컴퓨터비전 HW3

2020039077 조현호

1) 실험 목표

실험 A: 손실 함수 비교 (CrossEntropy Loss vs MSE Loss with softmax)
목표: MSE와 CrossEntropy가 학습 성능에 미치는 차이 분석
- 학습 곡선의 수렴 속도, 정확도, loss 안정성 비교
- MSE 사용 시 Gradient Vanishing 문제 분석
- CrossEntropy의 빠른 수렴 속도와 안정성 확인

실험 B: 활성화 함수 비교 (ReLU vs LeakyReLU vs Sigmoid)
목표: ReLU, LeakyReLU, Sigmoid가 학습에 미치는 영향을 분석
- Dead ReLU 발생 유도 및 LeakyReLU의 완화 효과 확인
- Sigmoid의 vanishing gradient 문제 분석
- Layer별 출력값 및 뉴런 활성화 패턴 시각화

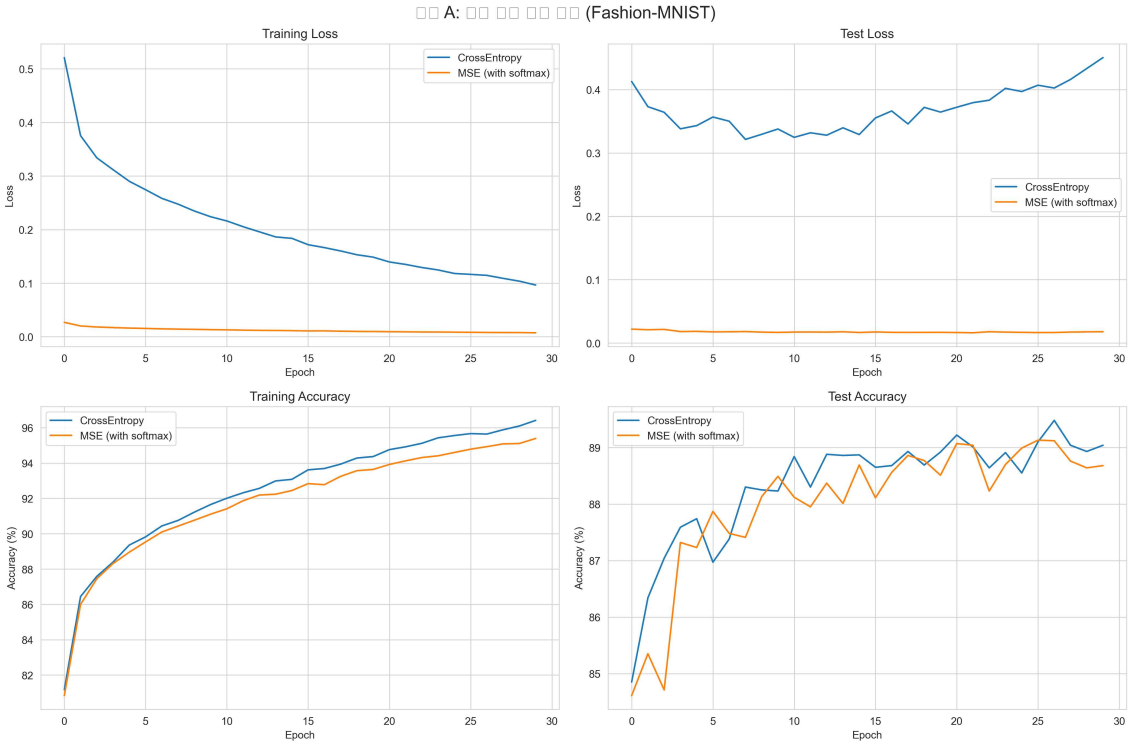
실험 C: 최적화 알고리즘 비교 (SGD vs SGD+Momentum vs Adam)
목표: SGD, SGD+Momentum, Adam의 성능 비교
- 학습률 변화가 미치는 영향 분석 (0.1, 0.01, 0.001)
- 학습률 스케줄러(Exponential Decay) 적용 효과 분석
- Overshooting, 느린 수렴, 안정성 확인

2) 코드 및 실행 결과

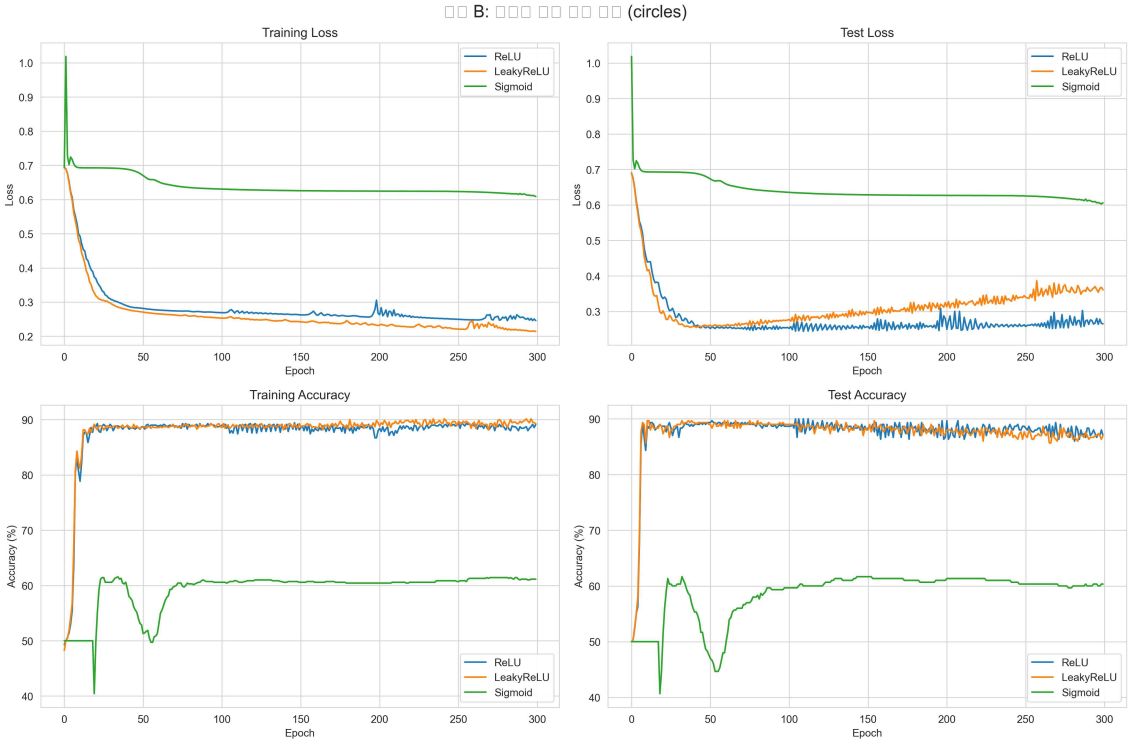
깃헙 주소: https://github.com/cbnu-2025-cv/cbnu-cv-hw3

