6.86: Introduction to Machine Learning

통합 강의 노트

Contents

Chapter 1

MIT 6.86x: 예측의 미학

Course Structure & Current Focus

- **Unit 1: Introduction to ML (현재 단원: 운동장 세팅)**
  - 1.1 Formalizing the Problem (함수 근사 문제)
  - 1.2 Hypothesis Space ($\mathcal{H}$)
  - 1.3 Learning Paradigms (지도 vs 비지도)
  - 1.4 Generalization (E_{in} vs E_{out})
- Unit 2: Linear Classifiers (Perceptron)
- Unit 3: Neural Networks

1.1 Unit 1. 학습의 기초 (Fundamentals of Learning)

> 우리는 *18.6501*(통계학)에서 "데이터가 어떤 분포에서 나왔는가(*Inference*)?"를 고민했습니다. *6.86x*(머신러닝)에서는 질문을 바꿉니다. **"그래서, 내일 주가가 오를까 내릴까(*Prediction*)?"** 분포의 모양보다는, 정답을 맞히는 성능에 목숨을 거는 새로운 여정이 시작됩니다.

□ 개요 (Overview)

이 단원은 코드를 짜기 전, 머신러닝 문제를 수학적으로 정의하는 단계입니다. 머신러닝을 **"미지의 목표 함수 f^*를 찾기 위해 가설 공간 $\mathcal{H}$를 탐색하는 과정"으로 정의하고, 학습의 궁극적 목표인** 일반화(Generalization)**의 개념을 확립합니다.

□ 핵심 용어 사전

용어 (Term)	직관적 의미 (Meaning)
Target Function (f^*)	신(God)만이 아는 진짜 정답 규칙. 우리는 이걸 모른다.
Hypothesis (h)	우리가 f^*라고 추측한 모델 (가설).
Hypothesis Space ($\mathcal{H}$)	h를 찾을 수색 범위 (예: 직선들, 신경망들).
Inductive Bias	데이터를 보기도 전에 정해둔 편견 (예: "답은 직선일 거야").
Generalization	안 배운 문제(Test Data)도 잘 맞히는 능력.

1.1.1 1. 학습 문제의 정의 (Formalizing the Problem)

□ 개념 1: 함수 근사 (Function Approximation)

한 줄 요약: 머신러닝은 입력(x)과 출력(y) 사이의 숨겨진 관계식(f^*)을 데이터(S_n)를 통해 역추적하는 과정입니다.

1) 직관적 비유: 장인의 레시피 복원

유명 맛집의 비법 소스(f^*)가 있습니다. 사장님은 레시피를 절대 안 알려줍니다.

- **Data (S_n):** 우리가 맛볼 수 있는 건 결과물(음식) 뿐입니다.
- **Goal:** 맛을 보고 재료와 비율을 역추적해서, 사장님의 맛과 거의 똑같은 맛을 내는 나만의 레시피(h)를 만드는 것입니다.

2) 수학적 구성 요소

- **Input ($\mathcal{X}$):** 재료 (벡터 $x \in \mathbb{R}^d$)
- **Output ($\mathcal{Y}$):** 맛 (값 y)

- **Unknown Target (f^*):** $f^* : \mathcal{X} \to \mathcal{Y}$. (이상적인 정답 규칙)
- **Data (S_n):** $\{(x^{(i)}, y^{(i)})\}_{i=1}^n$. ($f^*$에 노이즈가 섞인 관측치)
- **Final Hypothesis (h):** 우리가 찾은 최적의 함수. $h \approx f^*$이기를 희망함.

1.1.2 2. 가설 공간 (Hypothesis Space, $\mathcal{H}$)

□ 개념 2: 수색 범위 설정

한 줄 요약: "세상의 모든 함수" 중에서 정답을 찾을 수는 없습니다. "직선 중에서 찾자" 혹은 "곡선 중에서 찾자"처럼 탐색 범위를 제한해야 합니다.

1) 귀납적 편향 (Inductive Bias)

가설 공간 $\mathcal{H}$를 정하는 순간, 우리는 데이터에 대한 **편견(Bias)**을 갖게 됩니다.

- 선형 회귀를 쓴다면? → "세상은 선형적일 거야"라는 편견.
- 편견이 나쁜가요? **아니요, 필수입니다.** 편견(제약 조건)이 없으면, 데이터 점들을 잇는 방법이 무한히 많아서 학습 자체가 불가능합니다.

2) Trade-off

- $\mathcal{H}$가 너무 작음 (예: 상수 함수): 정답을 표현 못 함 (Underfitting).
- $\mathcal{H}$가 너무 큼 (예: 100차 다항식): 노이즈까지 다 외워버림 (Overfitting).

1.1.3 3. 지도 학습 vs 비지도 학습

□ 개념 3: 정답지의 유무

한 줄 요약: 문제집 뒤에 해설지가 있으면 지도 학습, 해설지 없이 문제들의 유형을 스스로 정리해야 하면 비지도 학습입니다.

1) 지도 학습 (Supervised)

- 데이터: (x, y) 쌍.
- 목표: $h(x) \approx y$ 인 h를 찾음. (회귀, 분류)

2) 비지도 학습 (Unsupervised)

- 데이터: x만 있음.

- 목표: 데이터의 구조, 패턴, 군집을 찾음. (클러스터링, 차원 축소)
- 예시: 넥슨 유저들의 행동 로그(x)만 보고 "이들은 하드코어 유저군이다"라고 그룹핑하기.

1.1.4 4. 일반화 (Generalization)

□ 개념 4: 머신러닝의 존재 이유

한 줄 요약: 연습문제(Training Data)를 100점 맞는 건 의미 없습니다. 실전 수능(Unseen Data)을 잘 보는 것이 진짜 목표입니다.

1) 두 가지 위험 (Risk)

- **경험적 위험 (E_{in}, Empirical Risk):** 지금 가진 학습 데이터(S_n)에서의 오차. (모의고사 성적)
- **실제 위험 (E_{out}, True Risk):** 전체 데이터 분포($\mathcal{D}$)에서의 기대 오차. (수능 성적)

2) 근본적인 긴장 관계 (Fundamental Tension)

우리는 알고리즘을 통해 E_{in}을 줄입니다. 하지만 E_{in}을 0으로 만든다고 E_{out}이 0이 될까요?

- **Overfitting:** $E_{in} \approx 0$이지만 E_{out}은 매우 큰 상태. (답을 달달 외워서 응용을 못함)
- **Goal:** E_{in}도 작게 하면서, 동시에 $E_{out} \approx E_{in}$이 되도록 보장하는 것. 이것이 6.86x 과정의 핵심 질문입니다.

1.2 실전 시나리오: 넥슨 게임 이탈자 예측

□ Scenario: 완벽한 예측 모델의 함정

당신은 유저 이탈 예측 모델을 만들었습니다. 과거 1년 치 데이터(S_n)로 학습시켰더니, 정확도 99.9%($E_{in} \approx 0$)가 나왔습니다. "완벽해!"라고 외치며 실 서버에 배포했습니다.

1. **결과:** 다음 달, 모델은 이탈자를 하나도 못 맞췄습니다. (E_{out} 폭망)
2. **원인 분석:** 모델을 뜯어보니 이런 규칙이 있었습니다. "아이디가 '*User1234*'이고 *3*월 *5*일에 접속한 사람은 이탈한다."
3. **해석:** 가설 공간 $\mathcal{H}$를 너무 복잡하게 잡아서, 유저의 행동 패턴(f^*)을 배운 게 아니라 특정 유저의 ID와 날짜(노이즈)를 외워버린 것입니다.
4. **해결:** 모델을 단순화(Regularization)하거나 데이터를 더 모아서 **일반화 성능(E_{out})**을 챙겨야 합니다.

1.3 자주 묻는 질문 (FAQ)

Q1. f^*와 h의 차이가 정확히 뭔가요? A. f^*는 '진리(Truth)'이고 h는 '추측(Guess)'입니다. 예를 들어, 물리 법칙($F = ma$)은 f^*입니다. 우리가 실험 데이터를 통해 $F = 0.98ma + 0.02$라는 식을 얻었다면 이것이 h입니다. 우리는 영원히 f^*를 완벽하게 알 수는 없고, h를 f^*에 가깝게 만들 뿐입니다.

Q2. Inductive Bias가 왜 필요한가요? A. 편견 없이는 학습도 없습니다(No Free Lunch Theorem). "모든 것이 가능하다"는 말은 "아무것도 알 수 없다"는 말과 같습니다. "답은 연속적일 거야", "답은 간단할 거야" 같은 가정이 있어야만 유한한 데이터로부터 무한한 미래를 예측할 수 있습니다.

> **Next Step:** "일반화"가 중요하다는 것은 알았습니다. 그렇다면 가장 단순하면서도 강력한 가설 공간인 **'선형 모델(Linear Model)'**부터 시작해볼까요? 다음 **Unit 2**에서는 직선 하나로 데이터를 분류하는 **퍼셉트론(Perceptron)** 알고리즘을 배웁니다.

Unit 1 핵심 요약

- **목표:** 데이터(S_n)를 이용해 미지의 함수 f^*에 근사하는 h를 찾는다.
- **가설 공간($\mathcal{H}$):** 탐색 범위를 제한하는 것. 필연적으로 편견(Bias)을 동반한다.
- **일반화:** 학습 데이터 오차(E_{in})가 아니라, 보지 못한 데이터의 오차(E_{out})를 줄이는 것이 진짜 목표다.
- **Overfitting:** E_{in}은 낮지만 E_{out}이 높은 상태. (암기왕)

Chapter 2

MIT 6.86x: 딥러닝의 시작점

Course Structure & Current Focus

- Unit 1: Introduction (학습의 정의)
- **Unit 2: Linear Classifiers (현재 단원: 선 긋기)**
  - 2.1 Geometric Structure (초평면과 법선 벡터)
  - 2.2 Perceptron Algorithm (틀릴 때만 고친다)
  - 2.3 Geometric Intuition (벡터의 회전)
  - 2.4 Linear Separability & Convergence (수학적 보장)
- Unit 3: Neural Networks (선형 분류기의 적층)

2.1 Unit 2. 선형 분류기 (Linear Classifiers)

> *Unit 1*에서 우리는 "일반화(*Generalization*)"가 학습의 목표임을 배웠습니다. 그렇다면 가장 단순하면서도 일반화가 잘 되는 모델은 무엇일까요? 바로 '직선'입니다. 이번 단원에서는 컴퓨터가 어떻게 데이터를 보고 스스로 최적의 직선(결정 경계)을 찾아내는지, 그 알고리즘인 **퍼셉트론(*Perceptron*)**을 배웁니다.

□ 개요 (Overview)

선형 분류기는 데이터 공간을 **초평면(Hyperplane)**으로 나누어 분류를 수행합니다. 이를 학습시키는 **퍼셉트론 알고리즘**은 "틀린 데이터가 나올 때마다 경계를 수정한다"는 단순한 원리로 작동합니다. 이 단원에서는 벡터의 내적과 합을 이용한 학습 원리와, 알고리즘이 언제 멈추는지 보장하는 **수렴 정리**를 기하학적으로 다룹니다.

□ 핵심 용어 사전

용어 (Term)	직관적 의미 (Meaning)
Hyperplane (초평면)	공간을 반으로 가르는 칼날 (2D에선 직선, 3D에선 평면).
Normal Vector (θ)	칼날의 방향을 결정하는 나침반 (법선 벡터).
Decision Boundary	$\theta \cdot x + \theta_0 = 0$ 인 지점. 여기를 넘어가면 판정이 바뀜.
Linear Separability	직선 하나로 데이터를 완벽하게 100% 가를 수 있는 상태.
Margin (γ)	경계선과 데이터 사이의 안전거리 (도로의 폭).

2.1.1 1. 기하학적 구조: 결정 경계 (Decision Boundary)

□ 개념 1: 공간을 가르는 칼

한 줄 요약: 법선 벡터 θ는 "정답(Positive)이 있는 방향"을 가리키고, 초평면은 그 θ에 수직인 벽입니다.

1) 초평면 (Hyperplane)

d차원 공간을 두 개의 영역으로 쪼개는 $(d-1)$차원 평면입니다.

$$h(x) = \text{sign}(\theta \cdot x + \theta_0)$$

- $\theta \cdot x + \theta_0 > 0$: θ가 가리키는 방향 (Positive Class, +1)
- $\theta \cdot x + \theta_0 < 0$: θ의 반대 방향 (Negative Class, -1)
- $\theta \cdot x + \theta_0 = 0$: 결정 경계 (Boundary)

2) 법선 벡터 (θ)의 의미

θ는 단순한 기울기가 아닙니다. 초평면과 **수직(Orthogonal)**이며, 분류의 **기준이 되는 방향**입니다.

2.1.2 2. 퍼셉트론 알고리즘 (Perceptron Algorithm)

□ 개념 2: 실수에서 배운다 (Mistake Driven)

한 줄 요약: 맞추면 가만히 있고, 틀렸을 때만 θ를 수정합니다.

알고리즘 순서

1. $\theta = 0$ (혹은 랜덤)으로 초기화.
2. 데이터 $x^{(i)}$를 하나 꺼내 예측함: $\hat{y} = \text{sign}(\theta \cdot x^{(i)})$.
3. **Case 1 (정답):** $\hat{y} = y^{(i)} \rightarrow$ 아무것도 안 함 (Pass).
4. **Case 2 (오답):** $\hat{y} \neq y^{(i)} \rightarrow \theta$ 업데이트.

업데이트 규칙 (Update Rule)

$$\theta_{new} = \theta_{old} + y^{(i)}x^{(i)}$$

- 정답이 $+1$인데 틀렸으면: 더한다 ($\theta + x$)
- 정답이 -1인데 틀렸으면: 뺀다 ($\theta - x$)

2.1.3 3. 업데이트의 기하학적 해석 (Geometric Intuition)

□ 개념 3: 벡터의 회전

한 줄 요약: θ에 x를 더하면, 벡터의 합 원리에 의해 θ가 x 쪽으로 회전합니다.

