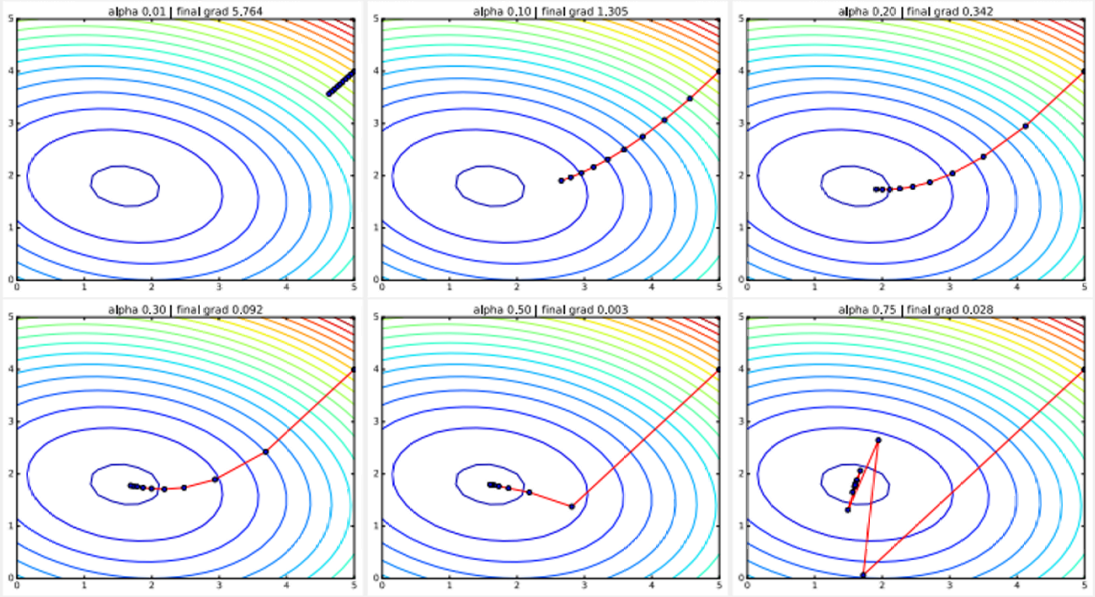
We run gradient descent for 10 iterations within initial position $(5, 4)$, while varying the learning rate in the range $\gamma = \{0.01, 0.1, 0.2, 0.3, 0.5, 0.75\}$.

