

정보의 항해: 데이터는 신경망을 어떻게 통과하는가?

순전파(Forward Propagation)와 행렬 연산으로 읽는 신경망 내부

◎ 강의 목표

- ✓ 신경망의 순전파 과정을 행렬곱(Matrix Multiplication) 관점에서 이해합니다.
- ✓ 층을 거치며 변화하는 데이터 텐서의 형태(Shape)를 추적합니다.
- ✓ 행렬 연산을 공간의 변형으로 시각화하여 직관을 얻습니다.

#순전파 (Forward Propagation)

#행렬곱 (Matrix Mult)

#특징 추출 (Feature Extraction)

#차원 변화 (Shape)

도입: 수억 개의 파라미터, 계산은 어떻게 한꺼번에?

개별 연산의 한계를 넘어 행렬 연산(Matrix Operations)으로



거대한 연산량의 집합

현대 신경망(LLM 등)은 수십억~수조 개의 파라미터를 가집니다. 이를 **for 루프 (Loop)**로 하나씩 계산한다면 엄청난 시간이 소요됩니다.

The Problem



벡터화 (Vectorization)

루프를 사용하지 않고 데이터를 **행렬(Matrix)** 단위로 묶어 한 번에 처리합니다. 단일 명령어(SIMD)로 수천 개의 연산을 동시에 수행합니다.

The Solution



하드웨어 친화적 구조

GPU와 TPU는 **행렬 곱셈(GEMM)**에 최적화되어 있습니다. 신경망을 행렬 연산으로 정의해야 하드웨어 성능을 100% 활용할 수 있습니다.

Acceleration



데이터의 흐름과 형태(Shape)

오늘의 핵심 관점입니다. 복잡한 수식 대신 데이터 텐서의 **Shape 변화**와 **공간 변형**을 따라가며 신경망 내부를 여행합니다.

Key Insight

개별 뉴런에서 층(Layer)으로

뉴런이 모여 구조적 층을 이루고, 가중치가 모여 행렬을 이룹니다.

03

단일 뉴런

하나의 뉴런은 입력을 받아 가중합(Weighted Sum)을 구하고 활성화 함수를 통과시키는 단순한 연산 유닛입니다.

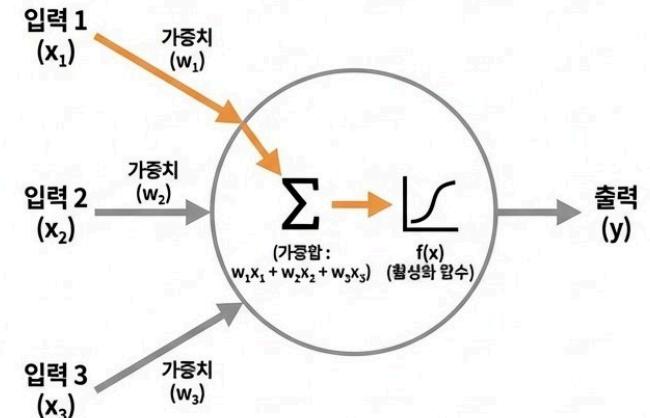
층 (Layer)의 확장

여러 뉴런을 수평으로 배치하여 동시에 서로 다른 특징을 추출합니다. 이것이 바로 신경망의 기본 빌딩 블록인 완전 연결 층(Fully Connected Layer)입니다.

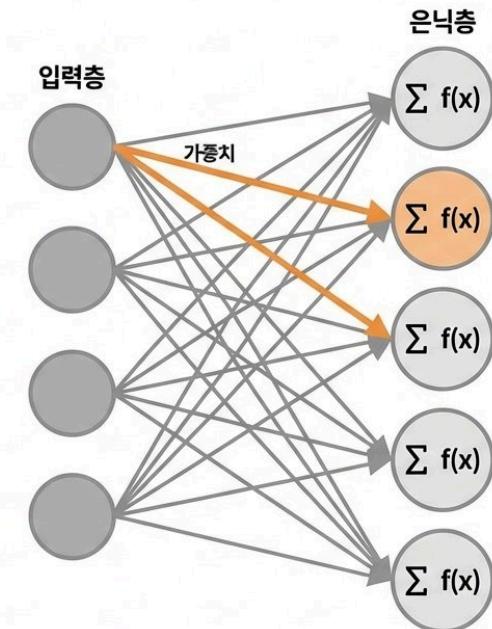
행렬(Matrix)의 형성

각 뉴런으로 향하는 가중치 묶음들이 모여 하나의 거대한 **가중치 행렬 W**를 형성합니다. 개별 계산은 이제 행렬 연산이 됩니다.

