰이는 것들과 비슷하게, 여러 정점으로 가는 각 간선에는 서로 다른 가중치가 부여되어 있다. 인공 신경망을 통한 학습의 목적은, 궁극적으로 이 가중치들을 적절히 조절하여 주어진 입력에서 알맞은 출력을 내

[1] 그냥 뉴런 자체를 의미한다고 하기도 한다.

는 것이다. 하지만 수많은 간선들에 대해 일일이 수동으로 가중치를 설정할 수 없으니, 원하는 결과로부터 가장 가까운 출력을 내도록 역추적을 하는 방법을 사용하여 자동으로 학습을 진행한다. 이 방법을 역전파(backpropagation)이라고 하고, 이를 실현하기 위해 다변수 미적분학과 선형대수학의 손을 빌린 수학적 아이디어를 경사하강법(gradient descent)라고 한다.

하지만 인공 신경망은 생각보다 오래된 기술이고, 단점이 명확하다. 연산의 효율성은 둘째치고, 인공 신경망은 지능이 아니라 특정 목적 달성을 위해서만 단련된 시스템이기 때문에, 예를 들어 개와 고양이 사진을 구별하는 인공 신경망에 사람의 사진을 넣으면 무조건 틀린 답을 내놓게 된다. 사람은 개도 고양이도 아무것도 아닌 것[2]도 아니기 때문이다. 인공 신경망은 그 태생적 한계 때문에, 범용적이 될 수 없다.

현재는 거대 언어 모델(LLM)[3]의 시대이다. 물론 기초적인 형태의 언어 모델은 특정 목적을 위해서만 만들어지기도 했지만, 구글의 '트랜스포머' 모델 개발 이후 문장 전체의 문맥을 파악할 수 있게 된 언어 모델은 급격히 범용 인공지능을 향해 달려나가 발전하고 있다. 지금 유명한 모든 또는 거의 모든 인공지능 언어 모델은 트랜스포머에 기반한다.

I. 인공 신경망

I.1. 인공 신경망의 수학적 구조

인공 신경망은 뇌의 구조에서 영감을 받았다고 하며, 실제 신경망을 모식적으로 본따 만든 그래프의 일종이다. 정점은 뉴런(neuron)이라고 부르며, 어떤 값을 나타내는 변수와 다른 계층의 정점을 가리키는 간선(또는 포인터)로 되어 있다. 각 뉴런이 가질 수 있는 값의 범위는 대개 실수 0에서 1이라던가(이런 걸 로짓[logit]이라고 한다), 정수 0에서 255라던가(16진수 RGBA 등) 하는 식으로 목적에 알맞게 임의로 정하지만 대개 관행을 따른다. 각 정점을 연결하는 간선에는 가중치가 부여되어 있으며, 출발 정점의 값이 간선을 거쳐 도착 정점에 값에 얼마나 영향을 주는지를 정하는 값이다.

이러한 계층 구조의 연속으로, 입력 계층에서 입력 데이터를 받고 은닉 계층들에서 정해진 규칙에 따라 선형적 가중치 합(weighted sum) 연산을 통해 데이터를 처리한 뒤 출력

[2] default 또는 empty 출력 정점
[3] 대규모 언어 모델? 그냥 편하고 엘엘엠이라고 하겠습니다.

계층으로 결과를 내놓는 것이 학습된 인공 신경망이 궁극적으로 수행해야 할 작업이다. 여기서 계층을 둠으로써 분류할 데이터가 가지는 특성을 점점 세분화하여 구분하게 될 수 있는데, 바로 후술하겠으나 그 '세분화'라는 것이 우리의 눈에 합리적으로 보이지 않을지도 모른다.

인공 신경망의 구조적 합리성은 사실 그리 직관적으로 와닿지 않는다. 어떻게 가중치들만 모아놓은 것이 모호한 데이터를 분류할 수 있게 하는 것이고, 도대체 각 정점에 할당된 값과 그것을 잇는 간선들의 가중치는 각각 무엇을 의미하는 것인지, 알기가 어렵다. 사실 인간이 수동으로 설계한 인공 신경망이 아닌 이상 학습 또한 결과 지향적으로 이루어지기 때문에 대부분의 경우 각 정점이 맡고 있는 바를 인간의 시각으로 보았을 때 논리적으로 합리적이지 않을 가능성이 높다.

조금 모호한데, 인공 신경망을 수학적으로 좀 더 엄밀하게 살펴보자. 먼저, 입력 계층을 제외한 계층에 속한 어떤 정점 N에 연결된 간선의 개수를 n 개라고 하자. 이때 간선이 연결된 정점의 값을 a_i, 간선의 가중치 값을 w_i라고 한다면 N의 값은 이전 정점과 간선들의 가중치에 의해서 정해지게 되는데, 직관적으로 선형 가중치 합인

$$\sum_{i=0}^{n} w_i a_i \tag{1.1}$$

가 된다. 이 값은 다시 다음 간선들의 가중치로 들어가 그 다음 계층의 정점들의 값에 영향을 주게 되는 연쇄적 반응을 일으킨다.

그런데, 정점에 할당될 수 있는 값의 범위는 정해져 있었지만 가중치 합으로 도출한 값은 어떤 값이든 될 수 있으므로 이 범위에 맞춰 주어야 하는데, 이때 활성화 함수를 사용한다. 대표적이고 잘 알려진 활성화 함수로는 시그모이드가 있다. 시그모이드는 특정한 함수를 가리키는 것이 아니라, 두 점근선을 사이로 기울어진 S자의 개형을 띠는 곡선들을 모두 이르는 것이며, 생물군이나 경제 등의 성장을 기술하는 로지스틱 곡선이기도 한다. 시그모이드의 종류로는 오차(誤差)함수 $\operatorname{erf} x$, 역탄젠트 $\arctan x$, 쌍곡탄젠트 $\tanh x$, 구데르만 함수 $\operatorname{gd} x \equiv \int_0^x \operatorname{sech} t \, \mathrm{d}t = \arctan(\sinh x)$ 등이 있으며, 별도의 스케일링(scaling) 조작 없이 0과 1 사이의 값을 가지는 아래 함수를 사용하기도 한다.

$$\frac{1}{1+e^{-x}} =: \sigma \tag{1.2}$$

앞으로 별도의 설명 없이 시그모이드 함수라고 하면 이렇게 임의로 정의한 σ 함수[4]를 일컫는 것으로 한다. 아까의 정점 N에 제대로 된(양식에 맞는) 값을 넣어 주려면, 앞선 가중치 합을 시그모이드 함수에 넣어야 한다.

$$N = \sigma\left(\sum_{i=0}^{n} w_i a_i\right) \tag{1.3}$$

활성화 함수에는 시그모이드만 있는 것이 아니다. 사실, 시그모이드는 오래된 활성화 함수로 기울기 소실 문제[5], 지수함수를 계산하기 위해 자원이 많이 드는 문제가 있다. 그래서 요즘은 ReLU(rectified linear unit) 등을 많이 사용한다. ReLU의 식은

$$\mathrm{ReLU}(x) = \max(0, x) = \begin{cases} x & (x \geq 0) \\ 0 & (x < 0) \end{cases} \tag{1.4}$$

로, 양수 부분에서 기울기가 사라지지 않으며 지수 연산이 없어 빠르다. 하지만 0 이하에서 기울기가 아예 없는데, 이것을 보완하기 위해 Leaky ReLU 같은 것도 있다.

$$\mathrm{LReLU}(x) = \max(0.1x, x) \tag{1.5}$$

다만 ReLU 계열도 단점은 있는데, 여전히 중심이 0이 아닌 문제가 있다. 다른 활성화 함수로는 maxout이라고 불리는, 입력값의 최대치를 그대로 반환하는 함수나 ELU, softmax 등등이 있다.