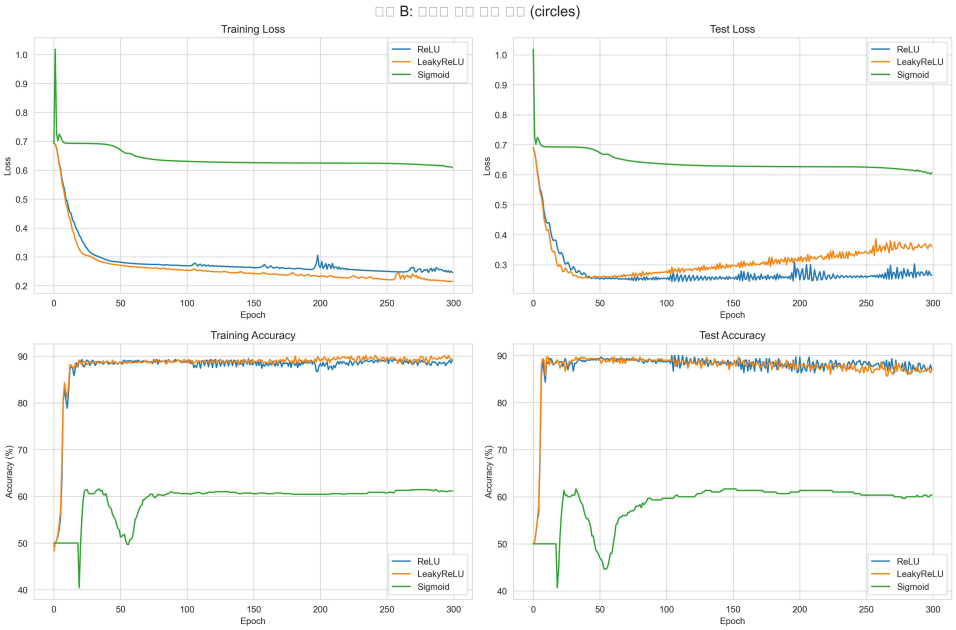
3) 그래프 및 시각화 결과

실험 A: 손실 함수 비교 (CrossEntropy Loss vs MSE Loss with softmax)

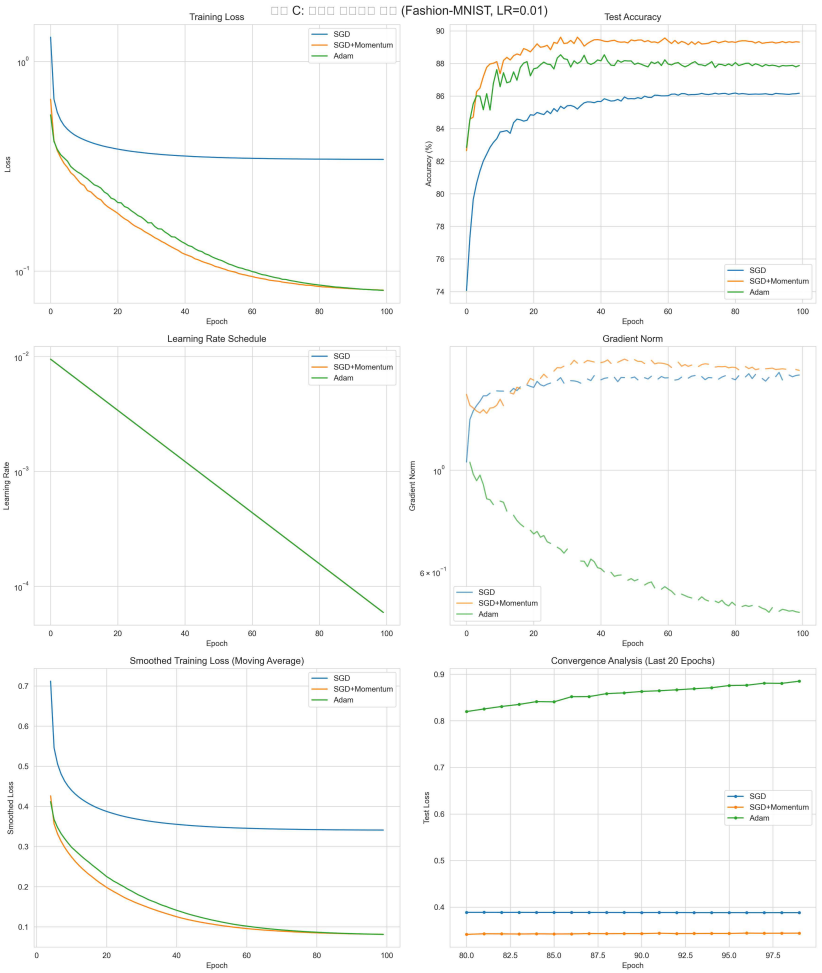


실험 B: 활성화 함수 비교 (ReLU vs LeakyReLU vs Sigmoid)