단일 뉴런 구조



완전연결층 (은닉층) 형성



▣ 그림: 단일 뉴런(좌)이 모여 완전 연결 층(우)을 형성하는 과정 시각화

가중치 합 다시 보기: $z = Wx + b$

04

입력을 가중치로 변환하고 편향을 더하는 선형 변환(Linear Transformation) 과정

$$z = Wx + b$$

W

가중치 행렬 (Weights)

Shape: (Output, Input)

x

입력 벡터 (Input Vector)

Shape: (Input, 1)

b

편향 벡터 (Bias)

Shape: (Output, 1)

z

선형 결합 결과 (Logits)

Shape: (Output, 1)

선형 변환의 관점

단순한 곱셈과 덧셈이 아닙니다. 이 수식은 입력 데이터가 존재하는 공간을 변형시키는 과정입니다.

기하학적 의미 (GEOMETRIC MEANING)



Wx (행렬곱): 입력 공간을 회전(Rotate)시키거나 확대/축소(Scale)합니다.



$+ b$ (덧셈): 변환된 공간을 원점에서 이동(Translation)시킵니다.

그 다음은?

이 선형 변환(z) 만으로는 복잡한 문제를 풀 수 없습니다. 그래서 비선형 함수인 활성화 함수 $f(z)$ 가 뒤따라오게 됩니다.

왜 행렬 연산인가?

전통적인 반복문(Loop) 처리와 현대적인 행렬곱(Matrix Multiplication)의 비교

Deep Learning Standard

반복문 (Loop)

```
# Python 순수 반복문
result = []
for i in range(len(X)):
    sum = 0
    for j in range(len(W)):
        sum += X[i] * W[j]
```

느린 성능 (Overhead)

인터프리터가 매 반복마다 타입을 체크하고 메모리를 할당하는 오버헤드가 발생합니다.

순차적 처리 (Serial)

한 번에 하나씩 계산하므로 최신 CPU/GPU의 병렬 코어를 활용하지 못합니다.

낮은 추상화

수학적 의도보다는 구현의 세부 사항(인덱스 관리 등)에 집중해야 합니다.

행렬곱 (MatMul)

```
# Vectorized Operation
import
torch

Z = torch.matmul(X, W)
# 또는 Z = X @ W
```

VS

압도적 성능 (Optimized)

BLAS/LAPACK 등 고도로 최적화된 저수준 라이브러리가 캐시 효율을 극대화합니다.

병렬성 (Parallelism)

SIMD(Single Instruction Multiple Data)를 통해 수천 개의 곱셈을 동시에 수행합니다.

수치적 안정성 & 추상화

부동소수점 오차를 최소화하며, 연산을 하나의 수학적 단위로 다룰 수 있습니다.

입력 벡터와 가중치 행렬의 만남

색상과 화살표로 추적하는 행렬곱($X \times W$)의 결합 과정

06

색상 매칭 규칙

입력 벡터(x)의 각 요소는 가중치 행렬(W)의 행(Row)과 1:1로 대응합니다. 빨간색 입력(x_1)은 행렬의 빨간색 행과 만나 연산됩니다.

점곱(Dot Product)의 수렴

같은 색상의 화살표들이 곱해져서 출력 노드(y)로 수렴(Sum)합니다. 이것이 $y_j = \sum (x_i \times w_{ij})$ 수식의 시각적 의미입니다.

필터로서의 열(Column)

가중치 행렬의 각 열(Column)은 하나의 출력 뉴런을 담당하는 '필터' 역할을 하며, 입력의 어떤 특징을 강조할지 결정합니다.