상황 분석

- **상황:** 정답은 양성($+1$)인데, 현재 θ는 x와 둔각($> 90°$)을 이뤄 음수로 예측했습니다.
- **처방:** $\theta \leftarrow \theta + x$
- **효과:**
  1. 벡터 덧셈의 평행사변형 법칙을 상상해 보세요.
  2. 기존 θ와 x의 중간 방향으로 새로운 θ_{new}가 생깁니다.
  3. 즉, θ가 x**를 향해 회전**합니다.

4. 각도가 줄어들었으므로, 다음번에 내적을 하면 양수(정답)가 될 확률이 높아집니다.

2.1.4 4. 선형 분리 가능성 (Linear Separability)

□ 개념 4: 자 하나로 가를 수 있는가?

한 줄 요약: 직선 하나로 Red팀과 Blue팀을 완벽히 나눌 수 있어야만 퍼셉트론이 작동합니다.

한계: XOR 문제

데이터가 대각선 방향끼리 같은 색이라면(XOR), 직선 하나로는 절대 100% 분류할 수 없습니다. 이 경우 퍼셉트론 알고리즘은 멈추지 않고 영원히 진동(Oscillate)합니다. (나중에 이를 해결하기 위해 다층 퍼셉트론, 즉 신경망이 등장합니다.)

2.1.5 5. 퍼셉트론 수렴 정리 (Perceptron Convergence Theorem)

□ 개념 5: 언제 끝나는가?

한 줄 요약: 데이터가 선형 분리만 가능하다면, 퍼셉트론은 유한한 횟수 내에 반드시 정답을 찾습니다. 그 횟수는 "도로 폭(γ)"에 달려 있습니다.

핵심 변수

- **반경 (Radius, R):** 데이터가 원점에서 얼마나 멀리 퍼져 있는가? (Max Norm)
- **마진 (Margin, γ):** 결정 경계와 가장 가까운 데이터 사이의 거리. (도로의 폭)

정리 (Theorem)

총 실수(업데이트) 횟수 k는 다음을 넘지 못합니다.

$$k \leq \left(\frac{R}{\gamma}\right)^2$$

- **의미 1 (유한성):** 상한선이 존재하므로, 언젠가는 반드시 멈춥니다(수렴).
- **의미 2 (난이도):** 마진 γ가 작을수록(도로가 좁을수록), 횟수는 **제곱**으로 폭증합니다. 즉, 아슬아슬하게 구분되는 문제는 학습이 매우 오래 걸립니다.

2.2 실전 시나리오: 스팸 메일 필터

□ Scenario: 초기 스팸 필터

당신은 1990년대 개발자입니다. "광고"라는 단어가 들어가면 스팸(+1), 아니면 정상(−1)인 분류기를 만듭니다.

1. **초기 상태:** $\theta = 0$.
2. **데이터 1:** "회의록 송부" (x_1). 정답 -1.
3. **예측:** $\theta \cdot x_1 = 0$. (초기엔 0이라 판정 불가, 틀린 걸로 간주).
4. **업데이트:** $\theta \leftarrow \theta - x_1$. (정상 메일 벡터 방향의 반대로 θ 이동).
5. **데이터 2:** "광고: 비아그라..." (x_2). 정답 $+1$.
6. **예측:** $\theta \cdot x_2 < 0$ (아까 뺐으므로 음수). $\rightarrow$ 오답! (스팸을 정상으로 분류).
7. **업데이트:** $\theta \leftarrow \theta + x_2$. ($\theta$를 스팸 메일 벡터 쪽으로 회전).
8. **결과:** 이제 θ는 "회의록"과는 멀어지고 "광고"와는 가까워진 방향을 가리킵니다.

2.3 자주 묻는 질문 (FAQ)

Q1. 선형 분리가 불가능하면 퍼셉트론은 쓸모없나요? A. 기본 퍼셉트론은 멈추지 않아서 못 씁니다. 하지만 **'평균 퍼셉트론(Averaged Perceptron)'**을 쓰거나, 약간의 오차를 허용하는 **SVM(Support Vector Machine)**을 쓰면 해결됩니다. 혹은 데이터를 고차원으로 보내버리는 (Kernel Trick) 방법도 있습니다.

Q2. 초기값 θ가 결과에 영향을 미치나요? A. 네, 미칩니다. 해가 하나가 아니라면(도로 폭이 넓다면), 초기값에 따라 최종적으로 얻어지는 경계선이 조금씩 달라질 수 있습니다. 하지만 선형 분리 가능하다면 "분류를 성공한다"는 사실 자체는 변하지 않습니다.

> **Next Step:** 퍼셉트론 하나로는 직선밖에 못 긋습니다(XOR 문제 해결 불가). 그렇다면 **퍼셉트론을 여러 개 묶어서(Layer)** 사용하면 어떨까요? 직선들을 조합하면 곡선도 만들 수 있지 않을까요? 다음 **Unit 3**에서는 퍼셉트론을 쌓아 올린 **신경망(Neural Networks)**으로 진화합니다.

Unit 2 핵심 요약

- **구조:** $\text{sign}(\theta \cdot x)$. θ는 초평면의 법선 벡터다.
- **알고리즘:** $\theta_{new} = \theta + yx$. 틀린 데이터를 더해서 벡터를 회전시킨다.
- **수렴 정리:** 선형 분리 가능하다면 $(R/\gamma)^2$ 횟수 안에 반드시 수렴한다.
- **한계:** XOR 같이 선형 분리 불가능한 데이터는 영원히 학습하지 못한다.

Chapter 3

MIT 6.86x: 가장 안전한 경계선

Course Structure & Current Focus

- Unit 2: Linear Classifiers (틀리지만 않으면 됨)
- **Unit 3: Support Vector Machines (현재 단원: 가장 완벽하게 나누기)**
  - 3.1 Hinge Loss (확신을 갖는 학습)
  - 3.2 Margin Maximization (도로 폭 넓히기)
  - 3.3 Regularization (λ의 역할)
- Unit 4: Feature Engineering (비선형 데이터 다루기)

3.1 Unit 3. 마진 최대화와 SVM (Maximum Margin & SVM)

*Unit 2*의 퍼셉트론은 훌륭했지만, "아무 선이나" 긋는다는 치명적인 단점이 있었습니다. 절벽 끝에 서 있어도 떨어지지만 않으면 안전하다고 판단했죠. *SVM*은 다릅니다. **"절벽에서 적어도 *1m*는 떨어져 있어야 안전하다"**라고 주장합니다. 이것이 바로 마진(*Margin*)입니다.

□ 개요 (Overview)

이 단원에서는 분류 경계면과 데이터 사이의 거리를 최대화하는 **SVM(Support Vector Machine)**을 배웁니다. 이를 위해 단순한 오분류(0/1 Loss) 대신 **힌지 손실(Hinge Loss)**을 도입하여 "여유 있는 정답"을 유도하고, **정칙화(Regularization)**를 통해 이상치(Outlier)에 대처하는 유연한(Soft) 경계면을 만드는 법을 익힙니다.

□ 핵심 용어 사전

용어 (Term)	직관적 의미 (Meaning)
Margin (마진)	결정 경계와 가장 가까운 데이터 사이의 거리 (도로 폭).
Support Vectors	도로의 경계선에 딱 붙어있는 데이터들 (결정적인 기준점).
Hinge Loss	"틀리면 혼나고, 아슬아슬하게 맞춰도 혼나는" 손실 함수.
Regularization (λ)	"마진을 넓힐래, 아니면 문제를 다 맞출래?" 사이의 조절 나사.

3.1.1 1. 힌지 손실 (Hinge Loss)

□ 개념 1: 턱걸이는 인정하지 않는다

한 줄 요약: 정답을 맞췄더라도, 경계선에서 충분히 멀리 떨어져 있지 않으면(확신이 없으면) 벌점을 부과합니다.

1) 직관적 비유: 낭떠러지 운전

- **Perceptron (0/1 Loss):** 바퀴가 낭떠러지에 걸치지만 않으면 OK. (위험천만)
- **SVM (Hinge Loss):** 낭떠러지에서 최소 **1m 이상** 떨어져야 OK. 50cm 떨어져서 운전하면, 비록 떨어지진 않았지만 "위험했다"며 벌점을 줍니다.

2) 수식과 해석

$$Loss(y, z) = \max(0, 1 - yz), \quad (z = \theta \cdot x)$$

- $yz \geq 1$ **(안전):** 정답을 맞췄고 여유 공간(Margin)도 1 이상임. $\rightarrow$ **Loss = 0.**

- $0 < yz < 1$ **(불안):** 정답은 맞췄지만 경계선에 너무 붙어있음. → **Loss 발생** ($0 \sim 1$).
- $yz < 0$ **(오답):** 틀렸음. → **Loss 큼** (1 이상).

그래프 모양이 문의 경첩(Hinge)처럼 꺾여 있어서 힌지 손실이라 부릅니다.

3.1.2 2. SVM의 목적 함수 (Primal Formulation)

□ 개념 2: 도로 확장 공사

한 줄 요약: 도로의 폭(Margin)을 넓히는 것은 수학적으로 법선 벡터의 길이 $\|\theta\|$를 줄이는 것과 같습니다.

1) 기하학적 원리

점 x에서 초평면 $\theta \cdot x = 0$까지의 거리는 $\frac{|\theta \cdot x|}{\|\theta\|}$ 입니다. 우리가 "안전 기준"을 1로 잡았으므로 ($|\theta \cdot x| \geq 1$), 마진 거리는 최소 $\frac{1}{\|\theta\|}$ 가 됩니다.

- 마진 최대화 ($\max \frac{1}{\|\theta\|}$) $\iff$ **벡터 길이 최소화** ($\min \|\theta\|^2$)

2) 최적화 문제 (Hard Margin)

$$\min_{\theta} \frac{1}{2}\|\theta\|^2$$

$$\text{subject to } y^{(i)}(\theta \cdot x^{(i)} + \theta_0) \geq 1 \quad (\forall i)$$

이 식은 2차 함수(Quadratic)이므로, 오목/볼록이 확실하여 **유일한 해(Global Optimum)**가 존재합니다.

3.1.3 3. 정칙화 (Regularization)와 Soft Margin

□ 개념 3: 융통성 있는 보안관

한 줄 요약: 완벽한 분리가 불가능할 때, "도로를 침범하는 것"을 일부 허용하되 벌금을 물려서 균형을 맞춥니다.

1) 현실의 문제 (Noise)

데이터에 노이즈가 섞여 있다면, 완벽하게 두 그룹을 나누는 직선은 존재하지 않거나 아주 구불구불한 억지스러운 선이 됩니다(과적합).

2) 해결책: Soft Margin SVM

$$J(\theta) = \underbrace{\frac{\lambda}{2}\|\theta\|^2}_{\text{마진 최대화}} + \underbrace{\frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y^{(i)}(\theta \cdot x^{(i)}))}_{\text{에러 최소화 (힌지 손실)}}$$

3) λ의 역할 (Trade-off 조절)

λ는 "마진(안전)"을 얼마나 중요하게 생각하는지 나타내는 파라미터입니다.

- **λ가 큼 (Regularization 중시):** "에러가 좀 나더라도 도로를 넓게 써라." → 단순한 모델, 과소적합(Underfitting) 주의.
- **λ가 작음 (Error 최소화 중시):** "도로가 좁아져도 좋으니 데이터 하나하나 다 맞춰라." → 복잡한 모델, 과적합(Overfitting) 주의.

3.2 실전 시나리오: 자율주행 차선 인식

□ Scenario: 넥슨 카트라이더 자율주행 모드

당신은 카트라이더의 AI 주행 보조 시스템을 개발합니다. 트랙 위에서 "주행 가능 구역(Positive)"과 "벽/장애물(Negative)"을 구분해야 합니다.

1. **Perceptron 적용 시:** 경계선을 벽에 딱 붙어서 긋습니다. 조금만 미끄러지면 바로 충돌 사고가 납니다. (불안정)
2. **SVM 적용 시:** 벽에서 최대한 멀리 떨어진 중앙선(Margin 최대화)을 찾습니다. 약간 미끄러져도(Noise) 충돌하지 않습니다. (안정적)
3. **Soft Margin 상황:** 트랙 중간에 바나나 껍질(Outlier)이 하나 떨어져 있습니다.
   - **Hard Margin:** 바나나를 피하려고 핸들을 급격히 꺾다가 오히려 사고가 납니다.
   - **Soft Margin:** "저건 그냥 이상치야"라고 무시하고(Loss 감수), 부드럽게 밟고 지나가면서 전체적인 주행 라인을 유지합니다.

3.3 자주 묻는 질문 (FAQ)

Q1. 왜 마진을 최대화하는데 $\|\theta\|$를 최소화하나요? A. 마진 공식이 $\frac{1}{\|\theta\|}$이기 때문입니다. 분모가 작아져야 전체 값이 커집니다. 기하학적으로는 θ 벡터의 길이가 짧아질수록, 결정 경계면($\theta \cdot x = 1$)이 원점에서 멀어지기 때문입니다.

Q2. 힌지 손실의 미분은 어떻게 하나요? 꺾인 점이 있잖아요. A. 맞습니다. $z = 1$인 지점에서는 미분 불가능합니다. 그래서 **'서브그라디언트(Subgradient)'**라는 개념을 사용합니다. 꺾인 점에서는 미분값을 0과 -1 사이의 임의의 값(보통 둘 중 하나)으로 정의해서 넘어갑니다.