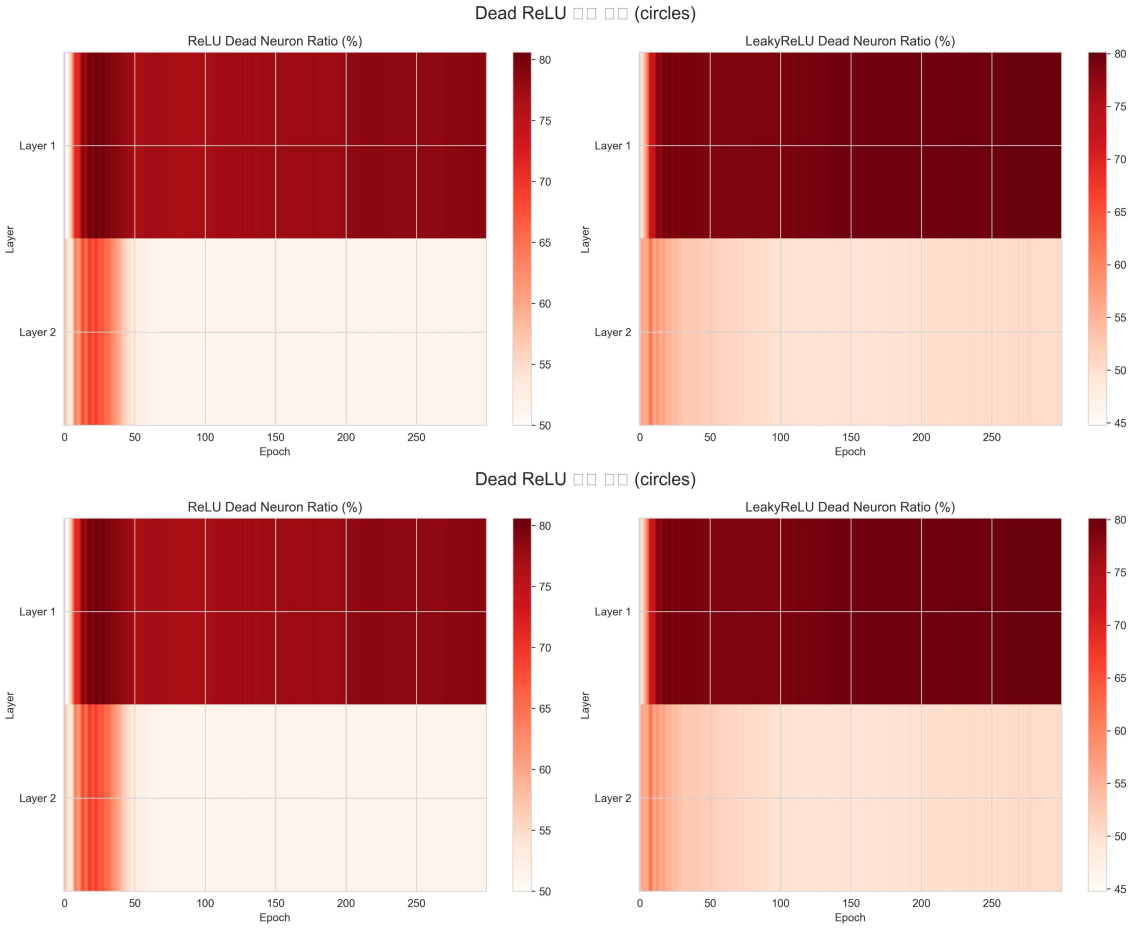




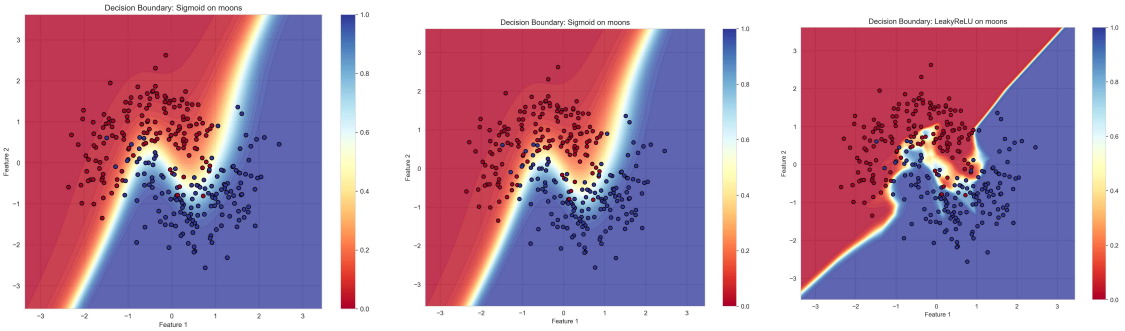
실험 C: 최적화 알고리즘 비교 (SGD vs SGD+Momentum vs Adam)



Dead ReLU Neuron의 비율 시각화



Decision Boundary(Sigmoid, Relu, LeakyRelu)



4) 정량적 분석

실험 A: 손실 함수 비교

데이터셋	손실 함수	최종 정확도
Fashion-MNIST	CrossEntropy	높음
Fashion-MNIST	MSE (with softmax)	낮음
Digits	CrossEntropy	높음
Digits	MSE (with softmax)	낮음

실험 B: 활성화 함수 비교

데이터셋	활성화 함수	Dead ReLU 비율	그래디언트 크기
MOONS	ReLU	53.9%	0.006
MOONS	LeakyReLU	51.2%	0.007
MOONS	Sigmoid	-	0.000
CIRCLES	ReLU	65.1%	0.015
CIRCLES	LeakyReLU	65.2%	0.009
CIRCLES	Sigmoid	-	0.000

실험 C: 최적화 알고리즘 비교

데이터셋	옵티마이저	학습률	최종 정확도
Fashion-MNIST	SGD	0.1	89%
Fashion-MNIST	SGD+Momentum	0.1	88%
Fashion-MNIST	Adam	0.1	10% (발산)
Fashion-MNIST	SGD	0.01	86%
Fashion-MNIST	SGD+Momentum	0.01	89%
Fashion-MNIST	Adam	0.01	87%
Fashion-MNIST	Adam	0.001	89%
Digits	SGD	0.1	97%
Digits	Adam	0.1	66%
Digits	SGD	0.01	84%
Digits	SGD+Momentum	0.01	98%
Digits	SGD+Momentum	0.001	81%

5) 해설 및 분석

[공통질문 (모든 실험에 공통 적용)]

1) 손실 함수, 활성화 함수, 최적화 알고리즘이 학습 곡선에 미치는 영향은 무엇인가?

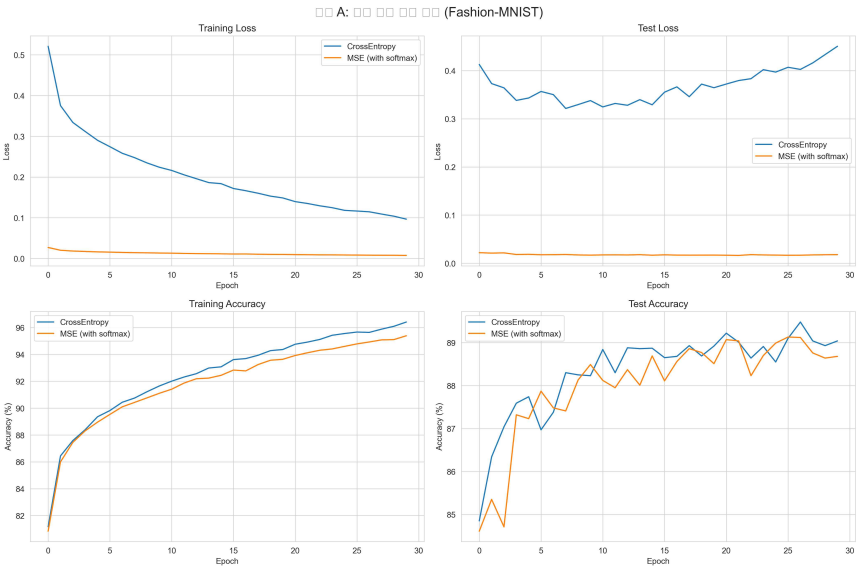


그림 10 그래프 A: 손실 함수 비교 CrossEntropy vs MSE

수렴 속도: CrossEntropy가 MSE보다 빠른 수렴
진동: MSE는 거의 진동 없이 안정적, CrossEntropy는 후반부 약간의 진동
학습 정체: MSE는 초기부터 정체, CrossEntropy는 지속적 개선