행렬곱 연산

입력 벡터 X

x_1
x_2
x_3

가중치 행렬 W

w_{11}	w_{12}
w_{21}	w_{22}
w_{31}	w_{32}

출력 벡터 Y

y_1
y_2

입력 벡터 X

가중치 행렬 W

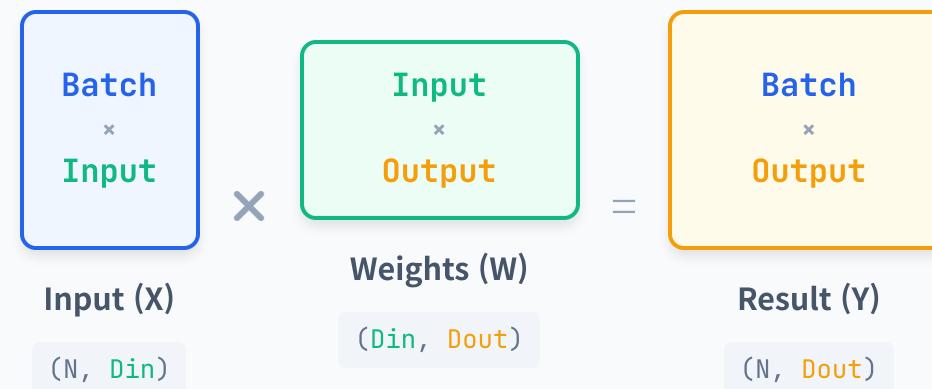
출력 벡터 Y

그림: 입력 벡터(좌)와 가중치 행렬(중)이 결합하여 출력(우)을 만드는 과정

데이터의 형상(Shape) 변화

07

행렬곱 연산을 통한 텐서 차원의 변화 추적



텐서 (Tensor)	역할	Shape 예시 (MNIST)	비고
Input (X)	입력 데이터 배치	(64, 784)	64장 이미지, 각 784 픽셀
Weights (W)	가중치 파라미터	(784, 256)	784 차원이 매칭되어 사라짐
Bias (b)	편향 (더하기)	(256,)	자동으로 (64, 256)으로 확장
Output (Y)	연산 결과	(64, 256)	Batch 크기는 유지됨

"1" 편향의 브로드캐스팅

편향 벡터 **b**는 원래 1차원 `(Output,)` 형태입니다.

하지만 행렬곱 결과인 **Y**는 2차원 `(Batch, Output)` 입니다. 이때 파이토치/넘파이는 자동으로 **b**를 배치 크기만큼 복사하여 더해줍니다.

$$(1, 256) \rightarrow (64, 256)$$

가중치 행렬의 기하학적 의미

행렬곱은 공간을 뒤틀어 데이터를 분류하기 쉽게 만듭니다.

08

▣ 공간의 선형 변환

행렬곱 Wx 는 단순한 숫자 계산이 아닙니다. 입력 공간 전체를 회전(Rotation)시키거나, 늘리고 줄이는(Scaling) 기하학적 변환 과정입니다.

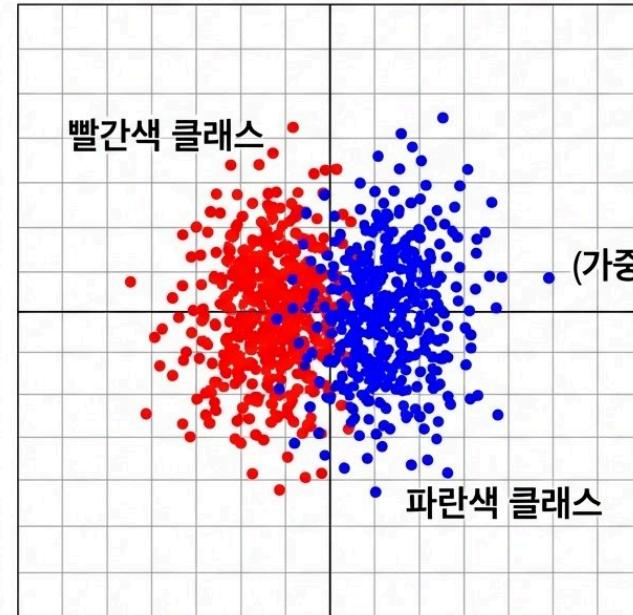
↗ 데이터 구름의 분리

원본 공간에서 서로 뒤엉켜 있던 데이터 포인트들이 공간의 뒤틀림을 통해 서로 멀어지거나 정렬됩니다. 마치 구겨진 종 이를 펴서 점들을 분리하는 것과 같습니다.