한 정점에 영향을 주는 요인은 너무 많고, 얼마나 제대로 학습을 시키던 간에 결과가 완벽하게 한 출력 정점에 100%로 전달될 수 없다. 사실, 대부분의 모호한 경우 80%의 정확도를 얻는 것도 힘든 일이며, 은닉 계층에서는 더욱 이 현상이 심하다. 은닉 계층에서의 일종의 나비효과로 출력 계층의 값이 들쭉날쭉하지 않게 하기 위해 작은 값들은 필터링 해버릴 수 있다. 우리는 선형적인 식을 다루고 있으므로, 필터링을 하기 위해서는 그냥 가중치 합에서 어떤 값을 빼버리면 된다. 특히 시그모이드는 가에서 기울기가 매우 완만하고 중앙 부분에서 가파르기 때문에, 값을 빼어서 그것이 충분히 작다면 거의 의미가 없는 값으로 변하여 초기 목적을 달성할 수 있게 된다. 이때 빼는 값을 편향(바이어스, bias)라고 한다. 구현할 때의를 위해 수식으로 표현할 때는 음의 값을 더하는 식으로 생각하자.

[4] 경우에 따라 이 표기는 표준편차와 헷갈릴 수 있다.
[5] 가에서 기울기가 0에 가까워서 기울기를 곱하는 연산을 시행하며 기울기가 점점 사라지는 문제

$$N = \sigma\left(\sum_{i=0}^{n} w_i a_i + b\right) \tag{1.6}$$

이렇게 한 정점의 값을 정할 수 있다. 하지만 우리는 단순히 여러 개의 정점을 다루는 것이 아니라, 각 계층에 속한 여러 개의 정점을 다루고 있으며, 계층은 간선의 연결 여부를 정하므로 중요한 정보이다. 따라서 정점을 합리적으로 표기하려면 계층과 계층에서 정점의 위치를 모두 표현해야 한다. 계층을 L, 계층 내의 위치를 i라고 한다면 한 인공 신경망 내의 어떤 정점은 $a_i^{(L)}$로 표현한다. 그러면 아까 한 정점을 정하는 식은 다음과 같다. $w_n^{(L)}$은 L 계층에서 오면서, 특정 정점에 연결된 것 중 n번째 간선의 가중치를 의미한다.

$$a_j^{(L)} = \sigma\left(\sum_{i=0}^{n} w_i^{(L-1)}\, a_i^{(L-1)} + b_{L-1}\right) \tag{1.7}$$

모양새가 끔찍하다! 이렇게 해서는 시그모이드에 넣기도 전에 어지럼증으로 쓰러지고 말 것이다. 그래서, 모든 계산은 선형적이라는 것을 이용해 매개변수들을 행렬곱으로 나타낼 수 있을 것이다. 예를 들면 이런 식으로 말이다.

$$\begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ \vdots \\ a_n^{(1)} \end{bmatrix} = \sigma\left(\begin{bmatrix} w_0^{(0)} & w_1^{(0)} & \dots & w_n^{(0)} \\ w_0^{(1)} & w_1^{(1)} & \dots & w_n^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_0^{(k)} & w_1^{(k)} & \dots & w_n^{(k)} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0^{(0)} \\ b_1^{(0)} \\ \vdots \\ b_n^{(0)} \end{bmatrix}\right) \tag{1.8}$$

이제 가중치 행렬을 $\mathbf{W}$, 정점 벡터를 $\mathbf{a}$, 바이어스 벡터를 $\mathbf{b}$라고 한다면 식은 아주 간단해지게 된다.

$$\mathbf{a}^{(1)} = \sigma\left(\mathbf{W}^{(0)}\mathbf{a}^{(0)} + \mathbf{b}^{(0)}\right) \tag{1.9}$$

더 포괄적으로,

$$\mathbf{a}^{(L+1)} = \sigma\left(\mathbf{W}^{(L)}\mathbf{a}^{(L)} + \mathbf{b}^{(L)}\right) \tag{1.10}$$

실제로 이런 식으로 순전파(feed forwarding)[6]가 코드 상으로 구현되며, 행렬의 배열인 텐서가 정말 많이 사용된다. 모 머신러닝 라이브러리는 아예 이름부터 텐서가 들어가기도 한다. 아래는 순전파 연산 구현 예시이다.

[6] 입력 계층으로 입력된 데이터가 여러 개의 은닉 계층을 거쳐서 출력 계층으로 값이 출력되는 과정을 순전파라고 한다.

```
import numpy as np
class FFNetwork:
    @staticmethod
    def sigmoid(x):
        return 1 / (1 + np.exp(-x));
    def __init__(self, weights, biases):
        self.weights = weights;
        self.biases = biases;
    def feedForwardUnit(self, a):
        for b, W in zip(self.weights, self.biases):
            a = sigmoid(np.dot(W, a) + b);
        return a;
    # 신경망 구조 구현 코드...
```

I.2. 인공 신경망의 학습 방법

지도학습

인공 신경망은 수많은 계층과 그 속의 더 수많은 정점과 간선을 거쳐 결과를 도출한다. 따라서 엄청나게 많은 간선의 가중치를 사람이 하나하나 조작하는 것은 비효율적이다. 따라서 학습은 특정한 알고리즘에 의해 자동으로 진행되어야 한다. 초등학교, 중학교 때 한 번씩은 접해봤고 인공지능 기초 수업시간에도 했었던 티처블 머신이나 엔트리 머신러닝 등등은 라벨을 통한 지도학습의 예시이다. 이것은 신경망에게 어떤 데이터가 어디에 속해야 하는지 알려줌으로써 비슷한 경향을 가진 데이터를 분류해내는데 적합한 가중치 값들을 정할 수 있도록 한다. 이것이 인공 신경망에서 '학습'의 본질이다.

처음에 신경망의 가중치는 모두 정해진 범위 안에서 무작위로 설정된다. 이 무작위성은 그리 중요한 것은 아니고, 서로 각각 어느 정도 차이가 있는 값들로 되어 있어서 어떤 데이터를 나와도 모든 출력 계층의 정점이 대체로 균등하게 활성화되면 된다. 이렇게 해야 지도가 용이하기 때문이다.

지도를 하려면 먼저 어떤 라벨에 속하는 데이터를 신경망에 입력한다. 신경망은 어떤 값을 출력할 것인데, 그 값에 관계 없이 그 데이터가 어디에 속하는지 알려준다. 신경망은 해당 데이터가 입력되었을 때 제시된 라벨에 대응하는 값을 출력하도록 자신의 가중치들을 조정한다. 이것을 반복함으로써 학습이 이루어진다. 그렇다면, 신경망에게 어떻게 데이터가 어디에 속하는지 알려줄 것이며, 그에 따라 가중치 값은 어떻게 조정해야 하는 것일까? 여기서 비용함수 또는 손실함수(cost function, loss function)의 개념이 등장한다.

비용함수

비용함수는 $C(\hat{\mathbf{y}}, \mathbf{y})$와 같이 표기한다. $\hat{\mathbf{y}}$는 모델이 예측하는 값[7], 즉 예측값의 벡터이고, $\mathbf{y}$는 실제값, 즉 정답의 벡터이다. 이 값이 작을수록 신경망 모델이 정확하다는 뜻이 된다. 여기서 비용함수의 이름이 나왔는데, loss(손실)는 말 그대로 정확도나 데이터 무결성 따위가 얼마나 손실되었느냐를 의미하고, cost(비용)은 이것을 최적화 문제의 관점으로 보아, 비용을 최소화하는 것을 목적으로 두었을 때 견해를 반영한 용어이다. 즉, 신경망의 학습은 사실상 가중치와 편향을 조정하여 비용함수의 값을 최소화하는 것이다.

$$\min_{\mathbf{W},\mathbf{b}} C(\hat{\mathbf{y}}, \mathbf{y}) \tag{1.11}$$

비용함수에는 여러 형태가 있는데, 대표적으로 평균제곱오차(MSE, mean squared error) 방식과 교차 엔트로피 방식이 있다.