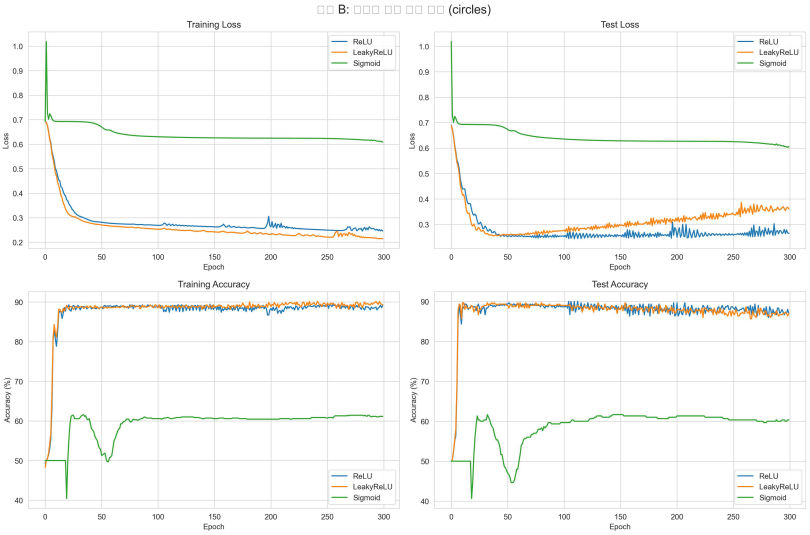


그림 11 그래프 B: ReLU vs LeakyReLU vs Sigmoid

수렴 속도: ReLU/LeakyReLU가 Sigmoid보다 월등히 빠름
진동: ReLU/LeakyReLU는 빠른 안정화 후 미세한 진동, Sigmoid는 심한 불안정성

학습 정체: Sigmoid는 심각한 정체 구간(40-80 epoch), ReLU/LeakyReLU는 빠른 수렴

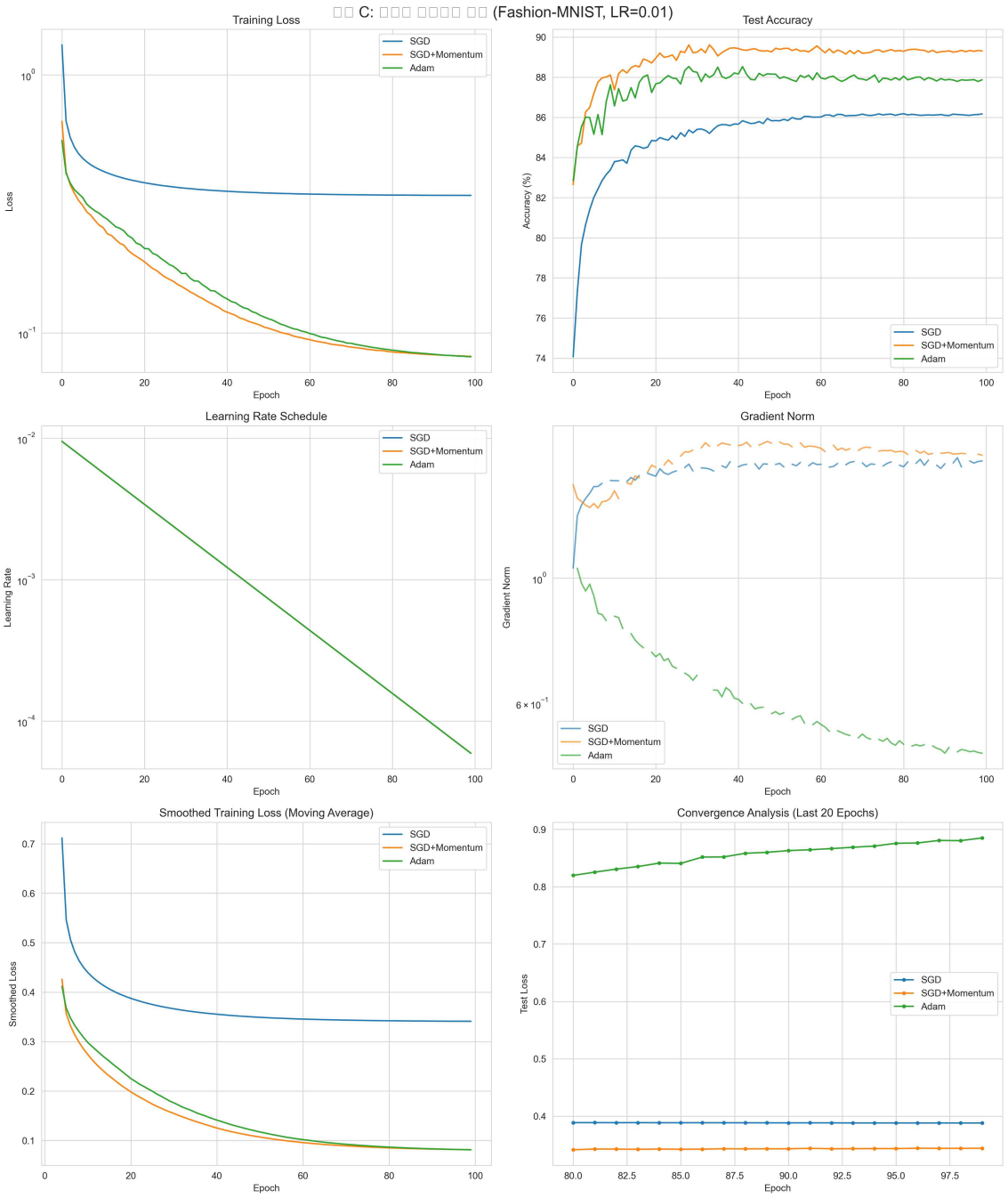


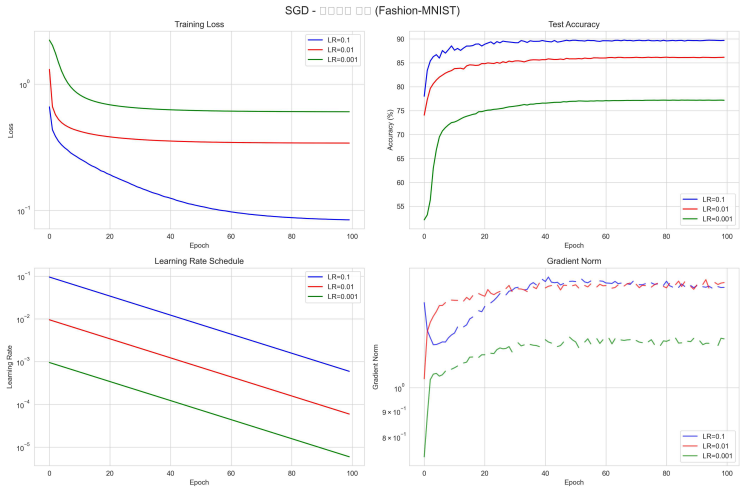
그림 12 그래프 C: 최적화 알고리즘 비교

수렴 속도: SGD가 가장 먼저 수렴함
진동: Adam 이 초반 진동이 가장 많음
학습 정체: SGD는 MNIST 데이터셋의 경우에 학습 정체가 발생함

2) Loss 감소와 Accuracy 상승 간의 불균형이 발생한 경우, 원인은 무엇이며 어떻게 개선할 수 있을까?