◆ 분류 경계의 확보

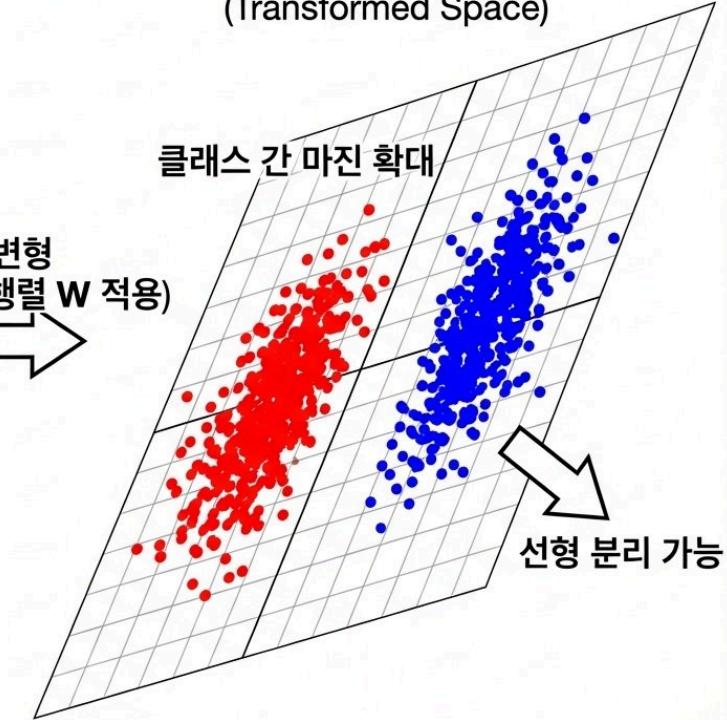
각 층을 통과할 때마다 데이터는 분류하기 더 쉬운 형태로 변합니다. 최종적으로는 **선형 분리(직선 하나로 구분)**가 가능한 상태가 되는 것이 목표입니다.

원본 좌표계
(Original Coordinate System)



변형
(가중치 행렬 W 적용)
→

변형된 공간
(Transformed Space)

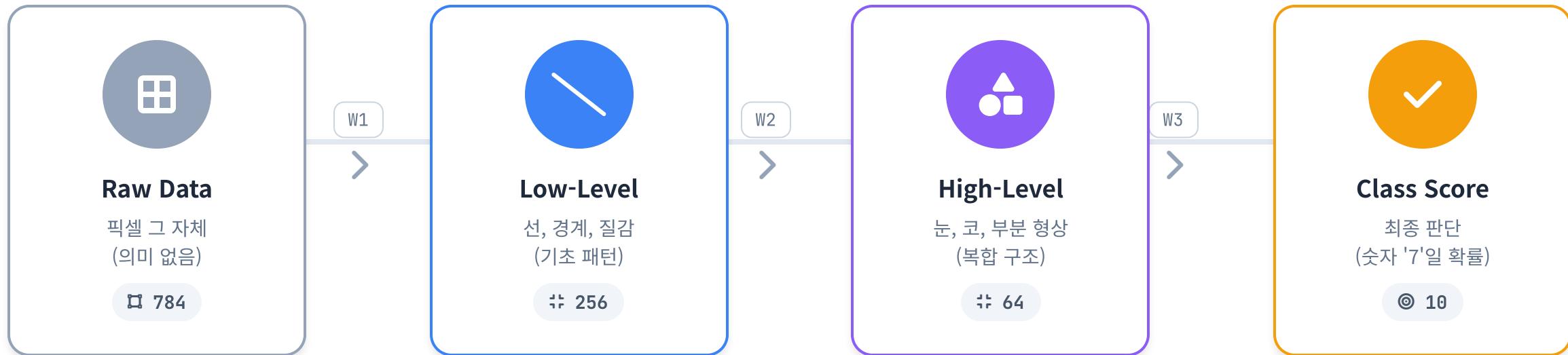


▣ 그림: 원본 공간(좌)이 가중치 행렬에 의해 변형되어 클래스가 분리되는 과정(우)

특징 추출 (Feature Extraction)

층을 거듭할수록 데이터는 압축되고 의미는 놓축됩니다.