먼저, MSE 꼴은 다음과 같다[8].

$$C = \frac{1}{2}|\mathbf{y} - \hat{\mathbf{y}}|^2 = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y)^2 \tag{1.12}$$

참고로, MSE는 분산과 식 구조가 동일하지만 주목하는 대상이 다르다. 근본적으로 MSE는 $\mathbb{E}\left[\left(Y - \hat{Y}\right)^2\right]$인데, 예측값 $\hat{Y}$를 기댓값 $\mathbb{E}(Y)$로 본다면(또는 두 값이 같다면) 이것은 분산 $\mathbb{E}\left[(Y - \mathbb{E}[Y])^2\right] = \mathbb{E}\left[Y^2\right] - (\mathbb{E}[Y])^2$과 같게 된다. 하지만 이것이 계산적인 측면에서 분산과 비슷한 의미를 가진다는 것을 실감하면 MSE가 의미하는 바를 어렵지 않게 알 수 있다. 이때, 차에 제곱을 취하므로, 정답보다 출력값이 멀면 멀수록 더 강한 페널티를 부여하게 된다.

교차 엔트로피(확률 분포) 꼴은 다음과 같다.

$$C = -\sum_{i=1}^{n} y_i \log \hat{y}_i \tag{1.13}$$

일반적으로 $\mathbf{y}$는

[7] expected(expectation) value 가 아니라 predicted value이다. 즉 기댓값(평균)과는 다르다.

[8] 벡터의 크기는 노름(norm) 표현 $\|\bullet\|$으로 쓰는 것이 좀 더 일반적이지만, 간단함을 위하여 절댓값 기호 $|\bullet|$로 통일하도록 하겠다.

$$\mathbf{y} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \tag{1.14}$$

과 같이 하나만 1인 one-hot encoded 벡터이므로 이 경우 남는 항은

$$C = -\log \hat{y}_{\text{정답 index}} \tag{1.15}$$

교차 엔트로피 꼴은 기댓값의 꼴 $\sum_i x_i p_i$과 비슷하게 생겼는데, $\mathbf{y} \sim y_i$는 실제 정답의 분포, $\hat{\mathbf{y}} \sim \hat{y}_i$는 모델이 예측한 분포라는 통계적 관점에서 봤을 때 그 의미가 드러나는 것이다. $\log \hat{y}$는 모델이 정답이 $\hat{y}$라고 확신하는 정도(confidence)를 의미한다. 여기에 음수를 취함으로써, 예측값이 실제값에 가까워져서, 정답에 대한 확실도가 커질수록 비용함수는 작아지게 된다.

분류에는 보통 교차 엔트로피 방식이 사용되나, 이 보고서에서 비용함수의 식이 추가로 등장하게 된다면 더 익숙하고, 무엇보다 대수함수라서 로우 레벨 언어로도 계산이 편리한 MSE로 가보도록 하겠다. MSE는 보통 분류보다는 회귀(regression), 실수값 예측 등에서 더 빈도 있게 사용되기는 한다.

경사하강법 개요

그렇다면 비용함수를 최소화할 방법은 무엇인가? 수학 II에서 배웠듯, 어떤 함수들은 도함수가 0인 지점에서 최솟값(global minimum) 또는 극솟값(Local minimum)을 가질 가능성이 있다. 즉, 임의의 매개변수 p에 대해

$$\frac{\mathrm{d}C}{\mathrm{d}p} = 0 \Rightarrow \min C = C(p) \tag{1.16}$$

가 성립…할까?

여기에는 몇 가지 문제점이 있다. 먼저, 가장 근본적으로 도함수의 값이 0이라고 해서 그 점이 무조건 극소점이 아니다. 반례는 널리고 널렸다. 다음으로, 비용함수는 1차원이 아니다. 하나의 매개변수에 의해서만 결정되는 함수가 아니라는 말이다. 즉, 도함수가 0인 지점을 방정식을 풀어 직접 찾아내는 것은 그리 좋은 방법이 아니다.

이것보다는, 아무 점에서 시작하여 함숫값이 작아지는 방향으로 점을 옮겨가면서 마침내 도함수가 0이 되는 곳을 찾는 것이 더 포괄적이고 유연할 것이다. 특히, 현재 위치한 점에서 접선의 기울기를 알 수 있다면 어느 방향으로 가야할지를 쉽게 알 수 있다. 접선의 기울기의 크기에 비례하여 움직이는 정도를 조정한다면 더 효율적으로 할 수 있을 것이다. 이 과정을 반복하다보면 어떤 극소점에 도달하게 된다. 한 가지 주목할 것은, 극소점은 이런 식으로 쉽게 찾을 수 있지만 최소점은 정말 찾기가 어렵다. 극소점이 최소점이라는 보장이 없으며, 어디서 점을 시작해야 최소점으로 굴러가는지 알 수 없기 때문이다.

어쨌든, 이 규칙은 매개변수 공간이 몇 차원이던간에 똑같이 적용된다. 해당 공간에서 점이 어느 방향으로 이동해야 함숫값이 작아지겠는가에 대한 것이기 때문에 꼭 1차원일 필요가 없는 것이다. 이렇게, 변수가 여러개일 때 사용할 수 있는 것이 그라디언트(gradient)[9] ∇C이다. 그라디언트는 공간의 기울기를 나타내는 연산자로, 이 경우에서는 어느 방향으로 가장 가파르게 증가하는지를 나타내는 벡터이다. 그러므로 $-\nabla C$는 어느 방향으로 가장 가파르게 감소하는지를 알려주게 된다. 극소를 찾는 방법은 어떤 점에서의 그라디언트를 계산하고, 그라디언트의 반대 방향으로 조금 이동하는 것을 반복하는 것이다.

이렇게 그라디언트를 이용해 함수 곡면을 따라 내려가면서 극소를 찾는다는 의미로 이 방법을 gradient descent 라고 하고, 한국어로 적절히 번역해서 경사하강법이라고 한다.

경사하강법 수식 전개

구체적으로 어떻게 하는 것인지 알아보자. 먼저 모델이 예측을 하는 것을 수학적으로 나타낼 필요가 있다. $\hat{\mathbf{y}}$라는 것은 $\mathbf{x}$를 입력했을 때, 가중치와 바이어스 $\boldsymbol{\theta}$에 따라 신경망 f를 통과해서 나온 출력이다. 즉, 다음과 같다.

$$\hat{\mathbf{y}} = f(\mathbf{x}, \boldsymbol{\theta}) \tag{1.17}$$

그러므로

$$C(\hat{\mathbf{y}}, \mathbf{y}) = C(f(\mathbf{x}, \boldsymbol{\theta}), \mathbf{y}) \tag{1.18}$$

[9] 이 보고서에서 델 연산자에 대해 더 깊이 다루지는 않는다. 하지만 그라디언트의 수학적 특성에 대해서는 조금 뒤에 서술하겠다.

이고, 지금 상황에서 입력 $\mathbf{x}$와 정답 $\mathbf{y}$는 정해진 것이므로 비용함수를 $C(\boldsymbol{\theta})$로 간단히 나타내도록 하자. 하지만 생략이 일어났음으로 명확히 하기 위해 변수의 종속 관계를 살펴보면,

$$\begin{aligned} \boldsymbol{\theta} &\mapsto f(\mathbf{x}, \boldsymbol{\theta}) = \hat{\mathbf{y}} \\ \hat{\mathbf{y}}, \mathbf{y} &\mapsto C(\hat{\mathbf{y}}, \mathbf{y}) \\ \Rightarrow \boldsymbol{\theta} &\mapsto C(f(\mathbf{x}, \boldsymbol{\theta}), \mathbf{y}) \\ \therefore C(\boldsymbol{\theta}) &= (C \circ f)(\mathbf{x}, \boldsymbol{\theta}) \end{aligned} \tag{1.19}$$

모델을 학습시킬 때는 모델이 예측한 출력값이 아닌 가중치와 바이어스를 조작하는 것이므로, $\boldsymbol{\theta}$에 주목할 필요가 있는 것이다. 이제 식을 유도하겠다. $\boldsymbol{\theta}$는 모든 가중치와 바이어스의 벡터이고, 편의를 위해 앞으로 볼드 표기를 하지 않고 그냥 θ로 쓰겠다. 목표는 아래를 구하는 것이다. d는 입력 벡터의 길이 또는 입력 변수의 개수, 즉 차원 수이다.

$$\min_{\theta \in \mathbb{R}^d} C(\theta) \tag{1.20}$$

그라디언트 $\nabla C(\theta)$는 각 성분에 대한 편미분으로 이루어진 벡터이다.

$$\nabla C(\theta) = \begin{bmatrix} \frac{\partial C}{\partial \theta_1} \\ \vdots \\ \frac{\partial C}{\partial \theta_d} \end{bmatrix} \tag{1.21}$$

함수 C에서 방향벡터 $\mathbf{u}$로의 순간변화율은 $\nabla C(\theta)^{\mathsf{T}}\mathbf{u}$이다[10]. 따라서 $\nabla C(\theta)$는 가장 가파르게 증가하는 방향이고, $-\nabla C(\theta)$는 가장 가파르게 감소하는 방향이다.