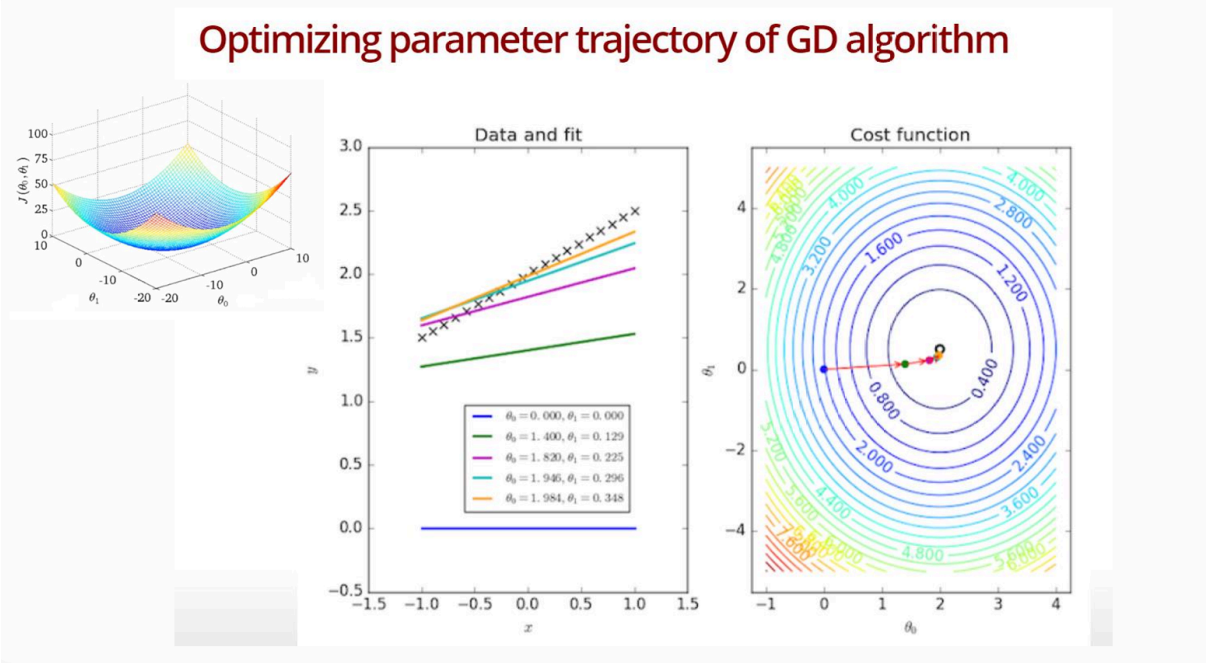


같은 타원형 함수 하나 위에서 학습률만 다르게 해서 여러 경사하강법 경로를 비교한 시각화

타원형 함수가 없으면 내부로 접근(수렴)하는지 파악이 어려움

경사하강법은 함수의 기울기를 따라
함수 값을 줄이도록 반복 이동하며,
지역 최소값(local minimum) 또는 전역 최소값(global minimum) 중 하나를 찾으려고 시도하는 알고리즘

f(x,y)=ax2+by2(a,b>0) 중심점이 최솟값



① 왼쪽 위 **3D** 그래프**:** 경사하강법이 이 곡면 위를 따라 내려가면서 최소값을 찾아간다**"**

- 비용 함수 **(Cost Function)** 를 나타냅니다. 일반적으로 MSE (Mean Squared Error)를 사용
- 모양은 볼록한 타원형으로, 중앙이 최솟값 (global minimum)을 의미합니다.

② 가운데 그래프 **(Data and fit):**

검은색 **'x'**들: 실제 데이터 포인트 (입력 xxx, 타깃 yyy)
색깔 있는 직선들: 경사하강법 과정 중 특정 단계에서의 모델 예측 결과

- 실제 데이터 (x, y) 위에 다양한 시점에서의 모델이 예측한 직선 (hypothesis)을 그린 것입니다.
- 범례에 있는 값들이 학습 과정 중 점진적으로 바뀌는 파라미터들을 보여줍니다.

③ 오른쪽 그래프 **(Cost function**의 등고선**):**

- **회귀예측 (Regression)**
  - MSE

$$J(\theta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

기호	뜻
$J(\theta)$	전체 비용 함수 (Cost Function)
n	전체 샘플 개수
y_i	**i번째 실제 정답 값 (label)**
$\hat{y}_i$	**i번째 예측 값** = $\theta_0 + \theta_1 x_i$ (선형 모델)
θ	모델 파라미터 집합 (ex: θ_0, θ_1)

각 데이터 포인트마다:

$$(y_i - \hat{y}_i)^2$$

→ 실제 값과 예측 값의 차이를 **제곱해서 오차**를 구함

$$J(\theta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

은 **정확히 말하면 "Sum of Squared Errors (SSE)"**, 즉 **제곱 오차의 총합**입니다.
MSE(Mean Squared Error) 라면 평균을 의미하니까, 분모에 n이 들어가야 합니다:

✅ 진짜 MSE 공식:

$$\boxed{\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

- **이진 분류 (Binary classification)**
  - Binary crossentropy

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y_i log(h_\theta(x_i)) + (1 - y_i) log(1 - h_\theta(x_i))$$

- $y_i = 1$: 정답이 **positive (참)** 라는 뜻
- $y_i = 0$: 정답이 **negative (거짓)** 라는 뜻
- $h_\theta(x_i)$: 모델이 해당 샘플이 정답(=1)일 **확률로 예측한 값** (ex. 0.9 → "90% 확률로 정답이다")

🔸 2. 시그모이드를 적용하면 확률처럼 변함

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

이 함수를 통과시키면:

- $z = 3 \rightarrow h(x) \approx 0.95$
- $z = -2 \rightarrow h(x) \approx 0.12$

즉, **시그모이드 함수가 "확률처럼 보이는 출력값"을 만들어주는 역할**을 해.