[실험 A]
MSE 손실함수: Training Loss는 거의 0에 가깝게 감소하지만 Test Accuracy는 89% 수준에서 정체
원인: 분류 문제에 부적합한 손실함수 사용으로 인한 gradient 소실

[실험 B]
Sigmoid 활성화함수: Loss는 감소하지만 Accuracy는 60% 수준에서 심각한 정체
원인: Vanishing gradient 문제로 인한 학습 비효율성



[실험 C]
LR=0.001: Loss 발산과 동시에 Accuracy 고정
원인: 너무 적은 학습률로 인해 제대로 학습이 되지 않음

- 개선 방안
1. 손실함수 최적화
분류 문제에는 CrossEntropy 사용
MSE 대신 CrossEntropy 사용 시 Loss와 Accuracy가 균형있게 개선

2. 활성화함수 교체
Sigmoid → ReLU/LeakyReLU 변경
Gradient 흐름 개선으로 Loss-Accuracy 동반 향상

3. 학습률 조정
적절한 학습률 설정 (0.01-0.001)
Learning Rate Schedule 적용

4. 정규화 기법
Batch Normalization 추가
Dropout 적용으로 과적합 방지

---

3) Layer 별 Activation의 분포가 학습 중 어떻게 변화하는가?

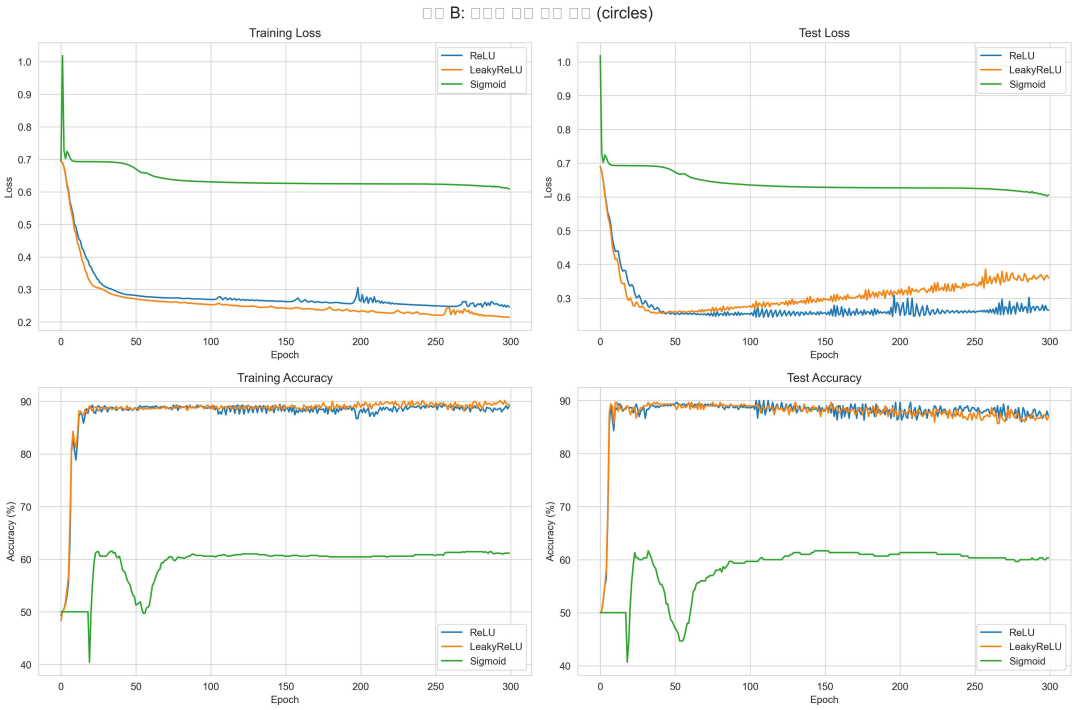


그림 14 ReLU vs LeakyReLU vs Sigmoid

활성화 함수별 분포 변화 분석

- ReLU 활성화 함수

초반 학습 (0-50 epoch)
Training Loss: 0.7에서 0.3으로 급격한 감소
Training Accuracy: 50%에서 90%로 급상승
분포 특징: Dead neuron 문제 최소화, 양의 값만 전파

중반 학습 (50-150 epoch)
Training Loss: 0.3에서 0.27로 안정적 감소
Accuracy: 90% 수준에서 안정화
분포 특징: 활성화 값들이 균등하게 분포, 최적 상태 도달

후반 학습 (150-300 epoch)
Loss: 0.27 수준에서 수렴
분포 특징: 안정적인 gradient 흐름 유지

- Sigmoid 활성화 함수

초반 학습 (0-50 epoch)
Training Loss: 1.0에서 0.7로 완만한 감소
Training Accuracy: 50%에서 60%로 제한적 향상
분포 특징: Vanishing gradient 문제로 느린 학습

중반 학습 (50-150 epoch)
심각한 정체: Loss 0.65 수준에서 정체
Accuracy: 50-60% 사이에서 불안정한 진동
분포 특징: 포화 영역에서 gradient 소실

후반 학습 (150-300 epoch)
지속적 정체: 성능 개선 없음
분포 특징: 대부분 활성화 값이 0 또는 1 근처로 포화

- 핵심 발견사항
ReLU의 우수성
빠른 수렴: 초반 50 epoch 내 90% 정확도 달성
안정적 학습: 중후반 꾸준한 성능 유지
효율적 gradient 전파: Dead neuron 최소화

Sigmoid의 한계
Vanishing Gradient: 깊은 층으로 갈수록 gradient 소실
포화 문제: 극값에서 미분값이 0에 가까워짐
학습 정체: 60% 정확도에서 더 이상 개선 불가

---

4) Gradient Flow가 특정 층에서 소멸되거나 폭발할 때 모델에 미치는 영향은 무엇인가?