가장 단순한 형태의 경사하강법은 현재 위치에서 음의 그라디언트 방향으로 그라디언트의 크기에 비례해 이동하는 것이다.

$$\theta_{k+1} = \theta_k - \eta \nabla C(\theta_k) \tag{1.22}$$

여러 문헌에서 볼 수 있는 좀 더 일반적인 표기는[11] 다음과 같다. 이런 식을 업데이트 식이라고 하기도 한다.

[10] 외적 연산을 풀어 쓴 것이다.

[11] $\leftarrow$는 수학적으로는 $=$과 비슷하지만, 우변을 좌변에 대입, 할당한다는 의미를 강조한 것이다. 가장 간단한 꼴은 $a \leftarrow a + k$ 같은 것이 있겠으며, 코드로는 `a = a + k;` 로 쓰는 느낌이다.

$$\theta \leftarrow \theta - \eta \nabla C(\theta) \tag{1.23}$$

여기서 $\eta > 0$는 학습률이다. 식으로 보면 알 수 있지만, 학습률은 매개변수가 학습에 반영되는 정도를 결정한다. 학습률이 너무 낮으면 모델이 자기 수치를 조정하는 정도가 작아져 학습이 느리거나 덜 된다. 학습률이 너무 높으면 흔히 오버슈팅(overshooting)이라고 하는, 상대적으로 작은 매개변수에도 수치가 과잉 조정되는 현상이 일어날 수 있다. 이 현상은 모두 점이 극점에 도달하는 것을 방해한다.

기본적으로 C는 $\mathbb{R}$에서 연속이고 미분 가능하다고 가정해야 뭔가를 할 수가 있다. 이 가정에 의해 립시츠(Lipschitz) 규칙[12] $\Delta f(x) \leq \Delta x \max f(x)$가 성립한다. 따라서 ∇C에 대해 아래도 성립하게 된다.

$$\forall \mathbf{x}, \mathbf{y}\ \exists L > 0 \quad |\nabla C(\mathbf{x} - \nabla C(\mathbf{y}))| \leq L|\mathbf{x} - \mathbf{y}| \tag{1.24}$$

립시츠 연속 특성과 테일러 전개에 의해 아래가 성립하게 된다. 앞으로 $\mathbf{x}, \mathbf{y}$는 x, y로 쓰겠다.

$$\forall \mathbf{x}, \mathbf{y} \quad C(\mathbf{y}) \leq C(\mathbf{x}) + \nabla C(\mathbf{x})^{\mathsf{T}}(\mathbf{y} - \mathbf{x}) + \frac{L}{2}|\mathbf{y} - \mathbf{x}|^2 \tag{1.25}$$

$y = x - \eta \nabla C(x)$를 대입하면,

$$\begin{aligned} C(x - \eta \nabla C(x)) &\leq C(x) + \nabla C(x)^{\mathsf{T}}(-\eta \nabla C(x)) + \frac{L}{2}|\eta \nabla C(x)|^2 \\ &= C(x) - \eta |\nabla C(x)|^2 + \frac{L\eta^2}{2}|\nabla C(x)|^2 \\ &= C(x) - \left(\eta - \frac{L\eta^2}{2}\right)|\nabla C(x)|^2 \end{aligned} \tag{1.26}$$

따라서 $\eta \in \left(0, \frac{2}{L}\right)$이면 계수 $\eta - \frac{L\eta^2}{2} > 0$이므로 매 반복마다 C가 감소하여 하강하게 된다. C가 하강하지 않고 일정하게 유지될 수도 있는데, 이것은 극소점을 찾았다는 뜻이 된다. 특히, $\eta \leq \frac{1}{L}$일 때는

$$C(x - \eta \nabla C(x)) \leq C(x) - \frac{\eta}{2}|\nabla C(x)|^2 \tag{1.27}$$

[12] 실수 전체에서 연속이고 미분 가능한 함수는 접선의 기울기의 크기가 항상 어떤 상수보다 작다는 규칙. Lipschitz continuity 영문 위키백과 페이지에 단번에 이해가 되는 좋은 GIF가 있다.

즉 학습률이 합리적으로 작은 값인 상태에서 음의 그라디언트 방향으로 이동하면 C의 값이 감소하거나 일정함을 보장한다. 또, 거의 자명하게 볼록한 함수와 볼록하지 않은 함수에서 그라디언트 하강 시 극소점으로 갈 수 있음을 증명할 수 있는데, 이것은 재미없으므로 생략한다.

참고로, 다른 형태의 하강법도 존재한다. 대표적인 것으로 확률적 경사하강법(SGD, stochastic gradient descent)와 뉴턴 방법이 있다.

확률적 경사하강법은 기존의 경사하강법[13]이 전체 데이터셋을 입력받아 작동하기 때문에 느리고 자원을 많이 먹는다는 단점을 보완하기 위해 만든 방법으로, 데이터셋에서 임의의 샘플을 추출하고 그 샘플로만 경사하강법을 수행하는 것이다. 이 덕분에 자원을 덜 요구하고 학습 속도도 빠르다. 하지만 데이터의 수가 적다보니 불안정하게 하강하고 요동치게 된다. 최적해인 극소점에 근접하게는 도달하나, 극소점에 정확히 도착하지 못할 수도 있다. 이러한 문제를 해결하기 위해 미니배치 경사하강법(mini-BGD)가 등장했다. 데이터셋을 딱 나누어떨어지는 일정한 수의 배치로 나눈 다음, 각 배치에 대해 각각 경사하강법을 수행하고 평균을 내는 방식이다. 보통 SGD를 얘기하면 보통 이 mini-BGD를 말하는 것이다. 확률적 경사하강법의 업데이트 식은

$$\theta_{k+1} = \theta_k - \eta_k g_k \quad \text{where} \ \ \mathbb{E}[g_k] = \nabla C(\theta_k) \tag{1.28}$$

이때 수렴하기 위해 일반적으로 만족해야 하는 로빈스–먼로(Robbins–Monro) 조건[14]이 있다.

$$\sum_{k=0}^{\infty} \eta_k = \infty, \sum_{k=0}^{\infty} \eta_k^2 < \infty \tag{1.29}$$

뉴턴 방법은 뉴턴이 처음에 기계학습이 아닌 함수의 근, 즉 $f(x^*) = 0$이 되게 하는 x^* 값을 찾을 때의 해법으로 제시한 것으로, 자신이 발명한 미분을 적극 활용하여 아래와 같은 논리를 제시했었다.

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{1.30}$$

[13] 여태까지 살펴본 경사하강법으로, BGD(batch gradient descent)라고 한다.
[14] 자세히 다루진 않겠으나 이 내용이 실린 논문이 발표됨으로써 SGD의 기반이 다져졌다.

최적화 문제에서는 함수 값이 아니라 도함수 값이 0이 되게 하는 것이 목표이므로 1차원에서

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)} \tag{1.31}$$

처럼 되어야 하고, 우리의 경우 다변수를 다루고 있으므로 그라디언트와 함께 헤시안(Hessian)[15]을 사용한다.

$$\theta_{k+1} = \theta_k - \frac{\nabla C(\theta_k)}{H_C(\theta_k)} = \theta_k - H_C(\theta_k)^{-1}\nabla C(\theta_k) \tag{1.32}$$

지수적으로 최적해에 수렴하고, 더 정확한 결과를 보이지만 헤시안과 그 역행렬을 계산하는 과정이 시간과 자원을 많이 소모한다. 그러므로 헤시안을 정확히 구하는 것이 아니라 근사하는 방법(BFGS 등)을 사용하여 이 단점을 보완한다.