09



정보의 증류 (Distillation)



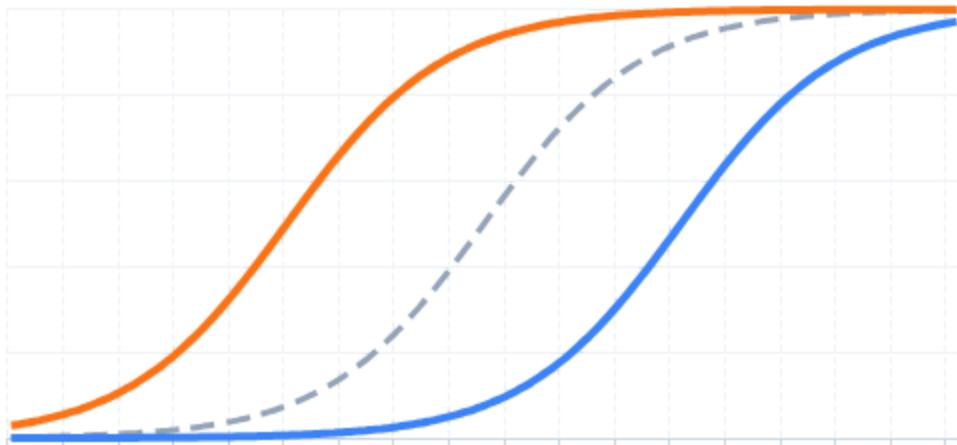
층을 통과할 때마다 벡터의 차원(크기)은 줄어들지만, 데이터가 담고 있는 **정보의 밀도**는 높아집니다. 신경망은 불필요한 노이즈를 버리고 정답을 맞추는데 필요한 **핵심 특징(Feature)**만을 남기는 방향으로 학습합니다.

편향(Bias)의 역할: 문턱 조절

활성화 함수를 좌우로 이동시켜 뉴런이 언제 '점화'될지 결정합니다.

10

$$y = f(Wx + b)$$



● $b = 0$ (기준)

● $b > 0$ (왼쪽)

● $b < 0$ (오른쪽)

활성화 함수의 평행이동

가중치(W)가 그래프의 **기울기(가파름)**을 조절한다면, 편향(b)은 그래프를 좌우로 밀어주는 역할을 합니다.

문턱(THRESHOLD) 비유

- ▣ **활성화의 조건:** 뉴런이 신호를 다음 층으로 전달하기 위해 넘어야 하는 '장벽'의 높이를 조절합니다.
- ✚ **양수 편향 (+ b):** 장벽을 낮추어(그래프를 왼쪽으로 이동), 더 작은 입력에도 쉽게 활성화되게 만듭니다.
- − **음수 편향 (- b):** 장벽을 높여(그래프를 오른쪽으로 이동), 강한 입력이 와야만 활성화되게 만듭니다.

브로드캐스팅 (Broadcasting)

Data: (Batch, Output) + Bias: (1, Output)