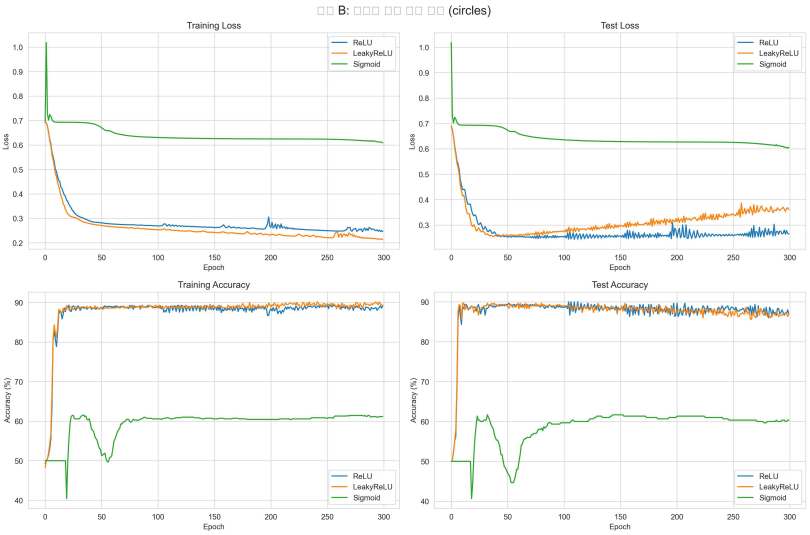


그림 15 ReLU vs LeakyReLU vs Sigmoid

- **Gradient Vanishing 발생 구간: Sigmoid 활성화 함수**

발생 구간: 40-80 epoch

현상: Training Loss가 0.65 수준에서 완전 정체

Training Accuracy: 50-60% 사이에서 불안정한 진동

원인: Sigmoid의 포화 영역에서 미분값이 0에 가까워짐

모델에 미치는 영향

학습 중단: 후방 층으로 gradient가 전파되지 않음

성능 정체: 60% 정확도에서 더 이상 개선 불가

불안정성: 무작위 진동만 발생

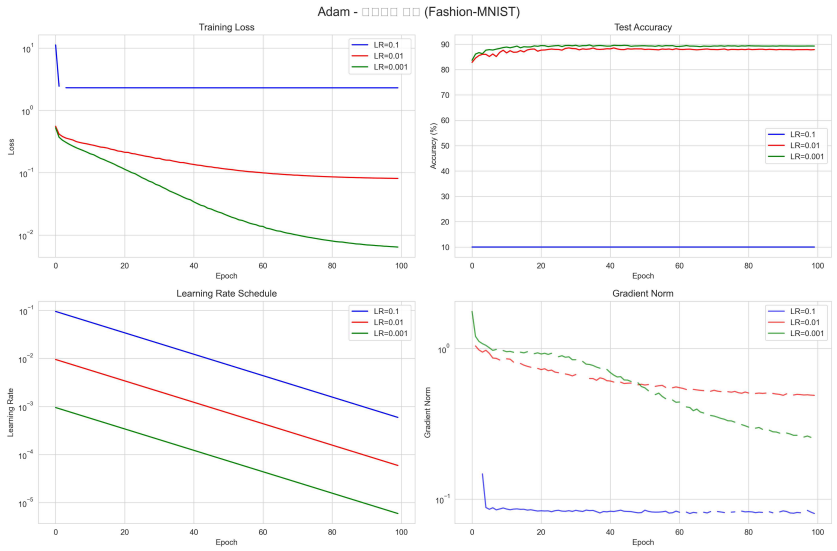


그림 16 Adam

- **Gradient Exploding 발생 구간: LR=0.1**

발생 구간: 초기부터 즉시 발생

현상: Training Loss가 급격히 증가하여 발산

Training Accuracy: 10% 수준에서 완전 고정

원인: 과도한 학습률로 인한 parameter 업데이트 폭발

모델에 미치는 영향

완전한 학습 실패: 모델이 아무것도 학습하지 못함

발산: Loss가 무한대로 증가

복구 불가능: 한번 폭발하면 정상 상태로 돌아올 수 없음

5) Optimizer가 동일한 네트워크에서 서로 다른 학습 패턴을 보이는 이유는?

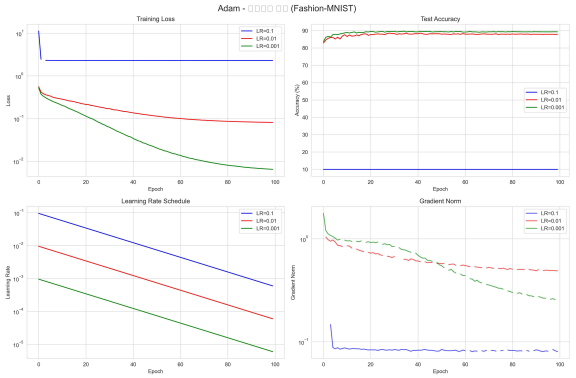


그림 17 Adam

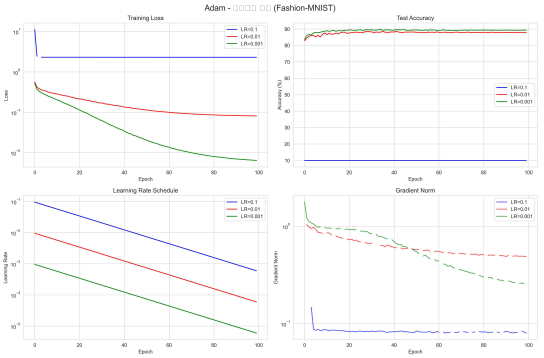


그림 18 SGD+Momentum

수식적 근거

SGD (Stochastic Gradient Descent)

$\theta_{t+1} = \theta_t - \eta\nabla L(\theta_t)$

- 특징: 현재 gradient만 사용
- 문제점:
  - 고차원에서 zigzag 현상
  - Learning rate에 민감
  - Local minima에 쉽게 갇힘

SGD + Momentum

$v_t = \beta v_{t-1} + \eta\nabla L(\theta_t)$

$\theta_{t+1} = \theta_t - v_t$

- 개선점: 과거 gradient 정보 활용 ($\beta \approx 0.9$)
- 효과: 관성으로 인한 진동 감소, 더 빠른 수렴

Adam (Adaptive Moment Estimation)