경사하강법 계산

모든 이론적인 부분이 완료되었으므로 이제 조금 더 구체적인 계산 부분으로 들어갈 수 있다. 어떤 입력 x에 대해 마지막 계층에서의 순전파는

$$z = Wx + b, \quad a = \sigma(z) \tag{1.33}$$

라고 할 수 있으며[16], MSE 비용함수 값은

$$C = \frac{1}{2}|a - y|^2 = \frac{1}{2}\sum_i (a_i - y_i)^2 \tag{1.34}$$

그러면 합성함수의 미분(chain rule)에 의해

[15] 헤시안 또는 헤세 행렬은 스칼라 함수 $f : \mathbb{R}^n \to \mathbb{R}$의 이계도함수 행렬이다. 일계 도함수 행렬은 야코비안(Jacobian)으로, 벡터 함수 $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$에 대응한다. 둘의 관계는 그라디언트와 라플라시안의 관계와 비슷하다.

[16] 이제 무엇이 벡터이고 무엇이 스칼라인지 알 수 있으므로 그냥 쓰겠다. 실제로 여러 이산수학, 기계학습 또는 다른 분야에서도 혼동의 여지가 없다면(있어도) 벡터와 스칼라를 구분하지 않는 교재가 많다.

$$\frac{\partial C}{\partial z} = \frac{\partial C}{\partial a} \odot \frac{\partial a}{\partial z} = (a - y) \odot \sigma'(z) \because C(a(z)) = \frac{1}{2}|\sigma(z) - y|^2,$$

$$\frac{\partial C}{\partial W} = \frac{\partial C}{\partial z}\frac{\partial z}{\partial W} = \left(\frac{\partial C}{\partial z}\right)x^{\mathsf{T}} \because \frac{\partial z}{\partial W} = x^{\mathsf{T}}, \tag{1.35}$$

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial z}\frac{\partial z}{\partial b} = \frac{\partial C}{\partial z} \because \frac{\partial z}{\partial b} = I$$

여기서 $\odot$은 원소별로 곱한다는 뜻이다. 이렇게 구한 값으로 경사하강을 하는 것이 역전파(back propagation)의 원리이며, 이를 통해 인공 신경망의 학습이 이루어진다.

II. 인공 신경망 직접 구현해보기

중학교 때 C로 연결리스트를 직접 만들던 것이 생각나서 순전파, 역전파로 학습하고 분류할 수 있는 간단한 인공 신경망을 구현하기로 했습니다. 파이썬으로 하는 게 정상이라던데 저는 사실 파이썬을 잘 못해서 익숙한 C++로 하기로 했습니다. 중간에 막히는 부분이 있어서 GPT한테 물어보면 파이썬으로만 대답하는 게 원통하덥니다. 그래도 알고리즘 짜는 측면으로의 도움은 좀 받을 수 있었습니다. 처음에는 MSE로 하려고 했으나 크로스 엔트로피가 해보니까 성능도 좋고 오히려 더 간단했어서 노선을 틀었습니다.

목표는 8×8 이미지를 2차원 `std::vector<int>` (사실상 `bool`)로 입력받아서 무슨 모양인지 분류하는 신경망을 만드는 것입니다.

코드 내장 주석 외에 추가 설명은 하지 않습니다.