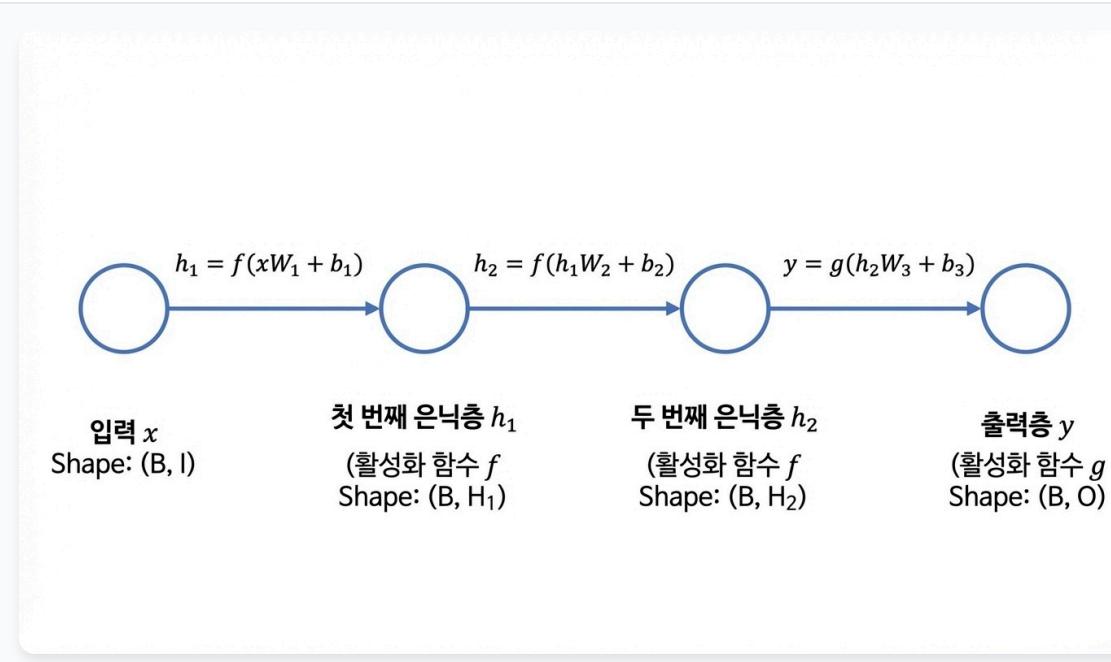
Result: (Batch, Output)

하나의 편향 벡터 b 가 배치의 모든 데이터 샘플에 동일하게 더해집니다.

다층 구조의 연산 흐름

입력에서 출력까지, 데이터가 거치는 연쇄적인 변환 과정

11



함수의 합성 (Composition of Functions)

문

신경망은 간단한 함수들의 연쇄적인 결합입니다. 입력 x 가 첫 번째 층을 통해 h_1 이 되고, 다시 두 번째 층의 입력이 되어 h_2 가 되는 과정을 반복합니다.

Step 1: Hidden Layer 1

$$h_1 = f(W_1x + b_1)$$



Step 2: Hidden Layer 2

$$h_2 = f(W_2h_1 + b_2)$$



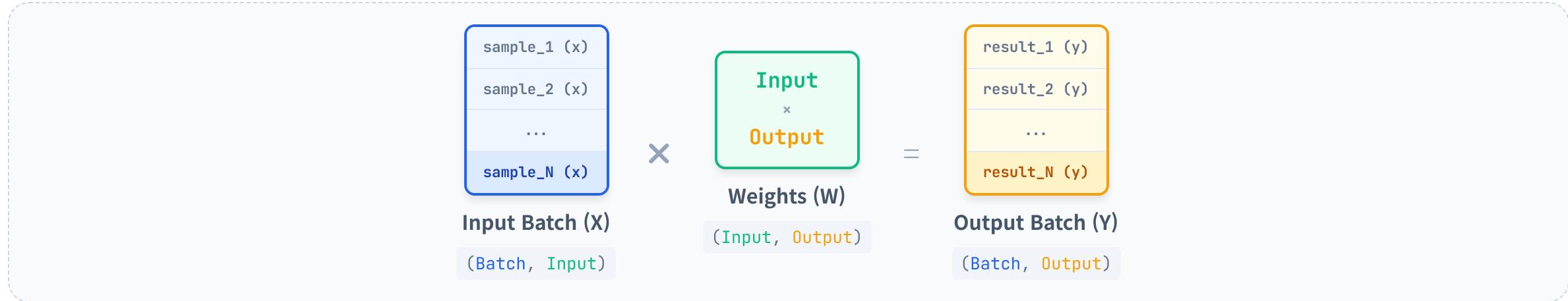
Step 3: Output Layer

$$y = g(W_3h_2 + b_3)$$

병렬 처리와 배치(Batch)의 개념

데이터를 하나씩이 아닌 '묶음'으로 처리하여 효율성 극대화

12



비교 항목	반복문 (For Loop)	행렬 배치 (Batch Matrix)
처리 방식	한 번에 하나씩 순차 처리	N개 데이터를 묶어서 동시 처리
하드웨어	CPU (순차적 연산에 강함)	GPU (병렬 연산에 최적화)
데이터 Shape	(1, Input)	(Batch_Size, Input)
속도	데이터 양에 비례해 느려짐	메모리 허용 범위 내에서 거의 일정함

⚡ 왜 배치가 더 빠를까?