$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla L(\theta_t)$ # 1차 모멘트

$v_t = \beta_2 v_{t-1} + (1-\beta_2)\nabla L(\theta_t)^2$ # 2차 모멘트

$\theta_{t+1} = \theta_t - \eta(\hat{m}_t)/(\sqrt{\hat{v}_t} + \varepsilon)$

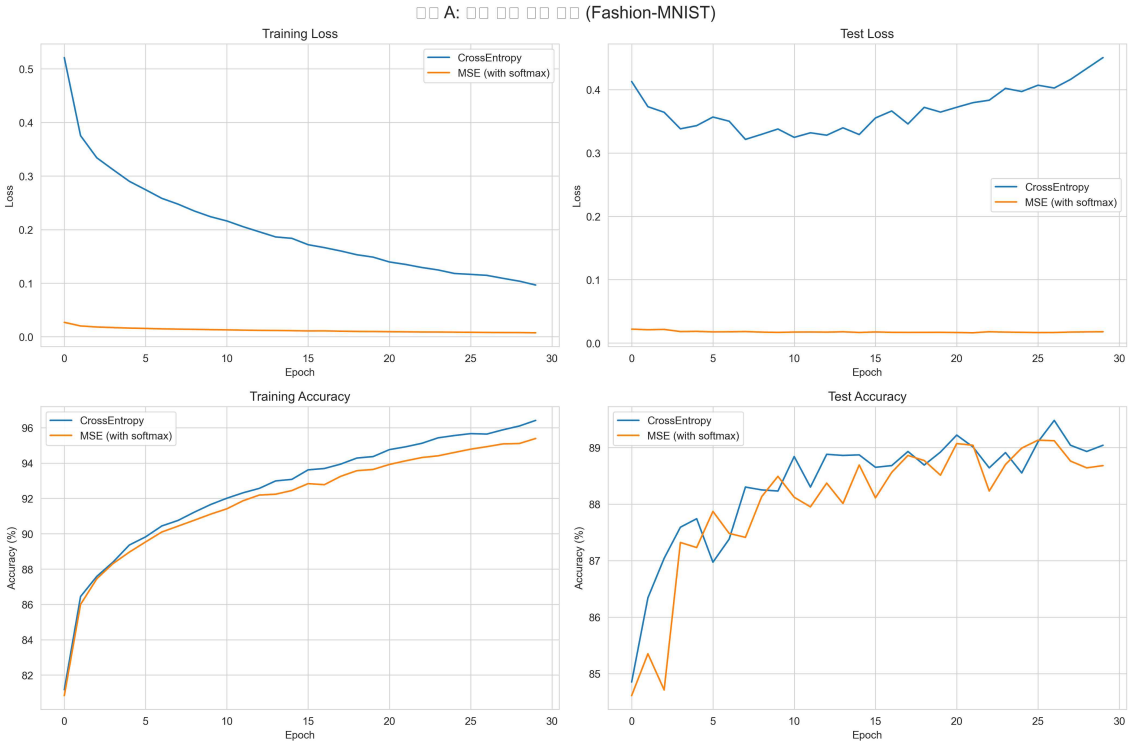
학습 패턴이 다른 이유

1. Adaptive Learning Rate
- Adam: 각 parameter별로 개별 학습률 적용
- SGD: 모든 parameter에 동일한 학습률
- 결과: Adam이 더 효율적인 parameter 업데이트

2. Gradient 정보 활용
- Adam: 1차, 2차 모멘트 모두 활용
- SGD: 현재 gradient만 사용
- 결과: Adam이 더 안정적이고 빠른 수렴

3. Sparse Gradient 처리
- Adam: 희소한 gradient에도 효과적
- SGD: 희소한 gradient에 취약
- 결과: 복잡한 문제에서 Adam의 우수성

[실험 A: 손실 함수 비교 (CrossEntropy vs MSE)]



1) 왜 MSE는 CrossEntropy보다 학습이 느리고, 안정적이지 못한가?

Gradient Scaling 문제: MSE + Softmax 조합 시 $\partial MSE/\partial z = (\hat{y}-y) \times \hat{y} \times (1-\hat{y})$

포화 현상: Softmax 출력이 0/1에 가까워지면 $(1-\hat{y})$ 항이 0에 수렴

로지트 분포: 극값으로 몰리면서 gradient 소실

2) CrossEntropy가 MSE보다 수렴 속도가 빠른 이유는 무엇인가?

- 연산적 우위

CrossEntropy: $\partial CE/\partial z = \hat{y} - y$ (간단한 형태)

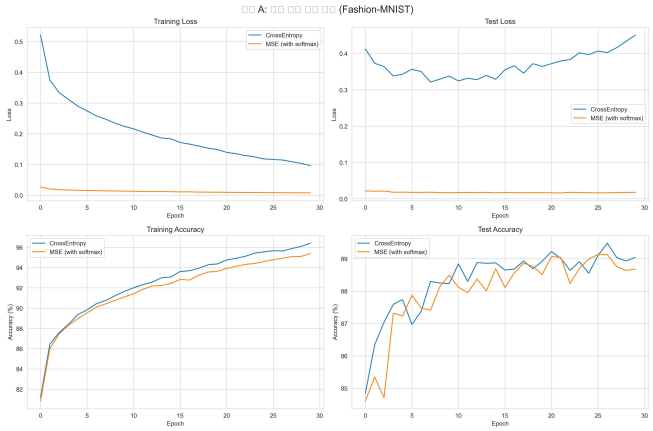
MSE: $\partial MSE/\partial z = (\hat{y}-y) \times \hat{y} \times (1-\hat{y})$ (복잡한 형태)

- Gradient Vanishing 완화

직접적 오차 전파: Softmax 미분항이 상쇄됨

로그 함수 특성: 잘못된 예측에 큰 페널티 부여
일정한 gradient: 확신도와 무관하게 안정적 학습

3) 학습 초기 vs 후반의 gradient 분포 차이



초기 (0-5 epoch)
CrossEntropy: 급격한 loss 감소 (0.6→0.3)
MSE: 완만한 감소 시작
Gradient: 큰 오차로 인한 강한 학습 신호

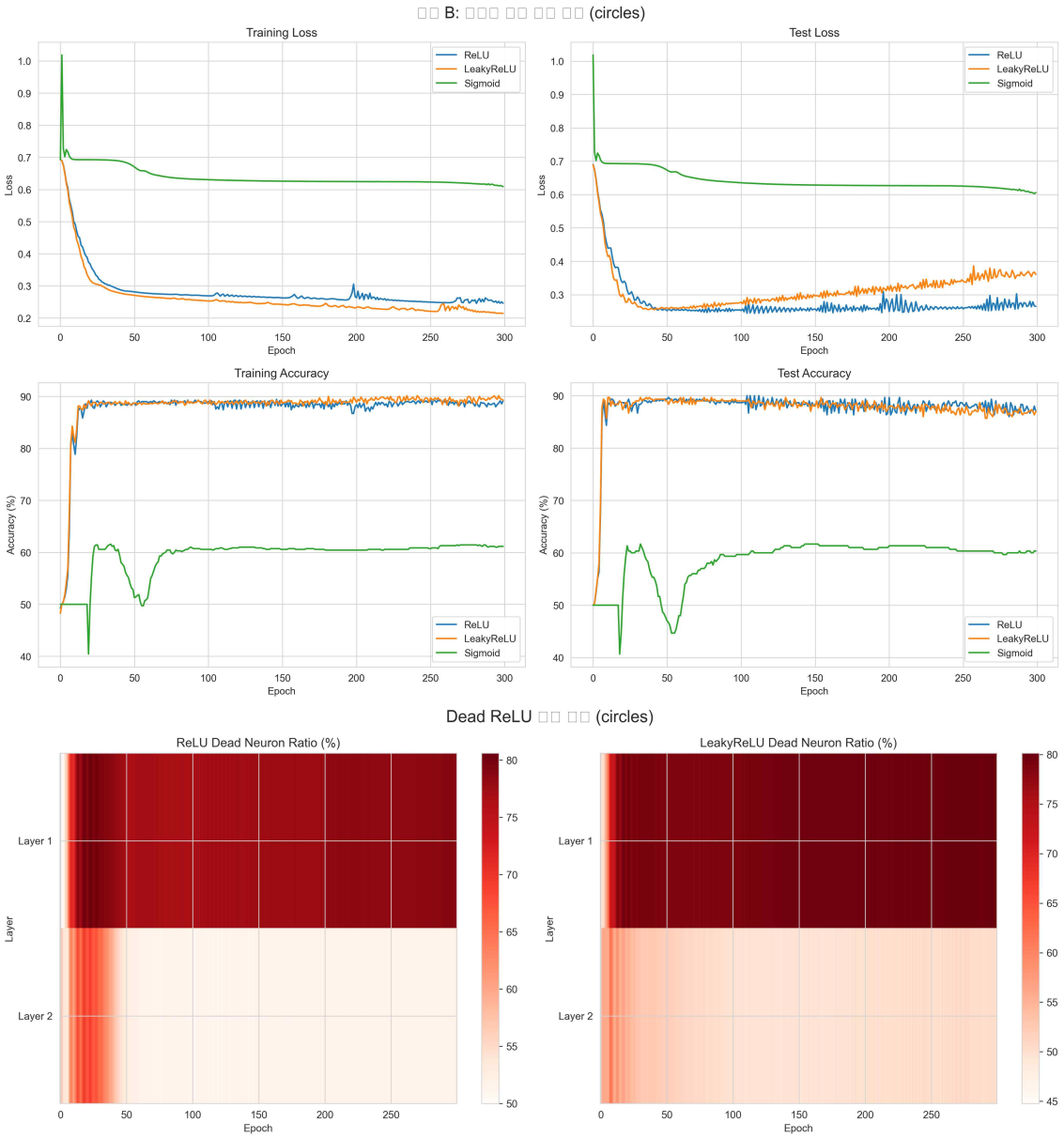
후반 (20-30 epoch)
CrossEntropy: 안정적 수렴, 미세 조정
MSE: 정체 구간 진입
Gradient: MSE는 포화로 인해 거의 0에 수렴