이 코드는 본 문서 외에도 Github에 올라가 있습니다(https://github.com/tjwhang/CPPNeuralNetwork/blob/main/neural_network.cpp).

```
/**
 * *doxygen은 md를 지원함.
 * # 간단한 인공 신경망
 * C++17, STL만 사용
 * ## 기능
 * - "조밀한" 계층, 활성화함수 지수 시그모이드/쌍곡탄젠트/ReLU/항등, softmax + 교차 엔트로피
 * - SGD 최적화
 * - 순전파, 역전파 및 학습 유틸리티
 * - 8 x 8 (64차원) 이진 이미지 분류
 *
*/

#include <algorithm>
#include <chrono>
#include <cmath>
#include <cstddef>
#include <cstdint>
#include <exception>
#include <iomanip>
```

```cpp
#include <iostream>
#include <limits>
#include <numeric>
#include <random>
#include <stdexcept>
#include <string>
#include <tuple>
#include <utility>
#include <vector>
#include <memory>

// <출력용 전역 선언>
const std::string shades = " .:-=+*#%@"; // 아스키 아트 원리로 어두울수록 더 자리 많이 차지하는 문자 보여주기. 현재는 0, 1밖에 없지만 추후 그 사이 값을 갖는 픽셀도 다룰 수 있음.

// <수학적 기반 자료구조>

// 행렬/텐서
struct Matrix
{
    size_t rows{0}, cols{0};
    std::vector<double> data;

    Matrix() = default;
    Matrix(size_t r, size_t c, double v = 0.0) : rows(r), cols(c), data(r * c, v) {}

    double &operator()(size_t r, size_t c) { return data[r * cols + c]; }
    double operator()(size_t r, size_t c) const { return data[r * cols + c]; }

    static Matrix zeros(size_t r, size_t c) { return Matrix(r, c, 0.0); }
};

// 전치
static Matrix transpose(const Matrix &A)
{
    Matrix T(A.cols, A.rows);
    for (size_t r = 0; r < A.rows; ++r)
        for (size_t c = 0; c < A.cols; ++c)
            T(c, r) = A(r, c);
    return T;
}

// 행렬곱
static Matrix matmul(const Matrix &A, const Matrix &B)
{
    if (A.cols != B.rows)
        throw std::runtime_error("행렬곱: 차원 불일치");
    Matrix C(A.rows, B.cols, 0.0);
    for (size_t i = 0; i < A.rows; ++i)
    {
        for (size_t k = 0; k < A.cols; ++k)
        {
            double a = A(i, k);
            const size_t Brow = k * B.cols;
            const size_t Crow = i * C.cols;
            for (size_t j = 0; j < B.cols; ++j)
            {
                C.data[Crow + j] += a * B.data[Brow + j];
            }
        }
    }
    return C;
}

// 바이어스 더하기
static void add_bias_inplace(Matrix &Z, const std::vector<double> &b)
{
    if (Z.cols != b.size())
        throw std::runtime_error("바이어스: 행렬 차원 불일치");
    for (size_t i = 0; i < Z.rows; ++i)
        for (size_t j = 0; j < Z.cols; ++j)
            Z(i, j) += b[j];
}

// 행 더하기
```

```
static Matrix sum_rows(const Matrix &A)
{
    Matrix s(1, A.cols, 0.0);
    for (size_t i = 0; i < A.rows; ++i)
        for (size_t j = 0; j < A.cols; ++j)
            s(0, j) += A(i, j);
    return s;
}

// 행렬 간 차
static Matrix operator-(const Matrix &A, const Matrix &B)
{
    if (A.rows != B.rows || A.cols != B.cols)
        throw std::runtime_error("차: 행렬 차원 불일치");
    Matrix C(A.rows, A.cols);
    for (size_t i = 0; i < A.data.size(); ++i)
        C.data[i] = A.data[i] - B.data[i];
    return C;
}

// 행렬 간 합
static Matrix operator+(const Matrix &A, const Matrix &B)
{
    if (A.rows != B.rows || A.cols != B.cols)
        throw std::runtime_error("합: 행렬 차원 불일치");
    Matrix C(A.rows, A.cols);
    for (size_t i = 0; i < A.data.size(); ++i)
        C.data[i] = A.data[i] + B.data[i];
    return C;
}

// 실수배
static Matrix operator*(const Matrix &A, double s)
{
    Matrix C(A.rows, A.cols);
    for (size_t i = 0; i < A.data.size(); ++i)
        C.data[i] = A.data[i] * s;
    return C;
}

// 선형결합 A_i = A_i + alpha X_i
static void axpy_inplace(Matrix &A, const Matrix &X, double alpha)
{
    if (A.rows != X.rows || A.cols != X.cols)
        throw std::runtime_error("선형결합: 행렬 차원 불일치");
    for (size_t i = 0; i < A.data.size(); ++i)
        A.data[i] += alpha * X.data[i];
}

// 벡터 선형결합 a_i = a_i + alpha x_i
static void axpy_inplace_vec(std::vector<double> &a, const std::vector<double> &x, double
alpha)
{
    if (a.size() != x.size())
        throw std::runtime_error("선형결합: 행렬 차원 불일치");
    for (size_t i = 0; i < a.size(); ++i)
        a[i] += alpha * x[i];
}

// 영벡터
static std::vector<double> zeros_vec(size_t n) { return std::vector<double>(n, 0.0); }

// 가장 큰 원소가 속한 행 반환
static size_t argmax_row(const Matrix &A, size_t r)
{
    size_t idx = 0;
    double best = -std::numeric_limits<double>::infinity();
    for (size_t j = 0; j < A.cols; ++j)
    {
        if (A(r, j) > best)
        {
            best = A(r, j);
            idx = j;
        }
    }
```

```
    return idx;
}

// <난수생성, 초기화>

// 난수 생성기 (random number generator). x in [a, b) 난수 생성
struct RNG
{
    std::mt19937_64 gen; // 필기: Mersenne twister 알고리즘 기반 2^19937 - 1 주기의 x64 STL 난수
생성기. 고맙다 GPT
    explicit RNG(uint64_t seed = std::random_device{}()) : gen(seed) {}

    double uniform(double a, double b)
    {
        std::uniform_real_distribution<double> dist(a, b);
        return dist(gen);
    }
};

// Xavier uniform 신경망 초기화 알고리즘
static Matrix xavier_uniform(size_t in_dim, size_t out_dim, RNG &rng)
{
    double limit = std::sqrt(6.0 / static_cast<double>(in_dim + out_dim));
    Matrix W(in_dim, out_dim);
    for (double &w : W.data)
        w = rng.uniform(-limit, limit);
    return W;
}

// <활성화 함수>

// 활성화 함수 "인터페이스"... (UE C++가 그리운 부분)
struct IActivation
{
    virtual ~IActivation() = default;
    virtual Matrix forward(const Matrix &Z) = 0;
    virtual Matrix backward(const Matrix &dA) = 0; // 순전파 시 캐시 필요
};

// 활성화 함수들 구현은 GPT 참고
struct ReLU : IActivation
{
    Matrix cacheZ;
    Matrix forward(const Matrix &Z) override
    {
        cacheZ = Z;
        Matrix A(Z.rows, Z.cols);
        for (size_t i = 0; i < Z.data.size(); ++i)
            A.data[i] = std::max(0.0, Z.data[i]);
        return A;
    }
    Matrix backward(const Matrix &dA) override
    {
        if (cacheZ.rows ≠ dA.rows || cacheZ.cols ≠ dA.cols)
            throw std::runtime_error("ReLU 역전파: 차원 불일치");
        Matrix dZ(dA.rows, dA.cols);
        for (size_t i = 0; i < dA.data.size(); ++i)
            dZ.data[i] = (cacheZ.data[i] > 0.0) ? dA.data[i] : 0.0;
        return dZ;
    }
};

struct Sigmoid : IActivation
{
    Matrix cacheA;
    Matrix forward(const Matrix &Z) override
    {
        Matrix A(Z.rows, Z.cols);
        for (size_t i = 0; i < Z.data.size(); ++i)
            A.data[i] = 1.0 / (1.0 + std::exp(-Z.data[i]));
        cacheA = A;
        return A;
    }
    Matrix backward(const Matrix &dA) override
    {
```

```
        Matrix dZ(dA.rows, dA.cols);
        for (size_t i = 0; i < dA.data.size(); ++i)
            dZ.data[i] = dA.data[i] * cacheA.data[i] * (1.0 - cacheA.data[i]);
        return dZ;
    }
};

struct Tanh : IActivation
{
    Matrix cacheA;
    Matrix forward(const Matrix &Z) override
    {
        Matrix A(Z.rows, Z.cols);
        for (size_t i = 0; i < Z.data.size(); ++i)
            A.data[i] = std::tanh(Z.data[i]);
        cacheA = A;
        return A;
    }
    Matrix backward(const Matrix &dA) override
    {
        Matrix dZ(dA.rows, dA.cols);
        for (size_t i = 0; i < dA.data.size(); ++i)
            dZ.data[i] = dA.data[i] * (1.0 - cacheA.data[i] * cacheA.data[i]);
        return dZ;
    }
};

struct Identity : IActivation
{
    Matrix forward(const Matrix &Z) override { return Z; }
    Matrix backward(const Matrix &dA) override { return dA; }
};

// <계층>

struct ILayer
{
    virtual ~ILayer() = default;
    virtual Matrix forward(const Matrix &X) = 0;
    virtual Matrix backward(const Matrix &dA) = 0;
    virtual void step(double lr, double weight_decay = 0.0) = 0;
    virtual size_t params() const = 0;
};

// 싹 다 연결된 일반적인 "dense" 계층
struct Dense : ILayer
{
    Matrix W;              // dim in x dim out
    std::vector<double> b; // dim out

    // 캐시
    Matrix X_cache;          // 순전파 입력 저장, dim batch x dim in
    Matrix Z_cache;          // 순전파 선형 변환 결과 저장, dim batch x dim out