Next Step: SVM은 직선(초평면)으로 데이터를 나누는 최강의 도구입니다. 하지만 데이터가 도넛 모양이나 소용돌이 모양이라면요? 직선으로는 절대 못 나눕니다. 다음 **Unit 4**에서는 데이터를 고차원으로 보내서 휘어진 경계면을 만드는 마법, **커널(Kernel) 기법**과 특징 공학(Feature Engineering)을 배웁니다.

Unit 3 핵심 요약

- **목표:** 단순 분류가 아니라, 가장 안전한 분류(Max Margin)를 한다.
- **Hinge Loss:** $\max(0, 1 - yz)$. 1 이상의 여유를 갖고 맞추도록 강제한다.
- **Primal Objective:** $\min \frac{\lambda}{2}\|\theta\|^2 + \text{Loss}$. 마진과 에러의 균형을 찾는다.
- **λ:** 모델의 단순함(마진)과 정확함(에러) 사이의 트레이드오프를 조절한다.

Chapter 4

MIT 6.86x: 가장 낮은 곳을 향하여

Course Structure & Current Focus

- Unit 2, 3: Modeling (목적 함수 $J(\theta)$ 정의)
- **Unit 4: Optimization (현재 단원: $J(\theta)$를 최소화하는 θ^* 찾기)**
  - 4.1 Gradient Descent (안개 낀 산 내려오기)
  - 4.2 Stochastic Gradient Descent (빠르고 거친 이동)
  - 4.3 Convergence & Learning Rate (멈추는 기술)
- Unit 5: Non-linear Classification (Neural Nets)

4.1 Unit 4. 최적화 방법론 (Optimization)

*Unit 3*에서 우리는 "마진을 최대화하라"는 멋진 목표(목적 함수)를 세웠습니다. 하지만 목표만 있다고 문제가 풀리지는 않습니다. 서울(현재 파라미터)에서 부산(최적 파라미터)을 가야 하는데, 지도도 없고 나침반(기울기)만 하나 덜렁 있는 상황입니다. 이번 단원에서는 이 나침반 하나에 의지해 가장 낮은 계곡을 찾아가는 알고리즘을 배웁니다.

□ 개요 (Overview)

이 단원에서는 손실 함수 $J(\theta)$의 최솟값을 찾기 위한 두 가지 핵심 알고리즘, **경사 하강법(GD)**과 **확률적 경사 하강법(SGD)**을 다룹니다. 특히 딥러닝의 표준인 SGD가 왜 수학적으로 정당한지(Unbiased Estimator), 그리고 진동하는 SGD를 어떻게 수렴시킬 것인지(Learning Rate Decay)에 대한 이론적 배경을 학습합니다.

□ 핵심 용어 사전

용어 (Term)	직관적 의미 (Meaning)
Gradient (∇J)	현재 위치에서 가장 가파르게 올라가는 방향. (우리는 반대로 감)
Learning Rate (η)	한 걸음의 보폭. 너무 크면 발산하고 너무 작으면 느리다.
Batch GD	한 걸음 갈 때마다 모든 데이터를 다 읽고 신중하게 가는 방법.
SGD	데이터 하나만 보고 "이쪽인 것 같아!" 하고 뛰어가는 방법.
Epoch	전체 데이터를 한 번 훑어보는 단위.

4.1.1 1. 경사 하강법 (Gradient Descent, GD)

□ 개념 1: 안개 낀 산에서 내려오기

한 줄 요약: 현재 위치에서 발밑을 더듬어 경사가 가장 급한 쪽으로 한 걸음씩 내려갑니다.

1) 알고리즘

전체 데이터셋 S_n에 대한 평균 손실 함수의 기울기를 계산합니다.

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta_t)$$

- **마이너스(−)의 의미:** 기울기는 '올라가는' 방향이므로, 내려가기 위해 반대로 이동합니다.
- **수학적 원리:** 테일러 급수 1차 근사(선형 근사)에 의해, η가 충분히 작다면 함수값 감소가 보장됩니다.

2) 치명적인 단점 (Computational Bottleneck)

한 걸음($t \to t+1$)을 떼기 위해 n개의 데이터를 모두 미분해야 합니다.

- $n = 1,000,000$이면, 1보 전진을 위해 100만 번 계산해야 합니다. ($O(n)$ Cost)
- 너무 느려서 빅데이터 학습에는 사용할 수 없습니다.

4.1.2 2. 확률적 경사 하강법 (Stochastic Gradient Descent, SGD)

□ 개념 2: 술 취한 사람의 귀가

한 줄 요약: 전체의 의견을 듣지 않고, 지나가는 사람(데이터) 한 명 붙잡고 길을 물어서 갑니다. 비틀거리지만(Noisy), 결국 집(최적해)으로 갑니다.

1) 아이디어

"기울기의 평균을 구하나, 샘플 하나만 뽑아서 구하나, 기댓값은 같지 않을까?"

2) 업데이트 규칙

랜덤하게 데이터 i를 하나 뽑아서 업데이트합니다.

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta Loss(x^{(i)}, y^{(i)}; \theta_t)$$

- **비용:** $O(n) \to O(1)$로 획기적 감소.
- **속도:** GD가 1걸음 갈 때, SGD는 n걸음을 갈 수 있습니다.

3) 수학적 정당성 (Unbiased Estimator)

개별 데이터의 기울기는 전체 기울기의 **비편향 추정량(Unbiased Estimator)**입니다.

$$\mathbb{E}[\nabla Loss(x^{(i)})] = \nabla J(\theta)$$

즉, 한 번 한 번은 엉뚱한 방향일 수 있어도(High Variance), 평균적으로는 올바른 방향을 가리킵니다.

4.1.3 3. SGD의 수렴성과 학습률 스케줄링

□ 개념 3: 멈추는 법을 배워라

한 줄 요약: GD는 바닥에 오면 알아서 멈추지만, SGD는 바닥에서도 계속 진동합니다. 따라서 갈수록 보폭(η)을 줄여야 합니다.

1) GD vs SGD의 경로 비교

- **GD:** 등고선의 수직 방향으로 부드러운 곡선을 그리며 최솟값에 안착.
- **SGD:** 지그재그로 요동치며(Fluctuation) 이동. 최솟값 근처에서도 멈추지 않고 주변을 맴돔.

2) 로빈스-몬로 조건 (Robbins-Monro Conditions)

SGD가 수학적으로 수렴하기 위한 학습률 η_t의 조건입니다.

1. $\sum \eta_t = \infty$: **(도달 가능성)** 보폭의 합은 무한대여야 합니다. (시작점이 아무리 멀어도 갈 수 있어야 함).
2. $\sum \eta_t^2 < \infty$: **(수렴 가능성)** 보폭 제곱의 합은 유한해야 합니다. (점점 줄어들어 진동이 멈춰야 함).

대표적인 스케줄: $\eta_t = \frac{1}{t}$ 또는 $\eta_t = \frac{C}{t+\text{offset}}$.

4.2 실전 시나리오: 넷플릭스 추천 시스템 학습

□ Scenario: 수억 명의 시청 기록 학습

당신은 전 세계 2억 명의 유저 데이터를 이용해 추천 알고리즘을 학습시킵니다.

1. **Batch GD 시도:** 파라미터를 한 번 업데이트하려면 2억 개의 로그를 다 읽어야 합니다. → 1 Epoch 도는 데 일주일 걸림. 학습 불가.
2. **SGD 적용:** 유저가 영상을 클릭할 때마다 즉시 그 데이터 하나로 파라미터를 미세 조정합니다. → 실시간으로 모델이 학습되며, 1시간 만에 괜찮은 성능에 도달.
3. **문제 발생 (진동):** 학습 후반부에 정확도가 오르락내리락하며 불안정합니다.
4. **해결 (Learning Rate Decay):** 학습률을 처음엔 0.1로 하다가, 에폭마다 0.1배씩 줄여나갑니다(Annealing). → 진동이 잦아들며 최고 성능(Global Minima 근처)에 안착.

4.3 자주 묻는 질문 (FAQ)

Q1. SGD가 GD보다 정확도는 떨어지나요? A. 이론적으로는 GD가 더 정밀하게 최솟값을 찾을 수 있습니다. 하지만 SGD의 '노이즈'가 오히려 장점이 되기도 합니다. GD는 얕은 함정(Local Minima)에 빠지면 못 나오지만, SGD는 비틀거리는 특성 덕분에 함정을 탈출하여 더 좋은 해(Global Minima)를 찾을 확률이 높습니다.

Q2. Mini-batch는 뭔가요? A. GD(전체)와 SGD(1개)의 절충안입니다. 데이터 32개나 64개씩 묶어서 업데이트하는 방식입니다.

- SGD보다 노이즈가 적어 안정적이고,
- 행렬 연산(GPU)을 활용해 속도도 빠릅니다.
- 현대 딥러닝은 99% 이 **Mini-batch SGD**를 사용합니다.

> **Next Step:** 최적화라는 엔진을 달았으니, 이제 더 복잡한 모델을 만들어볼까요? 직선(선형 분류기)으로는 XOR 문제를 풀 수 없습니다. 다음 **Unit 5**에서는 선형 분류기를 여러 층 쌓아서 곡선 경계를 만드는 **신경망(Neural Networks)**과 딥러닝의 기초를 배웁니다.

Unit 4 핵심 요약

- **Gradient Descent (GD):** 전체 데이터 사용. 안정적이지만 $O(n)$으로 느리다.
- **Stochastic GD (SGD):** 데이터 1개 사용. $O(1)$로 빠르지만 불안정(Noisy)하다.
- **Unbiased Estimator:** SGD의 기댓값은 GD와 같다. (평균적으로는 바른 방향).
- **Step Size Schedule:** SGD의 진동을 잡기 위해 학습률 η_t를 점점 줄여야 한다(Decay).

Chapter 5

MIT 6.86x: 데이터를 조각하는 기술

Course Structure & Current Focus

- Unit 2 (Part A): Linear Classifiers (직선으로 자르기)
- **Unit 2 (Part B): Feature Engineering (현재 단원: 데이터 차원 확장)**
  - 2.5 Non-linear Separability (직선의 한계)
  - 2.6 Feature Maps $\phi(x)$ (차원 확장)
  - 2.7 Polynomial Features (다항식 변환)
  - 2.8 Curse of Dimensionality (차원의 저주)
- Unit 3: Neural Networks (특징을 스스로 학습하기)

5.1 Unit 2 (Part B). 특징 공학 (Feature Engineering)

우리는 퍼셉트론과 *SVM*을 통해 최적의 '직선'을 긋는 법을 배웠습니다. 하지만 세상의 데이터가 모두 직선으로 깔끔하게 나뉠까요? 만약 데이터가 **도넛 모양**으로 분포한다면 직선 하나로 절대 자를 수 없습니다. 이 한계를 극복하기 위해 우리는 칼(모델)을 바꾸는 대신, **데이터의 모양을 바꾸는(*Transformation*)** 전략을 취합니다.

□ 개요 (Overview)

이 단원은 선형 분리가 불가능한 데이터를 해결하기 위해 입력 데이터 x를 고차원 특징 공간 $\phi(x)$로 매핑하는 방법을 배웁니다. 특히 **다항식 특징(Polynomial Features)**을 통해 곡선 경계를 만드는 원리와, 차원이 너무 높아질 때 발생하는 **계산 비용 및 과적합 문제(Trade-off)**를 다룹니다.

□ 핵심 용어 사전

용어 (Term)	직관적 의미 (Meaning)
Non-linear Separability	직선 하나로 100% 분류가 불가능한 상태 (예: XOR, 도넛).
Feature Map (ϕ)	데이터를 저차원에서 고차원으로 옮기는 함수.
Feature Space	매핑된 데이터들이 살고 있는 고차원 세상.
Polynomial Feature	x^2, x_1x_2 처럼 곱하기를 통해 만든 새로운 특징.
Linear in Feature Space	고차원에서는 평면으로 잘리지만, 원래 세상에선 곡선으로 보임.

5.1.1 1. 비선형성 (Non-linearity)의 필요성

□ 개념 1: 직선의 한계

한 줄 요약: 동그라미 안에 있는 점과 밖에 있는 점은 자 하나로 가를 수 없습니다. 가위(비선형 모델)가 필요하거나, 종이를 접어야(특징 변환) 합니다.

문제 상황: XOR과 도넛

- **XOR:** $(0,0), (1,1)$은 파란색, $(0,1), (1,0)$은 빨간색. 대각선 관계.
- **Donut:** 원점 $(0,0)$ 근처는 빨간색, 그 바깥 고리 모양은 파란색.

이런 데이터에 퍼셉트론(선형 분류기)을 적용하면, 오차가 줄어들지 않고 계속 진동(Oscillation)하며 실패합니다.

5.1.2 2. 특징 변환 (Feature Mapping)

□ 개념 2: 차원 띄우기 (Lifting)

한 줄 요약: 2차원 바닥에 흩어진 구슬들을 3차원 공중으로 띄우면, 그 사이로 판(평면)을 끼워 넣을 수 있습니다.

1) 직관적 비유

도넛 모양 데이터를 상상해 봅시다.

- 가운데 빨간 구슬들 (원점 근처)
- 바깥쪽 파란 구슬들 (반지름이 큼)

여기에 새로운 축(고도) $z = x_1^2 + x_2^2$를 추가합니다.

- 빨간 구슬: 원점 근처라 z 값이 작음 $\rightarrow$ 바닥에 깔림.
- 파란 구슬: 원점에서 멀어서 z 값이 큼 $\rightarrow$ 공중으로 떠오름.
- **해결:** 이제 바닥과 공중 사이에 **수평 판(Hyperplane)**을 끼워 넣으면 완벽하게 분리됩니다!