▶ $y_i = 1$인 경우

$$\text{Loss} = -\log(h_\theta(x_i))$$

- 이 말은 **모델이 정답(1)이라고 예측한 확률** $h_\theta(x_i)$ 가 **높을수록 손실이 낮아진다**는 뜻
- 예시:
  - $h_\theta(x_i) = 0.9$ → Loss = -log(0.9) ≈ 0.105
  - $h_\theta(x_i) = 0.1$ → Loss = -log(0.1) ≈ 2.30 ❌ 손실 큼
- ✅ **모델이 맞게 예측할수록 손실이 작아지고, 틀릴수록 손실이 커짐**

예측 함수 (Object function) 정의하기

▶ Softmax

$$h_\theta(x_i) = \frac{e^{zi}}{\sum_{j=1}^{k} e^{z_j}}, k = 1, 2, \dots M$$

- k개의 클래스
- i 번째 클래스
- $h_\theta(x_i)$ 는 i 번째 클래스일 확률

k = 3 인 경우

$$softmax(z) = [\frac{e^{z_1}}{\sum_{j=1}^{3} e^{z_j}}, \frac{e^{z_2}}{\sum_{j=1}^{3} e^{z_j}}, \frac{e^{z_3}}{\sum_{j=1}^{3} e^{z_j}}] = [p_1, p_2, p_3]$$

$$h_\theta(x_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}, \quad k = 1, 2, \dots, M$$

이건 다음을 의미합니다:

- z_i: 각 클래스에 대한 **로짓(logit)** — 즉, 선형 레이어를 거친 출력값
- e^{z_i}: 해당 클래스의 로짓에 **지수 함수**를 취한 것
- 분모: 모든 클래스의 e^{z_j}를 더한 값
- 전체는: **해당 클래스의 확률**을 구하는 식

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

- z_i: i번째 클래스의 로짓 (아직 확률 아님)
- e^{z_i}: i번째 클래스의 **상대적인 강도(점수)**를 지수 함수로 변환
- $\sum_{j=1}^{k} e^{z_j}$: 전체 클래스의 점수 총합
- → 결국, **i번째 클래스의 "점수 비율"**

✅ 왜 지수 함수(exp)를 쓸까?

1. 음수든 양수든 모두 양수로 만들기 위해

- z_i는 음수일 수도 있는데, 확률은 **항상 0~1** 사이여야 하니까
- e^{z_i}는 항상 양수!

2. 큰 로짓에 훨씬 더 큰 영향력을 주기 위해 (비선형 효과)

- 예: $z = [1, 2, 3] \rightarrow e^z = [2.7, 7.4, 20.1]$
- 큰 값일수록 **더 압도적인 점수**로 표현됨
- → **확실한 판단을 더 강조하는 효과**

✅ 요점 먼저

> **지수함수 e^x**를 쓰는 이유는:
>
> - "연산적으로 **매끄럽고 미분이 잘 되며**,"
> - "**상대적 차이를 자연스럽게 강조**할 수 있기 때문입니다."

그리고 **밑이 e**인 이유는:

> e^x는 자기 자신으로 미분되는 **유일한 함수**이고,
> 이는 **수학적 간결성 + 역전파(backprop)**에서 매우 유리합니다.

문제 유형	출력층 설정	활성화 함수	손실 함수 (Loss)	비고
이진 분류 (Binary Classification)	`nn.Linear(?, 1)`	`Sigmoid` (`torch.sigmoid`)	`BCELoss` `BCEWithLogitsLoss` **(추천)**	BCEWithLogitsLoss는 Sigmoid를 자동 적용
다중 분류 (Multi-class Classification)	`nn.Linear(?, num_classes)`	없음 (Softmax 내부 적용)	`CrossEntropyLoss`	CrossEntropyLoss 안에 Softmax 포함
다중 레이블 분류 (Multi-label Classification)	`nn.Linear(?, num_labels)`	`Sigmoid`	`BCELoss` `BCEWithLogitsLoss` **(추천)**	레이블별 독립적인 0/1 예측
회귀 (Regression)	`nn.Linear(?, 1)`	없음 (Raw output 사용)	`MSELoss`, `L1Loss`, `SmoothL1Loss`	수치(실수) 예측
다중 회귀 (Multi-output Regression)	`nn.Linear(?, n_outputs)`	없음	`MSELoss`, `L1Loss`, `SmoothL1Loss`	여러 실수 예측
순서 회귀 (Ordinal Regression)	출력 뉴런 여러 개	`Sigmoid` 또는 특별한 변형	`OrdinalCrossEntropyLoss` 등	레어 케이스

📌 딥러닝 회귀와 OLS의 차이점

항목	딥러닝 회귀 (`nn.Linear` + `MSELoss`)	OLS
방식	경사하강법으로 반복 최적화	해석적으로 한 번에 계산
표현력	비선형 모델도 가능	선형 모델
속도	느릴 수 있음 (반복)	빠름 (작은 데이터에 적합)
확장성	매우 유연함 (딥러닝 아키텍처 전체로 확장 가능)	단순한 선형 회귀에 특화됨