    Matrix dW;               // 역전파 시 가중치 그라디언트, dim in x dim out
    std::vector<double> db; // 역전파 시 바이어스 그라디언트, dim out

    std::unique_ptr<IActivation> act;

    // 생성자: Xavier 초기화, 바이어스는 0
    Dense(size_t in_dim, size_t out_dim, std::unique_ptr<IActivation> activation, RNG &rng)
        : W(xavier_uniform(in_dim, out_dim, rng)), b(out_dim, 0.0),
act(std::move(activation)) {}

    Matrix forward(const Matrix &X) override
    {
        if (X.cols ≠ W.rows)
            throw std::runtime_error("순전파: 차원 불일치");
        X_cache = X;

        // z = W x + b
        Z_cache = matmul(X, W);
        add_bias_inplace(Z_cache, b);

        // sigma(z)
        return act->forward(Z_cache);
```

```
    }

    Matrix backward(const Matrix &dA) override
    {
        Matrix dZ = act->backward(dA); // dim batch x dim out
        // 그라디언트
        Matrix Xt = transpose(X_cache); // dim in x dim batch
        dW = matmul(Xt, dZ);            // dim in x dim out
        Matrix ones = sum_rows(dZ);     // 1 x dim out
        db.assign(b.size(), 0.0);
        for (size_t j = 0; j < b.size(); ++j)
            db[j] = ones(0, j);
        // 그라디언트 입력
        Matrix Wt = transpose(W);   // dim out x dim in
        Matrix dX = matmul(dZ, Wt); // dim batch x dim in
        return dX;
    }

    void step(double lr, double weight_decay = 0.0) override
    {
        // L2 정규화 가중치 감쇠
        if (weight_decay != 0.0)
        {
            for (size_t i = 0; i < W.data.size(); ++i)
                W.data[i] *= (1.0 - lr * weight_decay);
        }
        // SGD 업데이트
        for (size_t i = 0; i < W.data.size(); ++i)
            W.data[i] -= lr * dW.data[i];
        for (size_t j = 0; j < b.size(); ++j)
            b[j] -= lr * db[j];
    }

    size_t params() const override { return W.data.size() + b.size(); }
};

// <비용함수: softmax 교차 엔트로피>

struct CrossEntropyWithLogits
{
    // 순전파에서 비용함수 기댓값 반환
    double forward(const Matrix &logits, const std::vector<int> &y)
    {
        if (logits.rows != y.size())
            throw std::runtime_error("CE: 배치 차원 불일치");
        double total = 0.0;
        tmp_softmax = Matrix(logits.rows, logits.cols);
        for (size_t i = 0; i < logits.rows; ++i)
        {
            // log-sum-exp for stability
            double maxv = -std::numeric_limits<double>::infinity();
            for (size_t j = 0; j < logits.cols; ++j)
                maxv = std::max(maxv, logits(i, j));
            double sumexp = 0.0;
            for (size_t j = 0; j < logits.cols; ++j)
            {
                double e = std::exp(logits(i, j) - maxv);
                tmp_softmax(i, j) = e;
                sumexp += e;
            }
            double logsumexp = std::log(sumexp) + maxv;
            int label = y[i];
            if (label < 0 || static_cast<size_t>(label) >= logits.cols)
                throw std::runtime_error("CE: label out of range");
            total += -logits(i, label) + logsumexp;
            // softmax 열 정규화
            for (size_t j = 0; j < logits.cols; ++j)
                tmp_softmax(i, j) /= sumexp;
        }
        return total / static_cast<double>(logits.rows);
    }

    // 역전파에서 dLogits 반환 (배치 x 클래스)
    Matrix backward(const Matrix &logits, const std::vector<int> &y)
    {
```

```
        if (logits.rows ≠ y.size())
            throw std::runtime_error("CE: 배치 차원 불일치");
        Matrix grad = tmp_softmax; // 이미 softmax
        for (size_t i = 0; i < logits.rows; ++i)
        {
            grad(i, y[i]) -= 1.0;
        }
        // 배치에서 기댓값
        double invB = 1.0 / static_cast<double>(logits.rows);
        for (double &g : grad.data)
            g *= invB;
        return grad;
    }

private:
    Matrix tmp_softmax; // 순전파/역전파 간 캐시
};

// <신경망>

struct NeuralNetwork
{
    std::vector<std::unique_ptr<ILayer>> layers;

    Matrix forward(const Matrix &X)
    {
        Matrix out = X;
        for (auto &ly : layers)
            out = ly->forward(out);
        return out;
    }

    // dLoss/dOut 부터 역전파 시작
    void backward(const Matrix &dOut)
    {
        Matrix grad = dOut;
        for (size_t i = layers.size(); i-- > 0;)
        {
            grad = layers[i]->backward(grad);
        }
    }

    void step(double lr, double weight_decay = 0.0)
    {
        for (auto &ly : layers)
            ly->step(lr, weight_decay);
    }

    size_t params() const
    {
        size_t p = 0;
        for (auto &ly : layers)
            p += ly->params();
        return p;
    }
};

// <학습할 때 써먹는거(유틸리티)>

struct SGDConfig
{
    double lr{0.05};
    double weight_decay{0.0};
    size_t epochs{10};
    size_t batch_size{16};
    uint64_t seed{42};
};

static std::vector<size_t> shuffled_indices(size_t n, std::mt19937_64 &g)
{
    std::vector<size_t> idx(n);
    std::iota(idx.begin(), idx.end(), 0);
    std::shuffle(idx.begin(), idx.end(), g);
    return idx;
}
```

```cpp
struct History
{
    std::vector<double> loss;
    std::vector<double> acc;
};

static double accuracy(const Matrix &logits, const std::vector<int> &y)
{
    size_t correct = 0;
    for (size_t i = 0; i < logits.rows; ++i)
        if (static_cast<int>(argmax_row(logits, i)) == y[i])
            ++correct;
    return static_cast<double>(correct) / static_cast<double>(logits.rows);
}

static History fit(NeuralNetwork &net, CrossEntropyWithLogits &loss_fn,
                   const Matrix &X, const std::vector<int> &y,
                   const SGDConfig &cfg)
{
    if (X.rows != y.size())
        throw std::runtime_error("학습: X/y 차원 불일치");
    History hist;
    RNG rng(cfg.seed);

    const size_t N = X.rows;
    std::vector<size_t> order(N);

    for (size_t epoch = 0; epoch < cfg.epochs; ++epoch)
    {
        // Shuffle
        order = shuffled_indices(N, rng.gen);
        double epoch_loss = 0.0;
        size_t seen = 0;

        for (size_t start = 0; start < N; start += cfg.batch_size)
        {
            size_t end = std::min(N, start + cfg.batch_size);
            size_t B = end - start;
            // 미니 배치
            Matrix Xb(B, X.cols);
            std::vector<int> yb(B);
            for (size_t i = 0; i < B; ++i)
            {
                size_t idx = order[start + i];
                yb[i] = y[idx];
                for (size_t j = 0; j < X.cols; ++j)
                    Xb(i, j) = X(idx, j);
            }

            // 순전파
            Matrix logits = net.forward(Xb);
            double L = loss_fn.forward(logits, yb);
            epoch_loss += L * static_cast<double>(B);

            // 역전파
            Matrix dlogits = loss_fn.backward(logits, yb);
            net.backward(dlogits);
            net.step(cfg.lr, cfg.weight_decay);

            seen += B;
        }
        epoch_loss /= static_cast<double>(N);

        // Evaluate on train set (quick)
        Matrix logits = net.forward(X);
        double accv = accuracy(logits, y);
        hist.loss.push_back(epoch_loss);
        hist.acc.push_back(accv);
        std::cout << "에포크 " << (epoch + 1) << "/" << cfg.epochs
                  << " - 손실(비용): " << std::fixed << std::setprecision(4) << epoch_loss
                  << " - 정확도: " << std::setprecision(4) << accv << "\n";
    }
    return hist;
}
```

```cpp
// <8 x 8 이미지 관련>

// 정수 0, 1을 64차원 double [0,1]로 flatten
static std::vector<double> flatten8x8(const std::vector<std::vector<int>> &img8x8)
{
    if (img8x8.size() != 8)
        throw std::runtime_error("flatten8x8: need 8 rows");
    std::vector<double> out;
    out.reserve(64);
    for (size_t r = 0; r < 8; ++r)
    {
        if (img8x8[r].size() != 8)
            throw std::runtime_error("flatten8x8: need 8 cols");
        for (size_t c = 0; c < 8; ++c)