2) 수학적 정의

입력 벡터 $x \in \mathbb{R}^d$를 고차원 벡터 $\phi(x) \in \mathbb{R}^D$로 변환합니다 ($D > d$).

$$\theta \cdot x + \theta_0 \quad \xrightarrow{\text{Transformation}} \quad \theta' \cdot \phi(x) + \theta'_0$$

5.1.3 3. 다항식 특징 (Polynomial Features)

□ 개념 3: 곱셈으로 특징 만들기

한 줄 요약: 원래 변수들을 서로 곱해서 (x_1^2, x_1x_2) 새로운 정보를 만들어냅니다.

1) 2차 다항식 (Quadratic Features)

입력 $x = [x_1, x_2]^T$ 일 때, 2차항까지 모두 만들면:

$$\phi(x) = [1, x_1, x_2, x_1^2, x_2^2, x_1x_2]^T$$

- 원래 차원: 2차원
- 확장된 차원: 6차원

2) 기하학적 의미

확장된 공간에서의 선형 결합 $\theta^T\phi(x) = 0$은 원래 공간(x_1, x_2)에서 보면 다음과 같습니다.

$$w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2 + w_5x_1x_2 = 0$$

이것은 원, 타원, 쌍곡선 같은 **2차 곡선(Conic Section)**의 방정식입니다. 즉, **"고차원에서의 평면 = 저차원에서의 곡선"**입니다.

5.1.4 4. 장점과 주의점 (The Trade-off)

□ 개념 4: 공짜 점심은 없다

한 줄 요약: 표현력(Power)이 좋아지지만, 그만큼 계산 비용과 과적합(Overfitting) 위험이 폭증합니다.

1) 차원의 저주 (Curse of Dimensionality)

차수(Degree) Q를 높이면 특징의 개수 D가 기하급수적으로 늘어납니다.

$$D \approx d^Q$$

- 입력 변수가 100개$(d = 100)$인데 3차 다항식$(Q = 3)$을 쓰면? 특징 개수가 약 **100만 개**가 됩니다.
- **계산 문제:** 100만 차원 벡터의 내적을 계산해야 함 $\rightarrow$ 느려짐.
- **통계적 문제:** 변수가 100만 개인데 데이터가 1만 개라면? $\rightarrow p \gg n$ 문제 발생 $\rightarrow$ **과적합(Overfitting)**.

커널 트릭 (Kernel Trick) 예고

"특징 개수가 100만 개라도, 실제로 $\phi(x)$를 계산하지 않고 내적값 $\phi(x) \cdot \phi(y)$만 빠르게 구할 수 있다면?" 이것이 다음 단원에서 배울 **SVM 커널 트릭**의 핵심 아이디어입니다.

5.2 실전 시나리오: 불법 프로그램(핵) 사용자 탐지

□ Scenario: 넥슨 FPS 게임 '서든어택'의 에임봇 탐지

당신은 보안팀입니다. 유저의 마우스 움직임 데이터를 분석해 '에임봇(Auto-aim)' 사용자를 잡고 싶습니다.

1. **데이터:** x_1 (조준 속도), x_2 (정확도).

2. **분포 확인:**
   - 일반 유저: 속도가 빠르면 정확도가 낮고, 느리면 높음 (반비례). 넓게 퍼져 있음.
   - 핵 유저: **속도가 엄청 빠르면서 동시에 정확도도 100%임**. 특정 영역(우상단)에 몰려 있음.
3. **선형 분류 시도:** 직선을 그어보지만, 고수 유저(빠르고 정확함)와 핵 유저가 섞여서 구분이 안 됩니다.
4. **다항식 특징 추가:** x_1^2, x_2^2, x_1x_2를 추가합니다.
   - 특히 $x_1 \times x_2$ **(속도 × 정확도)** 라는 교호작용 항(Interaction Term)이 핵심입니다.
   - 일반 유저는 이 값이 일정 한계를 넘지 못하지만(인간의 한계), 핵 유저는 이 값이 비정상적으로 높습니다.
5. **결과:** 고차원 공간에서 핵 유저 그룹만 뚝 떨어져 나와, 평면 하나로 깔끔하게 검거(Classification)할 수 있게 되었습니다.

5.3 자주 묻는 질문 (FAQ)

Q1. 차수는 몇 차(Degree)까지 늘려야 하나요? A. 정답은 없습니다. 보통 2차나 3차 정도만 써도 충분한 경우가 많습니다. 너무 높이면 훈련 데이터는 다 맞추지만(Overfitting), 새로운 데이터는 다 틀리는 괴물이 탄생합니다. 교차 검증(Cross Validation)으로 적절한 차수를 찾아야 합니다.

Q2. 요즘 딥러닝은 특징 공학을 안 한다던데요? A. 맞습니다! 딥러닝(Neural Networks)의 가장 큰 장점은, 사람이 손으로 $\phi(x)$를 설계하는 대신, **네트워크가 스스로 최적의 특징 $\phi(x)$를 찾아낸다(Representation Learning)**는 점입니다. 다음 Unit 3에서 바로 그 내용을 배웁니다.

> **Next Step:** 우리는 사람이 직접 $\phi(x) = x^2$ 처럼 식을 정해주는 방식을 배웠습니다. 그런데 이미지나 음성처럼 복잡한 데이터는 어떤 식을 써야 할지 감도 안 옵니다. 컴퓨터가 스스로 $\phi(x)$를 만들게 할 수는 없을까요? 다음 **Unit 3: 신경망(Neural Networks)**에서는 퍼셉트론을 여러 겹 쌓아 스스로 특징을 학습하는 딥러닝의 기초를 배웁니다.

Unit 2 (Part B) 핵심 요약

- **문제:** 현실 데이터는 직선 하나로 나뉘지 않는다 (Non-linear).
- **해결:** 데이터를 고차원으로 매핑($\phi(x)$)하면 선형 분리가 가능해진다.
- **방법:** 다항식 특징$(1, x, x^2, \dots)$이 가장 대표적이다.
- **대가:** 차원이 늘어나면 계산량이 폭증하고 과적합 위험이 생긴다.
- **의의:** 딥러닝 이전에는 사람이 이 특징(ϕ)을 찾는 것이 머신러닝의 전부였다.

Chapter 6

MIT 6.86x: 차원의 저주를 피하는 지름길

Course Structure & Current Focus

- Unit 3: SVM (최적의 직선 찾기)
- Unit 5: Feature Engineering (고차원 변환 $\phi(x)$)
- **Unit 6: Kernel Methods (현재 단원: 계산 비용 없이 고차원 효과 내기)**
  - 6.1 Dual Problem (내적의 발견)
  - 6.2 The Kernel Trick (지름길)
  - 6.3 RBF Kernel (무한 차원의 마법)
  - 6.4 Kernel SVM (비선형 분류의 완성)
- Unit 7: Neural Networks (딥러닝)

6.1 Unit 6. 커널 방법 (Kernel Methods)

*Unit 5*에서 우리는 데이터를 고차원으로 매핑($\phi(x)$)하면 선형 분리가 가능해진다는 것을 배웠습니다. 하지만 *100*차원의 데이터를 *2*차 다항식으로만 확장해도 특징이 수천 개로 늘어나고, 계산 속도는 거북이가 됩니다. **"고차원의 혜택은 누리면서, 계산은 저차원에서 빠르게 할 수는 없을까?"** 이 말도 안 되는 욕심을 수학적으로 실현한 것이 바로 커널 트릭입니다.

□ 개요 (Overview)

커널 방법은 SVM의 최적화 문제를 **듀얼(Dual)** 형태로 변형하여, 데이터가 오직 **내적(Dot Product)** 형태로만 등장한다는 점을 이용합니다. 이를 통해 데이터를 실제로 고차원으로 변환하지 않고도 고차원 공간에서의 내적값을 계산해주는 **커널 함수(Kernel Function)**를 도입, 계산 비용을 획기적으로 줄이면서도 복잡한 비선형 경계면을 만들어냅니다.

□ 핵심 용어 사전

용어 (Term)	직관적 의미 (Meaning)
Primal Problem	원래 문제. w, b를 찾자. 데이터 좌표 x가 직접 쓰임.
Dual Problem	쌍대 문제. α(가중치)를 찾자. 데이터 간 유사도(내적)만 쓰임.
Kernel Trick	고차원 계산 결과를 저차원에서 즉시 알아내는 마법의 함수.
RBF Kernel	두 점 사이의 거리를 이용해 유사도를 측정하는 커널 (가우시안).
Feature Space	실제로 가진 않지만, 수학적으로 존재하는 가상의 고차원 공간.

6.1.1 1. 듀얼 문제 (Dual Problem)와 내적의 발견

□ 개념 1: 관점의 전환

한 줄 요약: "데이터의 절대적 위치(좌표)"는 중요하지 않습니다. "데이터끼리 얼마나 닮았나(내적)"가 중요합니다.

1) Primal vs Dual

SVM의 목적식을 라그랑주 승수법을 이용해 변형하면 다음과 같은 **듀얼 문제**가 됩니다.

$$\max_{\alpha} \sum \alpha_i - \frac{1}{2} \sum_{i,j} y^{(i)} y^{(j)} \alpha_i \alpha_j (\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)})$$

2) 핵심 발견

위 식을 자세히 보세요. 데이터 x가 단독으로 쓰이지 않고, 반드시 **두 데이터의 내적** $(\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)})$ **형태**로만 등장합니다.

- 내적(Dot Product)은 기하학적으로 **유사도(Similarity)**를 의미합니다.
- 즉, 분류 경계선을 찾기 위해 필요한 것은 좌표가 아니라 **유사도 정보**뿐입니다.

6.1.2 2. 커널 트릭 (Kernel Trick)

□ 개념 2: 웜홀(Wormhole) 타기

한 줄 요약: $\phi(x)$를 계산해서 고차원으로 날아간 뒤 내적하고 다시 돌아오는 대신, 그냥 땅바닥에서 $K(x, z)$만 계산하면 똑같은 결과가 나옵니다.

1) 문제 상황 (The Hard Way)

데이터 x, z를 고차원으로 보낸 뒤 내적하려면:

1. 변환: $x \to \phi(x)$, $z \to \phi(z)$ (엄청난 연산 비용)
2. 내적: $\phi(x) \cdot \phi(z)$ (엄청난 연산 비용)

2) 해결책 (The Smart Way)

다음 성질을 만족하는 함수 K를 찾습니다.

$$K(x, z) = \phi(x) \cdot \phi(z)$$

이제 우리는 ϕ를 몰라도, K만 계산하면 됩니다.

- 예: 다항식 커널 $K(x, z) = (x \cdot z + 1)^2$
- 이걸 전개해보면 실제로 2차 다항식 특징들의 내적과 수학적으로 동일함이 증명됩니다. 하지만 계산은 단순히 **숫자 곱셈 한 번**이면 끝납니다.

6.1.3 3. 대표적인 커널: 가우시안 (RBF) 커널

□ 개념 3: 무한 차원의 마법

한 줄 요약: 데이터 주변에 산봉우리(유사도)를 세워서 지형을 만듭니다. 이는 수학적으로 데이터를 **무한대 차원**으로 보낸 것과 같습니다.

1) 수식

$$K(x, z) = \exp\left(-\gamma \|x - z\|^2\right)$$

- **의미:** 두 점 x, z 사이의 **유클리드 거리**가 가까우면 1, 멀면 0에 수렴. (유사도 측정)
- γ **(Gamma):** 산봉우리의 뾰족한 정도. γ가 크면 아주 뾰족한 산(Overfitting 위험), 작으면 완만한 언덕.

2) 왜 무한 차원인가?

지수 함수 e^x를 테일러 급수(Taylor Series)로 전개하면 $1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$ 처럼 무한한 다항식의 합이 됩니다. 즉, RBF 커널을 쓴다는 것은 **모든 차수의 다항식 특징을 다 고려한다**는 뜻과 같습니다.

6.1.4 4. 커널 SVM (Kernel SVM)

□ 개념 4: 휘어지는 경계선

한 줄 요약: 고차원에서는 평면으로 잘랐지만, 이를 다시 원래 차원으로 투영해보면 꼬불꼬불한 곡선이 되어 있습니다.

작동 원리

1. 데이터는 선형 분리 불가능한 복잡한 형태입니다.
2. 커널 함수 K를 사용하여 듀얼 문제를 풉니다. (실제 변환 없이 고차원 내적 값만 사용)
3. 최적의 α(가중치)를 찾습니다.
4. 새로운 데이터 u가 들어오면 판별 함수는 다음과 같습니다:

$$h(u) = \text{sign}\left(\sum_{i \in SV} \alpha_i y^{(i)} K(x^{(i)}, u) + b\right)$$

5. **해석:** 새로운 데이터 u가 기존의 중요한 데이터들(Support Vectors)과 얼마나 유사한지(K)를 측정하고, 가중치 투표를 통해 분류합니다.

6.2 실전 시나리오: 넥슨 게임 봇(Bot) 탐지

□ Scenario: 인간과 봇의 움직임 구분

당신은 '던전앤파이터'의 보안팀입니다. 봇들의 마우스 이동 경로를 분석하여 차단하려 합니다.

1. **데이터:** 마우스의 (x, y) 좌표 이동 궤적.
2. **특징:**
   - 봇: 기계적으로 정확하게 움직이지만, 가끔 랜덤 노이즈를 섞어서 선형적으로 구분이 안 됨.
   - 인간: 불규칙하지만 자연스러운 곡선을 그림.
3. **선형 SVM 실패:** 직선으로는 복잡한 궤적 패턴을 나눌 수 없음.
4. **다항식 특징 시도:** $x^2, y^2, xy \dots$ 몇 차까지 늘려야 할지 모르겠고 계산이 너무 느림.
5. **RBF 커널 SVM 적용:**
   - "특정 봇 패턴($x^{(i)}$)과 거리가 가까우면(유사하면) 봇이다"라는 논리로 접근.
   - 무한 차원의 특징을 고려하므로 아주 미세하고 복잡한 '봇만의 영역'을 곡선으로 도려내듯 찾아냄.
6. **결과:** 봇 탐지율 99% 달성. (단, γ 튜닝이 중요했음).