SIMD (Single Instruction, Multiple Data)

GPU는 수천 개의 코어를 가지고 있어, 하나의 명령(곱셈)을 서로 다른 데이터(배치 내 샘플들)에 동시에 적용할 수 있습니다.



→



64개 계산

완전 연결 층의 한계: 파라미터 폭발

모든 뉴런이 연결될 때 발생하는 비효율성과 구조적 해결 방안

⚠ 완전 연결 (Dense)

Input Img:	224 × 224 × 3 ≈ 150k
Output:	4,096 Units
Weights:	150k × 4096
Total:	≈ 616,000,000 (6억개)



메모리 부족 (Out of Memory)

단 한 층의 가중치만으로도 수 기가바이트(GB)의 VRAM을 차지하여 학습이 불가능합니다.



과적합 (Overfitting) 위험

파라미터가 너무 많아 데이터의 노이즈까지 암기해버리며, 일반화 성능이 떨어집니다.

✓ Modern Solution

구조적 개선 (CNN 등)

Filter:	3 × 3 × 3 (Kernel)
Channels:	64 Outputs
Calc:	(3×3×3) × 64
Total:	≈ 1,728 (천 개 수준)

VS



가중치 공유 (Weight Sharing)

이미지 전체에 동일한 필터를 적용하여 파라미터 수를 획기적으로 (99% ↓) 줄입니다.



지역적 연결 (Local Connectivity)

모든 픽셀이 아닌 인접한 픽셀끼리의 패턴(에지, 질감)만 우선적으로 학습합니다.

연산 결과 시각화: 이미지 → 추상 벡터

픽셀 격자가 층을 거치며 의미를 가진 수치 뭉치로 변환됩니다.

14

田 1. 원본 이미지 (Raw Input)

단순한 픽셀 밝기값의 2차원 배열입니다. 컴퓨터에게는 아직 의미 없는 숫자 격자일 뿐입니다.

☱ 2. 은닉층 (Hidden Layers)

가중치 행렬과 연산하며 **특징(Feature)**이 추출됩니다. 예지, 질감 등의 패턴이 활성화 맵 (Activation Map)으로 나타납니다.

☲ 3. 출력 벡터 (Abstract Vector)

최종적으로 데이터는 **고차원 공간의 좌표**로 변환됩니다. 이 벡터는 "숫자 7"과 같은 의미론적 정보를 담고 있습니다.



단원 요약: 순전파의 핵심

15

데이터가 신경망을 통과하며 일어나는 4가지 핵심 과정



선형과 비선형의 연쇄

순전파는 입력에 가중치를 곱하고 편향을 더하는 **선형 변환**($z=Wx+b$)과, 복잡한 패턴 학습을 위한 **활성화 함수**($f(z)$)가 층층이 반복되는 과정입니다.



텐서 형상(Shape)의 법칙

행렬곱 시 내적 차원은 사라지고 배치는 유지됩니다: $(B, I) \times (I, 0) = (B, 0)$. 또한 편향(Bias)은 브로드캐스팅을 통해 배치 전체에 자동으로 적용됩니다.



벡터화된 고속 연산

수백만 번의 개별 연산을 루프(Loop)로 처리하는 대신, **행렬곱(Matrix Multiplication)**으로 묶어 GPU의 병렬 처리 능력을 극대화하고 연산 효율을 높입니다.



공간을 뒤틀어 분류하다

가중치 행렬은 데이터가 존재하는 공간을 **회전시키고 늘립니다(Warping)**. 신경망은 이 변형을 통해 서로 얹힌 데이터들을 분류하기 쉬운 형태로 재배치합니다.