            out.push_back(img8x8[r][c] ? 1.0 : 0.0);
    }
    return out;
}

static Matrix to_matrix(const std::vector<std::vector<double>> &rows)
{
    if (rows.empty())
        return Matrix();
    size_t R = rows.size();
    size_t C = rows[0].size();
    Matrix M(R, C);
    for (size_t i = 0; i < R; ++i)
    {
        if (rows[i].size() != C)
            throw std::runtime_error("to_matrix: ragged rows");
        for (size_t j = 0; j < C; ++j)
            M(i, j) = rows[i][j];
    }
    return M;
}

// <테스트용 데이터셋 생성기>
/**
 * 1. 네 개의 8x8 모양을 생성: +, X, □, \
 * 2. 노이즈 적용해서 다양화하기
 */

static std::vector<std::vector<int>> pattern_plus()
{
    std::vector<std::vector<int>> g(8, std::vector<int>(8, 0));
    for (int i = 0; i < 8; ++i)
    {
        g[3][i] = 1;
        g[i][3] = 1;
    }
    return g;
}
static std::vector<std::vector<int>> pattern_x()
{
    std::vector<std::vector<int>> g(8, std::vector<int>(8, 0));
    for (int i = 0; i < 8; ++i)
    {
        g[i][i] = 1;
        g[i][7 - i] = 1;
    }
    return g;
}
static std::vector<std::vector<int>> pattern_box()
{
    std::vector<std::vector<int>> g(8, std::vector<int>(8, 0));
    for (int i = 0; i < 8; ++i)
    {
        g[0][i] = g[7][i] = g[i][0] = g[i][7] = 1;
    }
    return g;
}
static std::vector<std::vector<int>> pattern_diag()
{
    std::vector<std::vector<int>> g(8, std::vector<int>(8, 0));
```

```
    for (int i = 0; i < 8; ++i)
        g[i][i] = 1;
    return g;
}

static void add_noise(std::vector<double> &v, double prob, RNG &rng)
{
    std::uniform_real_distribution<double> u(0.0, 1.0);
    for (double &x : v)
    {
        if (u(rng.gen) < prob)
            x = 1.0 - x; // NOT 게이트 (비트 반전)
    }
}

static void make_synthetic_dataset(Matrix &X, std::vector<int> &y, size_t per_class = 32,
uint64_t seed = 123)
{
    RNG rng(seed);
    std::vector<std::vector<int>> base[4] = {pattern_plus(), pattern_x(), pattern_box(),
pattern_diag()};

    const size_t C = 4;
    const size_t N = per_class * C;
    X = Matrix(N, 64);
    y.assign(N, 0);

    size_t idx = 0;
    for (size_t cls = 0; cls < C; ++cls)
    {
        for (size_t n = 0; n < per_class; ++n)
        {
            auto fv = flatten8x8(base[cls]);
            // add small random flips
            add_noise(fv, 0.05, rng);
            std::cout << "\n라벨: " << cls << "\n";
            for (int i = 0; i < 8; i++)
            {
                for (int j = 0; j < 8; j++)
                {
                    double v = fv[j * 8 + i];
                    int idx = static_cast<int>(v * (shades.size() - 1));
                    if (idx < 0)
                        idx = 0;
                    if (idx ≥ (int)shades.size())
                        idx = shades.size() - 1;
                    char c = shades[idx];
                    std::cout << c << c; // 모노스페이스 문자가 얇아서 두번 출력
                }
                std::cout << "\n";
            }

            for (size_t j = 0; j < 64; ++j)
                X(idx, j) = fv[j];
            y[idx] = static_cast<int>(cls);
            ++idx;
        }
    }
}

/**
 * <8 x 8 데이터 학습 방법>
 * - 데이터 포맷: 정수 0, 1의 8 x 8 벡터 배열, flatten8x8로 64차원 double로 flatten하기, 라벨은 정수
 * - 모델: 입력계층 차원은 64, 각 계층 당 활성화 함수 지정
 * - 학습시키기는 방법
 * 1) 데이터셋 배열 만들기: `Matrix X(N, 64)`, `vector<int> y(N)`
 * 2) `X(i, j)`를 flatten한 샘플 i로 fill, `y[i]`는 라벨로 fill
 * 3) `num_classes` 알맞게 지정
 * 4) `SGDConfig`에서 하이퍼파라미터(학습률(`lr`), 에포크, 배치 크기, 가중치 감쇠 + L2) 지정
 * 5) `fit` 호출
 * - 분류시키기: (1, 64)의 `Matrix` 생성, `net.forward` 호출해 logit 도출, argmax로 클래스 유추
 */

// <학습시키는 양식>
// // 1) 클래스 개수 지정
```

```
// const size_t num_classes = /* 예) 10 */;

// // 2) 데이터셋 생성
// const size_t N = /* 샘플 개수 */;
// Matrix X(N, 64);
// std::vector<int> y(N);

// // Fill X, y
// for (size_t i = 0; i < N; ++i)
// {
//     // Suppose you have img8x8_i as std::vector<std::vector<int>>(8,
std::vector<int>(8))
//     auto fv = flatten8x8(img8x8_i);
//     for (size_t j = 0; j < 64; ++j)
//         X(i, j) = fv[j];
//     y[i] = /* class id for sample i */;
// }

// // 3) 하이퍼파라미터 설정, 학습
// CrossEntropyWithLogits celoss;
// SGDConfig cfg;
// cfg.lr = 0.1;             // 학습률: 알잘딱
// cfg.epochs = 50;          // 에포크: 알잘딱
// cfg.batch_size = 32;      // 배치크기: 알잘딱
// cfg.weight_decay = 1e-4; // L2 가중치 감쇠: 필수 아님, 알잘딱
// auto hist = fit(net, celoss, X, y, cfg);

// // 4) 분류 테스트
// Matrix X1(1, 64);
// auto fv = flatten8x8(my_8x8);
// for (size_t j = 0; j < 64; ++j)
//     X1(0, j) = fv[j];
// Matrix logits = net.forward(X1);
// size_t pred = argmax_row(logits, 0);
// std::cout << "예측: 클래스 " << pred << "\n";

// main

int main()
{
    try
    {
        std::cout << "신경망 생성 중, 기다리세요..." << "\n";

        // 1) 신경망 만들기, 계층 64(입력) -> 32(은닉1) -> 16(은닉2) -> const(출력)
        const size_t input_dim = 64;
        const size_t num_classes = 4; // 라벨(클래스) 종류
        RNG rng(42);

        NeuralNetwork net;
        net.layers.emplace_back(std::make_unique<Dense>(input_dim, 32,
std::make_unique<ReLU>(), rng));
        net.layers.emplace_back(std::make_unique<Dense>(32, 16, std::make_unique<ReLU>(),
rng));
        net.layers.emplace_back(std::make_unique<Dense>(16, num_classes,
std::make_unique<Identity>(), rng)); // logits

        std::cout << "매개변수: " << net.params() << "\n";

        // 2) 테스트용 데이터셋 생성
        Matrix X;
        std::vector<int> y;
        make_synthetic_dataset(X, y, /*per_class=*/40, /*seed=*/2025);

        // 3) 하이퍼파라미터 설정, 학습
        CrossEntropyWithLogits celoss;
        SGDConfig cfg;
        cfg.lr = 0.1;
        cfg.epochs = 30;
        cfg.batch_size = 32;
        cfg.weight_decay = 1e-4;
        auto hist = fit(net, celoss, X, y, cfg);

        // 4) 학습 성적 평가
        Matrix logits = net.forward(X);
```

```
        double acc = accuracy(logits, y);
        std::cout << "최종 학습 정확도: " << std::fixed << std::setprecision(4) << acc << "\n";

        // 5) 8x8 패턴 입력 후 결과 출력
        auto ex = flatten8x8(pattern_diag()); // pattern_plus, pattern_x, pattern_box,
pattern_diag 중 택1
        add_noise(ex, 0.1, rng);              // 노이즈 적용

        // 생성된 패턴 보여주기
        std::cout << "\n 생성된 입력 데이터 \n";
        for (int i = 0; i < 8; i++)
        {
            for (int j = 0; j < 8; j++)
            {
                double v = ex[j * 8 + i];
                int idx = static_cast<int>(v * (shades.size() - 1));
                if (idx < 0)
                    idx = 0;
                if (idx ⩾ (int)shades.size())
                    idx = shades.size() - 1;
                char c = shades[idx];
                std::cout << c << c; // 모노스페이스 문자가 얇아서 두번 출력
            }
            std::cout << "\n";
        }

        std::cout << "\n";

        Matrix X1(1, 64);
        for (size_t j = 0; j < 64; ++j)
            X1(0, j) = ex[j];
        Matrix out = net.forward(X1);
        size_t pred = argmax_row(out, 0);
        std::cout << "예측: 클래스 " << pred << " (0:+, 1:X, 2:ㅁ, 3:\\)\n";

        return 0;
    }
    catch (const std::exception &e)
    {
        std::cerr << "오류: " << e.what() << "\n";
        return 1;
    }
}
```

III. 1부 참고자료 및 출처

- 여러 위키백과 영문 문서: 개념 정립 및 서술에 큰 도움을 주었습니다. 학술적 내용의 위키백과의 경우 해당 문서에서 제시하는 모든 서술에 주로 전공서나 논문으로 출처가 적혀있기 때문에 신뢰 가능하다고 볼 수 있겠습니다(설마 누가 수학적 사실로 장난질을..!).
- 3Blue1Brown의 Neural Networks 시리즈: 직접적인 내용을 적는 데는 도움이 별로 안 되었지만 내용 구성을 참고하여, 산발적으로 탐구한 내용을 정리할 때, 논리적 흐름을 전개해서 짜임 있는 보고서를 만드는 데 도움을 주었습니다.
- Polyak, B. (1987), “Introduction to Optimization”
- Akilov, G. P. Kantorovich, L. V.(1982). “Functional Analysis” 2/e
- Holmes, M. (2023), “Introduction to Scientific Computing and Data Analysis” 2/e

- IBM 社의 학습 자료들. “What is gradient descent?”, “What is learning rate?” 등등

IV. LLM과 트랜스포머

기말 때 완성 예정입니다 ^^7

감사합니다.