6.3 자주 묻는 질문 (FAQ)

Q1. 어떤 커널을 써야 할지 어떻게 아나요? A. 정답은 없지만, **RBF 커널**이 가장 무난하고 강력해서 기본값(Default)으로 쓰입니다. 텍스트 데이터처럼 희소(Sparse)한 경우에는 선형 커널(Linear Kernel)이 더 좋을 때도 있습니다. 결국 교차 검증(Cross Validation)으로 찾아야 합니다.

Q2. RBF 커널은 무한 차원이라는데 과적합 안 되나요? A. 될 수 있습니다! 그래서 규제 파라미터 C **(Cost)**와 커널 파라미터 γ **(Gamma)**를 잘 조절해야 합니다.

- γ가 크면: 내 주변 데이터만 신경 씀 $\rightarrow$ 섬세한 경계 $\rightarrow$ 과적합.
- γ가 작으면: 멀리 있는 데이터까지 신경 씀 $\rightarrow$ 뭉뚱그린 경계 $\rightarrow$ 과소적합.

> **Next Step:** 커널 SVM은 강력하지만, 데이터가 많아지면($n > 100,000$) 내적 계산량이 많아져서 느려집니다. 이 한계를 극복하고, 빅데이터에서도 복잡한 패턴을 스스로 학습하는 끝판왕 모델, **Unit 7: 신경망(Neural Networks)과 딥러닝**으로 넘어갑니다.

Unit 6 핵심 요약

- **Dual Problem:** SVM은 데이터의 좌표가 아니라 '내적(유사도)'만 있으면 풀 수 있다.
- **Kernel Trick:** $K(x, z) = \phi(x) \cdot \phi(z)$. 고차원 변환 없이 고차원 내적값을 계산하는 지름길.
- **RBF Kernel:** 거리 기반 유사도 측정. 무한 차원으로 매핑하는 효과를 낸다.
- **장점:** 저차원의 계산 비용으로 고차원의 분류 성능을 얻는다.

Chapter 7

MIT 6.86x: 뇌를 모방하다

Course Structure & Current Focus

- Unit 2: Linear Classifiers (직선 하나 긋기)
- **Unit 3: Neural Networks (현재 단원: 선을 쌓아 면을 만들기)**
  - 7.1 Feedforward Neural Networks (구조)
  - 7.2 The Role of Non-linearity (활성화 함수의 필요성)
  - 7.3 Activation Functions (Sigmoid, Tanh, ReLU)
  - 7.4 Output Layers (Softmax)
- Unit 4: Backpropagation (학습의 원리)

7.1 Unit 3 (Part A). 신경망 기초 (Neural Network Basics)

*Unit 2*에서 우리는 *XOR* 문제처럼 직선 하나로 풀 수 없는 문제를 해결하기 위해, 사람이 직접 x^2, xy 같은 특징을 만들어줬습니다(*Feature Engineering*). 하지만 이미지나 음성 데이터에서 어떤 특징이 중요한지 사람이 어떻게 알 수 있을까요? 신경망은 "**선형 모델을 층층이 쌓으면 컴퓨터가 알아서 특징을 찾아낸다**"는 아이디어에서 출발합니다.

□ 개요 (Overview)

이 단원에서는 딥러닝의 가장 기본 구조인 **피드포워드 신경망(FNN)**을 배웁니다. 입력층-은닉층-출력층으로 이어지는 데이터의 흐름을 이해하고, 단순한 선형 결합을 강력한 비선형 모델로 바꿔주는 핵심 부품인 **활성화 함수(Activation Function)**의 종류와 역할을 학습합니다.

□ 핵심 용어 사전

용어 (Term)	직관적 의미 (Meaning)
FNN (Feedforward NN)	신호가 앞으로만 전달되는 신경망. (Loop 없음)
Hidden Layer (은닉층)	입력과 출력 사이에 숨어서 '특징'을 추출하는 층.
Activation Function	뉴런을 켤지 끌지 결정하는 스위치. 비선형성을 부여함.
ReLU	양수는 통과, 음수는 차단. 딥러닝의 표준 스위치.
Softmax	출력값들을 '확률(합=1)'로 변환해주는 함수.

7.1.1 1. 피드포워드 신경망 (FNN)

□ 개념 1: 정보의 조립 라인

한 줄 요약: 데이터가 공장 컨베이어 벨트(Layer)를 지나가면서 점점 더 추상적이고 고급진 정보로 가공되어 출력됩니다.

1) 구조와 흐름

$$\text{Input} \xrightarrow{W_1} \text{Hidden 1} \xrightarrow{W_2} \text{Hidden 2} \xrightarrow{W_3} \text{Output}$$

- **방향성:** 입력에서 출력으로 한 방향(Forward)으로만 흐릅니다. (순환 시 RNN)
- **각 층의 연산:** 선형 변환($Wx + b$) 후 비선형 함수(σ) 통과.

$$a^{[l]} = \sigma(W^{[l]}a^{[l-1]} + b^{[l]})$$

2) Deep Learning이란?

은닉층(Hidden Layer)이 2개 이상 깊게(Deep) 쌓이면 심층 신경망, 즉 딥러닝이라고 부릅니다. 은닉층이 많을수록 데이터의 복잡한 패턴을 더 잘 이해할 수 있습니다.

7.1.2 2. 활성화 함수가 왜 필요한가?

□ 개념 2: 선형 + 선형 = 선형

한 줄 요약: 활성화 함수(비선형)가 없으면, 신경망을 100층을 쌓아도 그냥 1층짜리 선형 회귀랑 똑같아집니다.

수학적 증명 (간단 버전)

만약 활성화 함수 σ가 없다면 ($y = x$):

- Layer 1: $h = W_1 x$
- Layer 2: $y = W_2 h = W_2(W_1 x) = (W_2 W_1)x = W_{new} x$

결국 행렬 곱셈 한 번 한 것과 차이가 없습니다. 신경망이 **곡선**을 그리고 **복잡한 경계**를 만들기 위해서는 반드시 비선형 함수(꺾임, 곡선)를 중간에 끼워 넣어야 합니다.

7.1.3 3. 주요 활성화 함수 (The Big 4)

1) Sigmoid ($\sigma(z) = \frac{1}{1+e^{-z}}$)

- **특징:** 입력을 $0 \sim 1$로 압축. S자 곡선.
- **용도:** 이진 분류(Binary Classification)의 **출력층**.
- **치명적 단점:** 입력값이 크거나 작으면 기울기가 0이 되어 학습이 멈춤 (**Vanishing Gradient**). 은닉층에는 잘 안 씀.

2) Tanh ($\tanh(z)$)

- **특징:** 입력을 $-1 \sim 1$로 압축. 0을 중심(Zero-centered)으로 함.
- **용도:** Sigmoid보다는 낫지만, 여전히 깊은 층에서는 기울기 소실 문제 발생.

3) ReLU (Rectified Linear Unit, $f(z) = \max(0, z)$)

- **특징:** 양수는 그대로, 음수는 0.
- **용도: 은닉층의 표준(Default).**

- **장점:**
  1. 계산이 초고속 (비교 연산 하나면 끝).
  2. 양수 구간에서 기울기가 1이라서, 깊게 쌓아도 학습이 잘 됨 (기울기 소실 해결).

7.1.4 4. 출력층 설계 (Output Layer)

□ 개념 3: 문제 유형에 따른 마무으리

한 줄 요약: 마지막 층의 활성화 함수는 "우리가 풀고 싶은 문제"가 무엇이냐에 따라 결정됩니다.

문제 유형	출력층 함수	예시
회귀 (값 예측)	Linear (None)	집값 예측 ($-\infty \sim \infty$)
이진 분류 (O/X)	Sigmoid	스팸 메일 분류 ($0 \sim 1$)
다중 분류 (A/B/C)	**Softmax**	숫자 인식 (0 9 중 하나)

Softmax 함수

입력값들을 **"확률의 총합이 1이 되도록"** 변환해 주는 함수입니다. 가장 큰 값을 더 돋보이게(Soft Max) 만들어줍니다.

7.2 실전 시나리오: 넥슨 캐릭터 직업 추천 AI

□ Scenario: 뉴비에게 딱 맞는 직업은?

신규 유저의 플레이 스타일 데이터(x)를 분석해 전사, 마법사, 도적 중 하나를 추천해 주는 AI를 만듭니다.

1. **입력층:** 공격성향, 이동빈도, 파티선호도 등 10개 변수.
2. **은닉층 (Hidden Layers):**
   - 3개의 층을 쌓음.
   - 활성화 함수: **ReLU** 사용. (복잡한 패턴 학습 + 빠른 속도)
   - 역할: "공격적이고 이동이 많음" 같은 추상적 특징을 스스로 조합해냄.
3. **출력층 (Output Layer):**
   - 클래스 개수: 3개 (전사, 마법사, 도적).
   - 활성화 함수: **Softmax** 사용.
4. **결과 예시:**

$$[2.5, 0.1, 4.8] \xrightarrow{\text{Softmax}} [0.09, 0.01, 0.90]$$

5. **해석:** "이 유저는 90% 확률로 '도적'이 적합합니다."

7.3 자주 묻는 질문 (FAQ)

Q1. 은닉층 개수는 어떻게 정하나요? A. 정해진 공식은 없습니다. 이를 **하이퍼파라미터(Hyperparameter)**라고 합니다. 보통 1 2층으로 시작해서 성능을 보며 늘려나갑니다. 너무 깊으면 학습이 어렵고(Overfitting), 너무 얕으면 복잡한 문제를 못 풉니다.

Q2. Dying ReLU가 뭔가요? A. ReLU의 단점입니다. 입력이 계속 음수가 들어와서 출력이 0이 되면, 해당 뉴런이 영원히 깨어나지 못하고 '죽어버리는' 현상입니다. 이를 막기 위해 Leaky ReLU(음수일 때 약간의 기울기를 줌)를 쓰기도 합니다.

> **Next Step:** 신경망의 구조(집)를 다 지었습니다. 이제 이 집에 지능을 불어넣으려면 **학습(Training)**을 시켜야겠죠? 정답과 예측의 오차를 계산해서, 거꾸로 전파하며 가중치를 수정하는 마법 같은 알고리즘, **역전파(Backpropagation)**를 다음 시간에 배웁니다.

Unit 3 (Part A) 핵심 요약

- **FNN:** 입력 → 은닉 → 출력으로 흐르는 가장 기초적인 신경망.
- **비선형성:** 활성화 함수가 있어야 딥러닝이 선형 회귀보다 똑똑해진다.
- **ReLU:** 은닉층의 지배자. 학습 속도가 빠르고 깊은 망에 유리하다.
- **Softmax:** 다중 분류 문제에서 출력값을 확률로 변환한다.

Chapter 8

MIT 6.86x: 오차를 통해 배우다

Course Structure & Current Focus

- Unit 3: Neural Networks Basics (구조 설계)
- **Unit 4: Backpropagation (현재 단원: 학습 메커니즘)**
  - 8.1 The Engine: Backpropagation Algorithm
  - 8.2 The Scorecard: Loss Functions (MSE vs Cross-Entropy)
  - 8.3 Setting the Stage: Initialization & Learning Rate
  - 8.4 Stabilization: Batch Norm & Dropout
- Unit 5: Recurrent Neural Networks (순차 데이터)

8.1 Unit 4. 역전파와 학습의 원리 (Backpropagation)

*Unit 3*에서 우리는 수천 개의 뉴런이 연결된 뇌(*FNN*)를 만들었습니다. 하지만 갓 태어난 뇌는 아는 것이 없습니다(가중치 W가 랜덤임). 이제 이 뇌에게 정답을 보여주고, 틀릴 때마다 **"누가 잘못했는지"**를 따져서 조금씩 똑똑해지게 만들어야 합니다. 이 과정이 바로 학습(*Training*)입니다.

□ 개요 (Overview)

이 단원에서는 신경망 학습의 핵심 엔진인 **역전파(Backpropagation)** 알고리즘을 배웁니다. 미분의 **연쇄 법칙(Chain Rule)**을 이용해 오차의 원인을 찾아내고, **손실 함수(Loss Function)**를 통해 성적을 매기며, **초기화(Initialization)**와 **정규화(Normalization)** 기법으로 학습을 안정화시키는 전체 파이프라인을 다룹니다.

□ 핵심 용어 사전

용어 (Term)	직관적 의미 (Meaning)
Forward Pass	입력 → 출력. 예측값을 뽑아내는 과정.
Backward Pass	출력 → 입력. 틀린 이유(미분값)를 배달하는 과정.
Chain Rule	꼬리에 꼬리를 무는 미분. (나비효과 추적)
Epoch	전체 문제집을 한 번 다 풀어보는 것.
Dropout	일부 뉴런을 랜덤하게 꺼서, 특정 뉴런 편애를 막는 기술.

8.1.1 1. 역전파 알고리즘 (Backpropagation)

□ 개념 1: 책임 소재 따지기 (The Blame Game)

한 줄 요약: 결과가 틀렸을 때, 출력층에서부터 거꾸로 거슬러 올라가며 각 뉴런이 오차에 기여한 만큼(기울기) 가중치를 수정합니다.

1) 작동 원리: 연쇄 법칙 (Chain Rule)

우리는 최종 오차 L을 줄이기 위해 가중치 w를 조절하고 싶습니다. 즉, $\frac{\partial L}{\partial w}$가 필요합니다. 하지만 L과 w는 멀리 떨어져 있습니다.

$$\frac{\partial L}{\partial w} = \underbrace{\frac{\partial L}{\partial y}}_{\text{오차 변화}} \cdot \underbrace{\frac{\partial y}{\partial h}}_{\text{출력 변화}} \cdot \underbrace{\frac{\partial h}{\partial w}}_{\text{은닉층 변화}}$$

- 마치 통역을 거치듯, 미분값을 층층이 곱해서 전달합니다.

- **** (위 이미지 참고): δ 값(오차 신호)이 오른쪽(a^4)에서 왼쪽(a^0)으로 전파되는 것을 볼 수 있습니다.

2) 숫자 예시

간단한 식 $L = (y - wx)^2$ 에서, $y = 10, x = 2, w = 3$ 이라고 합시다.

1. **순전파:** 예측 $\hat{y} = 3 \times 2 = 6$.
2. **오차:** $L = (10 - 6)^2 = 16$.
3. **역전파:** w를 조절하면 L이 얼마나 변할까?

$$\frac{\partial L}{\partial w} = 2(y - \hat{y}) \cdot (-x) = 2(4) \cdot (-2) = -16$$

4. **의미:** 기울기가 음수(-16)이므로, w를 키워야 오차가 줄어듭니다.

8.1.2 2. 손실 함수 (Loss Function)

□ 개념 2: 채점 기준표

한 줄 요약: 문제의 유형(회귀 vs 분류)에 따라 "얼마나 틀렸는지"를 계산하는 방식이 다릅니다.

구분	MSE (Mean Squared Error)	Cross-Entropy
용도	**회귀** (집값, 온도 예측)	**분류** (스팸, 아이템 등급)
수식	$\frac{1}{n}\sum(y - \hat{y})^2$	$-\sum y \log(\hat{y})$
특징	거리가 멀수록 제곱으로 혼냄.	확률이 틀릴수록 급격히 혼냄.

왜 분류 문제에 MSE를 안 쓰나요?

Sigmoid + MSE 조합을 쓰면, 오차가 엄청 큰데도 미분값(Gradient)이 0에 가까워져서 학습이 안 되는 현상이 발생합니다. 반면 **Cross-Entropy**는 로그 함수 특성상 틀리면 미분값이 폭발적으로 커져서 "정신 차려!"라고 강하게 피드백을 줍니다.

8.1.3 3. 초기화와 학습률 (Initialization & Learning Rate)

□ 개념 3: 시작이 반이다

한 줄 요약: 출발점(초기화)을 잘 잡고, 보폭(학습률)을 적절히 조절해야 산 아래로 잘 내려갈 수 있습니다.

1) 가중치 초기화 (Initialization)

- **0으로 초기화하면? (The Trap of Symmetry):** 모든 뉴런이 똑같은 값을 내뱉고, 똑같이 업데이트됩니다. 100개를 써도 1개 쓰는 것과 같습니다.
- **해결책:**
  - **He Initialization:** ReLU 함수를 쓸 때 표준. (분산을 $2/n$로 맞춤)
  - **Xavier Initialization:** Sigmoid/Tanh를 쓸 때 표준.

2) 학습률 스케줄링 (Learning Rate Decay)

처음엔 성큼성큼(큰 η) 가다가, 정답 근처에서는 조심조심(작은 η) 가야 합니다.

$$\eta_t = \eta_0/(1+kt) \quad \text{(시간이 갈수록 줄어듦)}$$

8.1.4 4. 학습 안정화 기법: 배치 정규화 & 드롭아웃

□ 개념 4: 흔들리지 않는 편안함

한 줄 요약: 딥러닝은 깊어질수록 학습이 불안정해집니다. 이를 막기 위해 데이터를 강제로 정렬하거나(Batch Norm), 일부러 핸디캡을 줍니다(Dropout).

A. 배치 정규화 (Batch Normalization)

- **문제:** 앞단 레이어의 가중치가 바뀌면, 뒷단 레이어 입장에서는 들어오는 데이터 분포가 계속 바뀝니다 (Internal Covariate Shift).
- **해결:** 각 층마다 들어오는 데이터를 **평균 0, 분산 1**로 강제 조정해 버립니다.
- **효과:** 학습 속도가 비약적으로 빨라지고, 초기화에 덜 민감해집니다.

B. 드롭아웃 (Dropout)

- **개념:** 학습할 때마다 은닉층 뉴런의 50%를 랜덤하게 꺼버립니다.
- **비유:** 축구팀에서 에이스(특정 뉴런) 한 명만 믿는 것을 방지하기 위해, 연습 때 에이스를 벤치에 앉혀두고 나머지 선수들끼리 훈련시킵니다. 팀 전체 전력이 올라갑니다(과적합 방지).

8.2 실전 시나리오: 넥슨 게임 이탈(Churn) 방지 모델

□ Scenario: VIP 유저가 떠나는 이유 찾기

당신은 넥슨의 PM입니다. "이번 달에 접속할 것"이라고 예측(0.9)했던 VIP 유저가 실제로는 접속하지 않았습니다(Target 0.0). 오차가 큽니다.

1. **Forward:** 유저 데이터(길드 활동, 결제액 등) → FNN → 예측확률 90%.
2. **Loss:** Cross-Entropy로 계산하니 손실이 매우 큽니다. ("확신했는데 틀렸군!")
3. **Backward:**
   - 역전파가 시작됩니다. 출력층에서 은닉층으로 거슬러 올라갑니다.
   - 범인 색출: "길드 활동 점수"에 연결된 가중치가 범인이었습니다. "길드 활동이 많으면 무조건 남는다"고 과대평가하고 있었네요.
4. **Update:** 해당 가중치를 대폭 삭감합니다.
5. **Result:** 모델은 이제 "길드 활동이 많아도 최근 접속이 뜸하면 이탈할 수 있다"는 미묘한 패턴을 배우게 됩니다.

8.3 자주 묻는 질문 (FAQ)

Q1. 역전파를 손으로 계산할 줄 알아야 하나요? A. 개념만 알면 됩니다. PyTorch나 TensorFlow 같은 프레임워크가 'loss.backward()' 한 줄이면 자동으로 미분을 다 해줍니다. (이걸 **Autograd**라고 합니다.)

Q2. Dropout은 테스트(실전) 때도 쓰나요? A. 절대 아닙니다! 학습(Training) 때는 뉴런을 끄지만, 테스트(Inference) 때는 모든 뉴런을 켜서 전력을 다해야 합니다. 대신 출력값에 비율(0.5)을 곱해서 보정해 줍니다.

Next Step: 지금까지 배운 신경망은 정지된 이미지나 테이블 데이터에는 잘 작동하지만, "문장"이나 "주가"처럼 **순서가 있는(Sequential) 데이터**에는 약합니다. 다음 **Unit 5**에서는 기억력(Memory)을 가진 신경망, **RNN(Recurrent Neural Networks)**과 LSTM을 배웁니다.

Unit 4 핵심 요약

- **역전파 (Backprop):** Chain Rule을 이용해 오차의 책임(Gradient)을 각 가중치에 배분한다.
- **손실 함수:** 회귀는 MSE, 분류는 Cross-Entropy가 국룰이다.
- **He Initialization:** ReLU를 쓸 때는 초기화를 잘해야 학습이 시작된다.
- **Batch Norm & Dropout:** 학습을 빠르고 안정적으로 만들고 과적합을 막는 필수 테크닉.

Chapter 9

MIT 6.86x: 컴퓨터가 세상을 보는 법

Course Structure & Current Focus

- Unit 3 (Part A): Neural Network Basics (FNN)
- **Unit 3 (Part B): Convolutional Neural Networks (현재 단원: 이미지 처리의 혁명)**
  - 9.1 Convolutional Layer (Filter, Stride, Padding)
  - 9.2 Pooling Layer (Max Pooling)
  - 9.3 CNN Architecture (Feature Extractor + Classifier)
- Unit 4: Backpropagation (학습 원리)

9.1 Unit 3 (Part B). 합성곱 신경망 (CNN)

*Part A*에서 배운 일반 신경망(*FNN*)은 이미지를 처리할 때 치명적인 약점이 있었습니다. 28×28 이미지를 784개의 *1*줄짜리 벡터로 펴버리는 순간(*Flatten*), 픽셀 간의 위아래/좌우 관계가 모두 파괴되기 때문입니다. *CNN*은 이 문제를 해결하기 위해 **"이미지를 펼치지 않고, 있는 그대로 훑어보는"** 방식을 사용합니다.

□ 개요 (Overview)

CNN은 사람의 시각 피질 구조를 모방하여 만들어졌습니다. **합성곱 층(Convolutional Layer)**을 통해 이미지의 국소적인 특징(선, 면, 패턴)을 감지하고, **풀링 층(Pooling Layer)**을 통해 중요한 정보만 남기고 압축하여, 위치가 조금 변해도 사물을 잘 알아보는 강건한 모델을 만듭니다.

□ 핵심 용어 사전

용어 (Term)	직관적 의미 (Meaning)
Filter (Kernel)	이미지를 훑어보는 작은 돋보기 도장 (3×3 행렬).
Feature Map	필터가 훑고 지나간 뒤 남은 '특징들의 지도'.
Stride	필터가 이동하는 보폭. 크면 듬성듬성, 작으면 꼼꼼히.
Padding	이미지 테두리에 0을 채워 크기가 줄어드는 것을 방지.
Pooling	이미지를 축소/요약하는 과정 (해상도 줄이기).

9.1.1 1. 합성곱 계층 (Convolutional Layer)

□ 개념 1: 도장 찍기 (Sliding Window)

한 줄 요약: 필터(도장)를 이미지 왼쪽 위부터 오른쪽 아래까지 꾹꾹 눌러가며(Sliding) 특정 패턴이 있는지 검사합니다.

1) 필터 (Filter / Kernel)

- **정의:** 작은 가중치 행렬(보통 3×3). 특징 탐지기(Feature Detector) 역할을 합니다.
- **작동:** 이미지의 특정 영역과 필터를 내적(곱하고 더함)합니다. 값이 클수록 해당 필터가 찾는 모양(예: 가로선)과 일치한다는 뜻입니다.
- **학습:** 초반 층 필터는 '선'을 찾고, 깊은 층 필터는 '눈, 코' 같은 복잡한 모양을 찾도록 **스스로 학습**됩니다.

2) 스트라이드 (Stride)

필터가 이동하는 간격입니다.

- **Stride 1:** 한 칸씩 꼼꼼하게 이동. (출력 크기 유지)
- **Stride 2:** 두 칸씩 점프하며 이동. (출력 크기가 절반으로 줄어듦 → Downsampling)

3) 패딩 (Padding)

필터를 거치면 귀퉁이가 깎여나가 이미지가 점점 작아집니다.

- **Zero Padding:** 테두리에 0을 한 바퀴 둘러줍니다.
- **목적:** 출력 크기를 입력과 똑같이 유지(Same Padding)하여 층을 깊게 쌓을 수 있게 합니다.

9.1.2 2. 풀링 계층 (Pooling Layer)

□ 개념 2: 요약 노트 만들기

한 줄 요약: 너무 디테일한 정보는 버리고, "여기에 뭔가 있었다"는 핵심 정보만 남겨서 이미지 사이즈를 줄입니다.

1) 최대 풀링 (Max Pooling)

가장 널리 쓰이는 방식입니다. 2×2 영역에서 **가장 큰 값 하나**만 남기고 나머지는 버립니다.

- **의미:** "이 구역에서 가장 강한 특징 하나만 기억해!"
- **효과:** 노이즈가 제거되고, 데이터 양이 1/4로 줄어 연산 속도가 빨라집니다.

2) 이동 불변성 (Translation Invariance)

풀링을 하면 이미지가 약간 흔들리거나(Shift), 사물의 위치가 조금 바뀌어도 결과값이 거의 변하지 않습니다. 덕분에 CNN은 고양이가 사진 구석에 있든 중앙에 있든 "고양이"라고 잘 인식합니다.

9.1.3 3. CNN의 전체 구조 (Architecture)

□ 개념 3: 샌드위치 구조

한 줄 요약: [특징 추출]과 [분류]의 두 단계로 나뉩니다.

1) 특징 추출기 (Feature Extractor)

$$\text{Input} \rightarrow [\text{Conv} + \text{ReLU} + \text{Pool}] \times N$$

- 블록을 반복할수록 이미지는 작아지고(가로/세로 축소), 특징은 풍부해집니다(채널 깊이 증가).
- 예: $224 \times 224 \times 3$ (이미지) $\rightarrow \cdots \rightarrow 7 \times 7 \times 512$ (압축된 특징 맵).

2) 분류기 (Classifier)

$$\text{Flatten} \rightarrow \text{Fully Connected} \rightarrow \text{Softmax}$$

- 추출된 입체적인 특징 맵을 한 줄로 쫙 펴서(Flatten), 우리가 Unit 3(Part A)에서 배운 일반 신경망에 넣고 최종 확률을 계산합니다.

9.2 실전 시나리오: 넥슨 게임 에셋 자동 태깅

□ Scenario: 수만 개의 아이콘 정리하기

당신은 넥슨의 테크 PM입니다. 20년 치 게임 아이콘(검, 방패, 포션 등) 5만 개가 폴더에 뒤섞여 있습니다. 이를 자동으로 분류하는 AI를 만듭니다.

1. **입력:** 64×64 픽셀 컬러 아이콘 (채널 3).
2. **Conv 1:** 3×3 필터로 외곽선(Edge)을 따냅니다. (검의 날카로운 선 감지)
3. **Pool 1:** 해상도를 줄여서 노이즈를 없앱니다.
4. **Conv 2:** 이제 선들이 모여 '손잡이', '칼날' 같은 부분 모양을 인식합니다.
5. **Conv 3:** 부분들이 모여 '십자가 모양(검)', '동그라미(방패)' 같은 전체 형태를 인식합니다.
6. **Classifier:** "이 특징은 98% 확률로 [전설 등급 한손검]입니다"라고 분류합니다.
7. **결과:** 디자이너들이 "한손검"만 검색해도 수천 개의 검 이미지가 즉시 조회되는 시스템 구축 완료.

9.3 자주 묻는 질문 (FAQ)

Q1. 필터의 값(숫자)은 사람이 정해주나요? A. 아닙니다! 과거에는 사람이 정했지만(SIFT, HOG 등), 딥러닝에서는 **역전파(Backpropagation)**를 통해 AI가 데이터에 맞는 최적의 필터 값을 스스로 학습합니다.

Q2. 채널(Channel)이 정확히 뭔가요? A. 정보의 두께입니다.

- 입력 이미지: R, G, B 3장의 필름이 겹쳐져 있으므로 채널이 3입니다.
- Conv 층 통과 후: 필터를 64개 썼다면, 64가지의 서로 다른 특징(가로선, 세로선, 대각선...)을 찾아낸 64장의 지도가 생기므로 채널이 64가 됩니다.

Next Step: CNN 덕분에 우리는 "공간(Spatial)" 정보를 다루는 법을 마스터했습니다. 그런데 텍스트나 음성처럼 **"시간(Temporal)" 순서**가 중요한 데이터는 어떻게 할까요? "나는 밥을 먹었다"와 "먹었다 밥을 나는"은 다르잖아요? 다음 시간에는 순차 데이터를 위한 신경망, **Unit 5: RNN (Recurrent Neural Networks)**을 배웁니다.

Unit 3 (Part B) 핵심 요약

- **CNN:** 이미지를 1줄로 펴지 않고, 도장(Filter)을 찍으며 특징을 찾는다.
- **Conv Layer:** 특징 탐지기. Stride로 간격 조절, Padding으로 크기 유지.
- **Pool Layer:** 정보 압축기. Max Pooling으로 중요 정보만 남긴다.
- **구조:** Conv(특징 추출) 반복 후 FC(분류)로 마무리하는 샌드위치 구조.

Chapter 10

MIT 6.86x: 기억을 가진 신경망

Course Structure & Current Focus

- Unit 3: CNN (공간 정보 처리)
- Unit 4: Backpropagation (학습 원리)
- **Unit 5: Recurrent Neural Networks (현재 단원: 시간 정보 처리)**
  - 10.1 The Loop: RNN Architecture
  - 10.2 The Problem: Vanishing Gradient
  - 10.3 The Solution: LSTM & GRU
  - 10.4 The Evolution: Attention Mechanism
- Unit 6: Unsupervised Learning (군집화)

10.1 Unit 5. 순환 신경망 (RNN)

> *CNN* 덕분에 컴퓨터는 '보는 눈'을 가졌습니다. 하지만 "나는 밥을 먹었다"라는 문장을 *CNN*에 넣으면 "나", "밥", "먹"의 위치 관계는 알 수 있을지 몰라도, 단어가 등장하는 **순서(*Sequence*)**가 만드는 인과관계는 놓치기 쉽습니다. 언어, 주가, 영상처럼 흐름이 있는 데이터를 처리하기 위해 우리는 신경망에 **'기억력(*Memory*)'**을 심어주어야 합니다.

□ 개요 (Overview)

RNN은 이전 단계의 출력(Hidden State)을 현재 단계의 입력으로 다시 사용하는 **재귀적(Recurrent) 구조**를 가집니다. 이 단원에서는 기본 RNN의 구조와 한계(장기 의존성 문제), 이를 해결한 **LSTM**, 그리고 현대 자연어 처리(NLP)의 핵심인 **Attention** 메커니즘까지의 진화 과정을 다룹니다.

□ 핵심 용어 사전

용어 (Term)	직관적 의미 (Meaning)
Sequence Data	순서가 바뀌면 의미가 깨지는 데이터 (텍스트, 시계열).
Hidden State (h_t)	과거의 정보를 압축해서 들고 있는 '기억 상자'.
Vanishing Gradient	문장이 길어지면 앞부분의 기억이 희미해져 학습이 안 되는 현상.
LSTM / GRU	기억을 오래 유지하기 위해 '문지기(Gate)'를 단 똑똑한 RNN.
Attention	모든 걸 다 기억하려 하지 말고, 필요할 때마다 원본을 다시 찾아보는 기술.

10.1.1 1. RNN의 핵심 아이디어: "기억(Memory)"

□ 개념 1: 돌림 노래 (Loop)

한 줄 요약: 현재의 나(h_t)는 어제의 기억(h_{t-1})과 오늘의 경험(x_t)이 합쳐져서 만들어집니다.

1) 구조적 특징

일반 신경망(FNN)은 입력 x가 출력 y로 직행하지만, RNN은 은닉층의 출력이 다시 은닉층의 입력으로 들어옵니다. 이를 시간 순서대로 펼치면(Unfold) 마치 여러 개의 신경망이 옆으로 연결된 것처럼 보입니다.

2) 수식 (Update Rule)

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b)$$

- h_t: 현재 시점(t)의 상태(기억).
- h_{t-1}: 이전 시점($t-1$)의 상태(과거의 기억).
- x_t: 현재 시점의 새로운 입력.
- **핵심:** 가중치 W_{hh}, W_{xh}는 모든 시점에서 **공유(Shared)**됩니다. 즉, 어제의 나를 만든 규칙과 오늘의 나를 만드는 규칙은 같습니다.

10.1.2 2. LSTM (Long Short-Term Memory)

□ 개념 2: 정보의 고속도로와 문지기

한 줄 요약: 중요한 정보는 고속도로(Cell State)에 태워 끝까지 보내고, 불필요한 정보는 휴게소(Forget Gate)에서 버립니다.

1) 한계: 장기 의존성 (Long-Term Dependency)

기본 RNN은 문장이 길어지면(t가 커지면) 역전파 시 기울기가 계속 곱해지면서 0에 수렴해버립니다. (예: "나는... (100단어) ... 한국 사람이다"에서 '나는'을 까먹음).

2) 해결책: 3개의 게이트 (Gate)

LSTM은 **셀 상태(Cell State, C_t)**라는 정보 전용 통로를 뚫고, 3개의 문지기가 정보를 관리합니다.

- **Forget Gate (f_t):** "과거 기억 중 쓸모없는 건 지워라." (Sigmoid $\rightarrow$ 0이면 삭제)
- **Input Gate (i_t):** "현재 정보 중 중요한 것만 저장해라."
- **Output Gate (o_t):** "다음 단계로 무엇을 넘겨줄지 결정해라."

GRU: LSTM이 너무 복잡해서, 게이트를 2개(Update, Reset)로 줄여 속도를 높인 가성비 모델입니다. 성능은 비슷합니다.

10.1.3 3. 어텐션(Attention) 메커니즘

□ 개념 3: 컨닝 페이퍼 (Cheat Sheet)

한 줄 요약: 시험 볼 때(출력할 때) 모든 걸 외워서 풀려 하지 말고, 필요할 때마다 교과서(입력 문장)의 해당 부분을 훔쳐보세요.

1) 병목 현상 (Bottleneck)

LSTM조차도 아무리 긴 문장이라도 결국 마지막에 하나의 벡터(Context Vector)로 압축해야 합니다. 이 과정에서 정보 손실이 발생합니다.

2) 핵심 아이디어

출력 단어를 하나 만들 때마다, 입력 문장 전체를 다시 훑어봅니다. 단, **가중치(Attention Score)**를 다르게 둡니다.

- 예: "I ate an apple" → "나는 사과를 먹었다"
- "사과를" 예측할 때: 입력의 "apple"에 집중도(Score) 90%, "I"에 5%를 줍니다.
- **수식:** Context $= \sum \alpha_i h_i$ (α_i: 집중도 가중치)

10.2 실전 시나리오: 글로벌 게임 실시간 번역

□ Scenario: '메이플스토리 월드' 글로벌 채팅

한국 유저와 미국 유저가 실시간으로 소통합니다. 채팅 번역 시스템을 구축합니다.

1. **입력 문장:** "배가 너무 고파서 배를 먹었어."
2. **문제 (동음이의어):** 앞의 '배(Stomach)'와 뒤의 '배(Pear)'를 구분해야 합니다.
3. **RNN/LSTM의 한계:** 문맥을 파악하긴 하지만, 문장이 복잡하면 두 '배'를 혼동하여 "Stomach... ate stomach"으로 오역할 수 있습니다.
4. **Attention 적용:**
   - 첫 번째 '배' 번역 시: "고파서(hungry)" 단어에 Attention이 꽂힙니다. → **Stomach**로 번역.
   - 두 번째 '배' 번역 시: "먹었어(ate)" 단어에 Attention이 꽂힙니다. → **Pear**로 번역.
5. **결과:** "I was so hungry that I ate a pear." (완벽한 번역)

10.3 자주 묻는 질문 (FAQ)

Q1. 요즘도 LSTM을 쓰나요? Transformer(GPT)가 다 대체하지 않았나요? A. 대규모 언어 모델(LLM)은 Transformer가 지배했습니다. 하지만 시계열 데이터(주가, 서버 로그 예측)나, 리소스가 제한적인 모바일 기기에서의 간단한 음성 인식 등에는 여전히 가볍고 효율적인 LSTM/GRU가 현역으로 쓰입니다.

Q2. 이미지는 CNN, 텍스트는 RNN으로 딱 정해져 있나요? A. 예전엔 그랬지만 경계가 무너졌습니다.

- **Vision Transformer (ViT):** 이미지를 조각내서 순서대로 처리하는 Transformer가 CNN을 위협하고 있습니다.

- **1D-CNN:** 텍스트를 1차원 이미지로 보고 CNN을 돌려서 빠르게 분류하기도 합니다.

Next Step: 지도 학습(정답 맞히기)의 여정이 끝났습니다. 이제 정답이 없는 데이터에서 스스로 패턴을 찾아내는 **비지도 학습(Unsupervised Learning)**의 세계로 넘어갑니다. 다음 시간에는 비슷한 유저끼리 묶는 **군집화(Clustering)**와 복잡한 데이터를 압축하는 차원 축소를 배웁니다.

Unit 5 핵심 요약

- **RNN:** 출력이 입력으로 돌아오는 루프 구조. 순차 데이터 처리에 특화됨.
- **LSTM:** 셀 상태(Cell State)와 게이트(Gate)를 도입하여 장기 기억력을 확보함.
- **Attention:** 입력 데이터의 중요한 부분에 가중치를 두어 병목 현상을 해결함. (Transformer의 시초)
- **활용:** 번역, 챗봇, 주가 예측, 로그 분석 등 시간 흐름이 있는 모든 곳.

Chapter 11

MIT 6.86x: 정답 없는 세계의 질서

Course Structure & Current Focus

- Unit 1 3: Supervised Learning (정답 y가 있는 학습)
- **Unit 4: Unsupervised Learning (현재 단원: 정답 y가 없는 학습)**
  - **11.1 K-Means Algorithm (좌표 하강법 관점)**
  - 11.2 K-Medoids (이상치에 강건한 방법)
  - 11.3 Hierarchy Clustering (계층적 군집화)
- Unit 5: Generative Models (데이터 생성)

11.1 Unit 4. 군집화 (Clustering)

> 지금까지 우리는 선생님이 채점해주는(*Label*이 있는) 문제를 풀었습니다. 하지만 현실 세계 데이터의 *90%*는 정답지가 없습니다. 넥슨의 로그 데이터도 "이 유저는 하드코어 유저다"라고 써붙여져 있지 않죠. 이제 우리는 아무도 가르쳐주지 않은 상태에서, **데이터끼리 뭉쳐있는 구조(*Structure*)**를 스스로 찾아내는 방법을 배웁니다.

□ 개요 (Overview)

군집화는 유사한 데이터끼리 그룹을 묶는 기술입니다. 가장 대표적인 **K-Means** 알고리즘을 단순한 반복 절차가 아닌 **좌표 하강법(Coordinate Descent)**이라는 최적화 이론으로 해석하고, 평균(Mean)의 한계를 극복하기 위한 **K-Medoids** 알고리즘을 비교 학습합니다.

□ 핵심 용어 사전

용어 (Term)	**직관적 의미 (Meaning)**
Centroid (μ)	클러스터의 중심점. (무게 중심)
Assignment (r)	"너는 A팀, 쟤는 B팀"이라고 배정하는 변수.
Inertia (Cost J)	각 팀원들이 중심점으로부터 떨어진 거리의 제곱 합. (응집도)
Coordinate Descent	x, y를 동시에 못 찾으니, x 고정하고 y 찾고, y 고정하고 x 찾는 전략.
Medoid	클러스터를 대표하는 **실제 데이터 포인트**. (반장)

11.1.1 1. K-Means 알고리즘과 좌표 하강법

□ 개념 1: 깃발 꽂기 게임

한 줄 요약: 깃발(중심)을 꽂고 사람들이 가장 가까운 깃발로 모입니다. 사람이 모인 곳의 한가운데로 깃발을 다시 옮깁니다. 이를 반복합니다.

1) 작동 원리 (The Iterative Process)

1. **초기화:** K개의 중심점(μ)을 랜덤하게 찍습니다.
2. **할당 (Assignment):** 모든 데이터는 가장 가까운 중심점 소속이 됩니다.
3. **업데이트 (Update):** 각 그룹의 무게중심(평균)으로 중심점을 이동시킵니다.
4. **반복:** 중심이 안 움직일 때까지 2 3번을 반복합니다.

2) 수학적 해석: 좌표 하강법 (Coordinate Descent)

이 직관적인 과정은 사실 엄밀한 수학적 최적화입니다.

$$J(r, \mu) = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2$$

이 함수 J를 최소화해야 하는데, 변수 r(소속)과 μ(위치)가 서로 얽혀 있어서 미분이 불가능합니다. 그래서 **하나를 고정(Freeze)하고 나머지를 최적화**하는 전략을 씁니다.

- **Step 1 (r 최적화):** μ를 고정.

$$\min_r J \implies \text{각 데이터를 가장 가까운 } \mu_k \text{에 할당.}$$

- **Step 2 (μ 최적화):** r을 고정.

$$\min_\mu J \implies \mu_k \text{를 해당 그룹 데이터들의 평균(Mean)으로 이동.}$$

이 과정은 J 값을 계단식으로 계속 낮추며, 더 이상 낮아지지 않는 **지역 최적해(Local Minimum)**에 반드시 수렴합니다.

11.1.2 2. K-Medoids 알고리즘

□ 개념 2: 평균의 함정과 실제 대표 선출

한 줄 요약: K-Means는 허공에 있는 평균을 중심으로 잡지만, K-Medoids는 실제 데이터 중 하나를 대장(Representative)으로 뽑습니다.

1) K-Means의 치명적 약점 (Outliers)

데이터가 $(1,1), (2,2), (3,3)$ 그리고 저 멀리 $(100, 100)$에 하나 있다고 합시다.

- **K-Means (μ):** 평균은 약 $(26, 26)$이 됩니다. 중심점이 데이터가 모인 곳을 떠나 엉뚱한 곳에 위치하게 됩니다. (이상치에 민감)
- **K-Medoids:** 중간값인 $(2,2)$ 혹은 $(3,3)$을 대표로 삼습니다. $(100, 100)$은 무시됩니다. (강건함)

2) 비교 분석

특징	K-Means	K-Medoids (PAM)
중심점	평균 (가상의 좌표)	Medoid (실제 데이터)
계산 속도	매우 빠름 ($O(N)$)	느림 ($O(N^2)$)
이상치	매우 민감함 (취약)	강건함 (Robust)
적용	일반적인 대용량 데이터	노이즈가 많은 데이터

11.2 실전 시나리오: 넥슨 유저 세분화 (User Segmentation)

□ Scenario: 고래(Whale) 유저와 일반 유저 분류

당신은 게임 내 유저들을 '과금러', '즐겜러', '페인' 등으로 자동 분류하고 싶습니다.

1. **데이터:** [플레이 시간, 결제 금액]
2. **K-Means 적용 시도:**
   - 한 명의 '슈퍼 과금러(10억 원 결제)'가 데이터에 포함되어 있습니다.
   - K-Means의 중심점(평균)이 이 사람 쪽으로 쭈욱 끌려갑니다.
   - 결과: 일반 과금 유저(10만 원)들이 '무과금 그룹'으로 잘못 분류되고, '고래 그룹'은 범위가 너무 넓어집니다.
3. **K-Medoids 전환:**
   - 중심점을 실제 유저 중에서 뽑습니다.
   - 슈퍼 과금러는 그냥 하나의 점일 뿐, 중심(대표 유저)을 바꾸지 못합니다.
   - 결과: 대다수의 일반 유저 패턴을 반영한 합리적인 그룹핑이 완성됩니다.

11.3 자주 묻는 질문 (FAQ)

Q1. K(그룹 개수)는 어떻게 정하나요? A. 가장 어려운 문제입니다. 보통 **Elbow Method**를 씁니다. K를 1부터 늘려가며 J(거리 제곱 합)를 그려봅니다. K가 늘면 J는 무조건 줄어드는데, 줄어드는 폭이 급격히 둔화되어 팔꿈치처럼 꺾이는 지점을 선택합니다.

Q2. K-Means는 항상 정답을 찾나요? A. 아닙니다. **초기값**을 어디에 찍느냐에 따라 결과가 달라집니다(Local Optima 문제). 그래서 보통 'K-Means++' 같은 초기화 기법을 쓰거나, 랜덤 초기화로 여러 번 돌려서 가장 좋은 결과를 선택합니다.

Next Step: K-Means는 데이터를 "너는 A팀, 너는 B팀"식으로 딱 잘라 말합니다(Hard Clustering). 하지만 "이 유저는 A팀 성향 70%, B팀 성향 30%야"라고 확률적으로 말할 순 없을까요? 다음 시간에는 통계적 군집화 방법인 **GMM (Gaussian Mixture Models)**과 이를 푸는 **EM 알고리즘**을 배웁니다.

Unit 4 핵심 요약

- **Clustering:** 정답 없이 데이터의 구조를 찾는 비지도 학습.
- **K-Means:** 중심 할당 $\leftrightarrow$ 중심 이동을 반복하는 좌표 하강법 알고리즘.
- **Coordinate Descent:** 변수들을 한 번에 최적화하기 어려울 때, 하나씩 번갈아 가며 최적화하는 기법.
- **K-Medoids:** 평균 대신 실제 데이터를 대표로 사용하여 이상치에 강하다.

Chapter 12

MIT 6.86x: 데이터 압축과 추천의 원리

Course Structure & Current Focus

- Unit 4 (Part A): Clustering (비슷한 것끼리 묶기)
- **Unit 4 (Part B): Dimension Reduction & Matrix Factorization**
  - 12.1 PCA: Reconstruction Error View (데이터 압축)
  - 12.2 Matrix Completion Problem (빈칸 채우기)
  - 12.3 Matrix Factorization (추천 시스템의 엔진)
  - 12.4 Collaborative Filtering (협업 필터링)
- Unit 5: Generative Models

12.1 Unit 4 (Part B). 차원 축소와 행렬 분해

> 지난 파트(*Clustering*)에서는 데이터를 그룹핑하여 구조를 파악했습니다. 이번에는 데이터 자체를 **'압축'**하거나, 비어 있는 데이터를 **'복원'**하는 방법을 배웁니다. 이는 마치 저화질 이미지를 고화질로 복원하거나, 내가 보지 않은 영화의 평점을 예측하는 마법 같은 기술의 기초가 됩니다.

□ 개요 (Overview)

이 단원에서는 **PCA(주성분 분석)**를 '분산 최대화'가 아닌 **'재구성 오차 최소화'**라는 머신러닝 관점에서 재해석합니다. 또한, 넷플릭스 추천 시스템의 근간이 되는 **행렬 분해(Matrix Factorization)**를 통해 희소 행렬(Sparse Matrix)의 빈칸을 채우고 사용자 취향을 예측하는 원리를 학습합니다.

□ 핵심 용어 사전

용어 (Term)	직관적 의미 (Meaning)
Reconstruction Error	원본 → 압축 → 복원했을 때, 원본과 얼마나 달라졌는가?
Autoencoder	PCA의 딥러닝 버전. 입력을 그대로 출력하도록 학습하는 신경망.
Sparse Matrix	대부분이 0이나 비어있는(NaN) 행렬. (유저는 대부분의 영화를 안 봄)
Latent Factor	데이터 뒤에 숨어있는 진짜 원인. (예: 장르 선호도, 영화의 분위기)
Collaborative Filtering	"너랑 비슷한 사람이 이거 좋아했으니 너도 좋아할 거야."

12.1.1 1. PCA의 머신러닝적 접근: 재구성 오차

□ 개념 1: 파일 압축과 해제

한 줄 요약: 좋은 압축이란, 압축을 풀었을 때 원본과 가장 똑같은 상태여야 합니다. 손실(Error)을 최소화하는 압축 방법을 찾습니다.

1) 통계학 vs 머신러닝 관점 비교

- **통계학 (18.6501):** "데이터가 가장 넓게 퍼진(분산이 큰) 방향을 찾자."
- **머신러닝 (6.86x):** "데이터를 저차원 평면에 수직으로 내리꽂았을 때(투영), **이동 거리가 가장 짧은(오차가 적은)** 평면을 찾자."

결국 피타고라스 정리에 의해 두 목표는 **수학적으로 동치**입니다. (분산 최대화 $\iff$ 재구성 오차 최소화)

2) 목적 함수 (Objective Function)

$$\text{Minimize } J = \frac{1}{N}\sum_{n=1}^{N} ||x_n - \hat{x}_n||^2$$

- x_n: 원본 데이터 (고차원)
- $\hat{x}_n$: 압축했다가 다시 복원한 데이터 (재구성된 값)
- 의미: 원본과 복원본 사이의 거리 제곱 합을 최소화하는 축(Principal Component)을 찾습니다.

12.1.2 2. 추천 시스템: 행렬 분해 (Matrix Factorization)

□ 개념 2: 빈칸 채우기 퍼즐

한 줄 요약: 거대한 구멍 숭숭 뚫린 행렬을 두 개의 꽉 찬 작은 행렬의 곱으로 쪼개서, 빈칸의 값을 추론합니다.

1) 문제 상황: 희소 행렬 (Sparse Matrix)

넷플릭스에 영화가 1만 개 있다고 칩시다. 한 유저가 본 영화는 기껏해야 100개입니다. 평점 행렬 R은 99%가 비어 있습니다 (NaN). 우리의 목표는 이 빈칸에 들어갈 예상 평점을 맞추는 것입니다.

2) 핵심 원리: $R \approx U \times V^T$

평점 행렬 R $(N \times M)$을 두 개의 행렬로 분해합니다.

- U **(User Matrix,** $N \times K$**):** 유저들의 **잠재 취향** (예: 액션 선호도, 로맨스 선호도...)
- V **(Item Matrix,** $M \times K$**):** 영화들의 **잠재 특성** (예: 액션 지수, 로맨스 지수...)

3) 예측 (Prediction)

유저 u가 영화 i에 줄 예상 평점 $\hat{r}_{ui}$는 두 잠재 벡터의 **내적(Dot Product)**입니다.

$$\hat{r}_{ui} = \mathbf{u}_u \cdot \mathbf{v}_i = \sum_{k=1}^{K} u_{uk} v_{ik}$$

- **직관:** (내 액션 선호도 × 영화의 액션성) + (내 로맨스 선호도 × 영화의 로맨스성) = 총점
- 취향과 특성이 일치할수록(내적 값이 클수록) 높은 평점이 예측됩니다.

4) 최적화 (Optimization)

실제 평점이 있는 데이터에 대해서만 오차를 줄이도록 학습합니다.

$$\min_{U,V} \sum_{(u,i)\in \text{Observed}} (r_{ui} - \mathbf{u}_u \cdot \mathbf{v}_i)^2 + \lambda(\|U\|^2 + \|V\|^2)$$

주로 **ALS (Alternating Least Squares)** 알고리즘을 사용하여 U를 고정하고 V를 풀고, V를 고정하고 U를 푸는 방식을 반복합니다.

12.2 실전 시나리오: 넷플릭스 영화 추천

□ Scenario: 당신이 아직 안 본 '명작' 찾기

당신은 넷플릭스 엔지니어입니다. 유저 A가 '아이언맨', '어벤져스'에는 5점을 줬고, '노트북'에는 평점을 안 남겼습니다. A에게 '노트북'을 추천해야 할까요?

1. **잠재 요인 학습 ($K = 2$라고 가정):**
   - 차원 1: [액션/폭파], 차원 2: [감동/로맨스]
   - 행렬 분해를 해보니, 유저 A의 벡터 $u_A = [5.0, 0.2]$ (액션광)
   - 영화 '노트북'의 벡터 $v_{Nb} = [0.1, 4.8]$ (완전 로맨스)
2. **예상 평점 계산 (내적):**

$$\hat{r} = (5.0 \times 0.1) + (0.2 \times 4.8) = 0.5 + 0.96 = 1.46$$

3. **결정:** 예상 평점이 1.46점으로 매우 낮습니다. → **추천하지 않습니다.** (대신 액션 지수가 높은 '트랜스포머'를 추천합니다.)

12.3 자주 묻는 질문 (FAQ)

Q1. 잠재 요인(Latent Factor)이 무엇인지 컴퓨터가 알려주나요? A. 아니요, 컴퓨터는 그냥 숫자 벡터만 줍니다. $k = 1$번째 차원이 "액션"인지 "공포"인지는 사람이 결과를 보고 해석해야 합니다. 때로는 해석 불가능한 추상적인 특징일 수도 있습니다.

Q2. 콜드 스타트(Cold Start) 문제는 뭔가요? A. 새로 가입한 유저나 막 개봉한 영화는 데이터(평점)가 하나도 없어서 U나 V 벡터를 학습할 수 없는 문제입니다. 이 경우엔 협업 필터링 대신, 나이/성별 기반의 베스트셀러 추천(Top-N)이나 콘텐츠 기반 필터링을 섞어서 사용합니다.

Next Step: 지금까지는 주어진 데이터를 분석하는 데 집중했습니다. 그런데 AI가 데이터를 분석하는 것을 넘어, **새로운 데이터를 창조**할 수는 없을까요? 다음 **Unit 5:**

생성 모델 (Generative Models)에서는 데이터를 직접 만들어내는 VAE와 GAN에 대해 배웁니다.

Unit 4 (Part B) 핵심 요약

- **PCA:** 데이터의 재구성 오차를 최소화하는 평면을 찾아 차원을 축소한다.
- **행렬 분해 (MF):** $R \approx UV^T$. 희소 행렬을 분해하여 숨겨진 패턴을 찾는다.
- **협업 필터링:** 유저와 아이템의 잠재 벡터 내적을 통해 선호도를 예측한다.
- **활용:** 넷플릭스, 유튜브, 쇼핑몰 등의 개인화 추천 시스템의 핵심 원리.

Chapter 13

MIT 6.86x: 시행착오의 과학

Course Structure & Current Focus

- Unit 1 4: Supervised/Unsupervised Learning (정적 데이터 학습)
- **Unit 5: Reinforcement Learning (현재 단원: 동적 상호작용 학습)**
  - 13.1 MDP Framework (S, A, P, R, γ)
  - 13.2 Bellman Equations (가치의 정의)
  - 13.3 Solving MDPs: Policy vs Value Iteration
- Unit 5 (Part B): Q-Learning (환경을 모를 때)