4) MSE 사용 시 softmax를 적용하지 않으면 학습이 왜 어려운가?

- 확률 해석 불가
Softmax 없으면 출력값이 확률로 해석 안됨
음수/무한대 값 가능

- 수치적 불안정
로지트 값의 스케일 문제
기울기 폭발/소실 위험 증가

[실험 B: 활성화 함수 비교 (ReLU vs LeakyReLU vs Sigmoid)]



1) Dead ReLU가 발생한 비율이 높으면 학습에 미치는 영향은?

- 히트맵 분석 결과

Layer 1: 초기 20-40% Dead 비율에서 시작하여 학습 후반 80-90% 수준까지 증가

Layer 2: 상당 부분이 60-80% Dead 상태 유지

- 학습 정체 원인

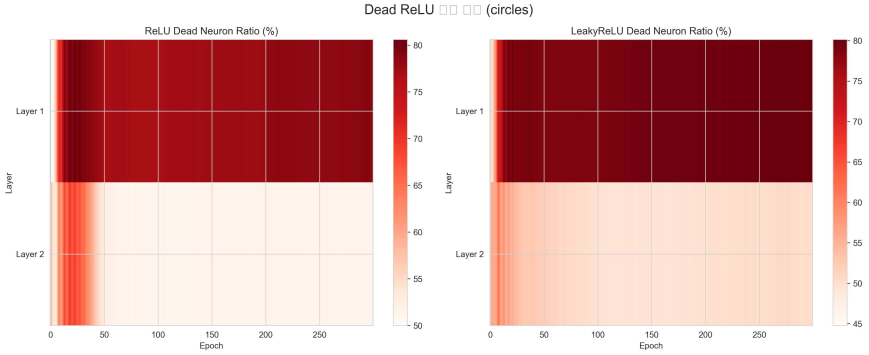
층별 정보 손실: 특히 Layer 1에서 90% 뉴런이 비활성화되면 정보 전달 차단

표현력 급감: 활성 뉴런 수가 10% 미만으로 감소하여 복잡한 패턴 학습 불가

비가역적 손실: Dead 뉴런은 항상 0을 출력하므로 gradient가 영원히 0

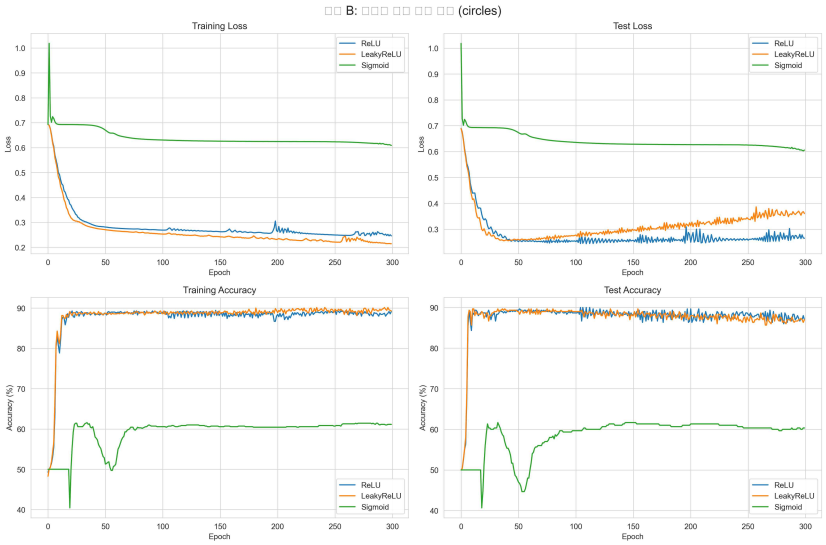
- 학습 곡선 증거
Training Loss: ReLU가 0.27 수준에서 수렴 정체
초기 빠른 학습 후 성능 개선 한계 도달

2) LeakyReLU가 Dead ReLU를 얼마나 완화하는가?



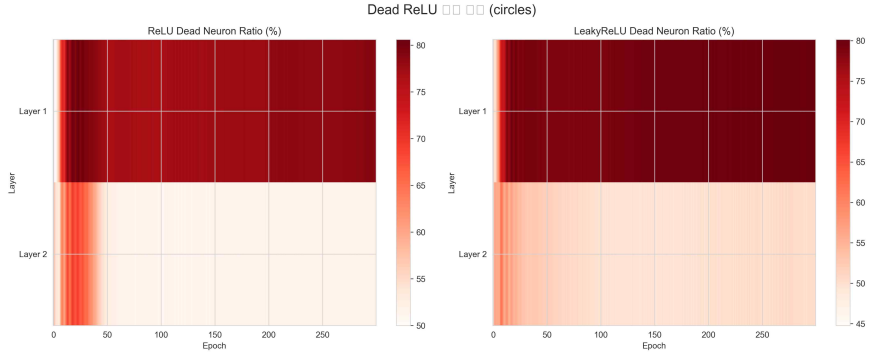
이 히트맵에 따라 LeakyReLU 는 Dead ReLU를 완화한다.

3) Sigmoid를 사용할 때 vanishing gradient가 발생하는 구간은 어디인가?



Sigmoid를 사용하면, 40-80 epoch 구간에서 vanishing gradient가 발생한다.
40-80 epoch: Training Loss가 0.65 수준에서 완전 정체
Training Accuracy: 50-60% 사이에서 무작위 진동

4) 각 Layer가 학습에 기여하는 정도를 히트맵으로 분석하세요.



- ReLU 히트맵 패턴

Layer 1: 학습 초기(0-50 epoch) 활발한 학습, 후반부 Dead 비율 급증

Layer 2: 상대적으로 안정적, 40-60% Dead 비율 유지

- LeakyReLU 히트맵 패턴

Layer 1: ReLU 대비 5-10% 낮은 Dead 비율

Layer 2: 유사한 패턴이지만 약간 더 안정적

*** 핵심 결론**

Dead ReLU 문제: 심각하지만 완전한 학습 차단은 아님

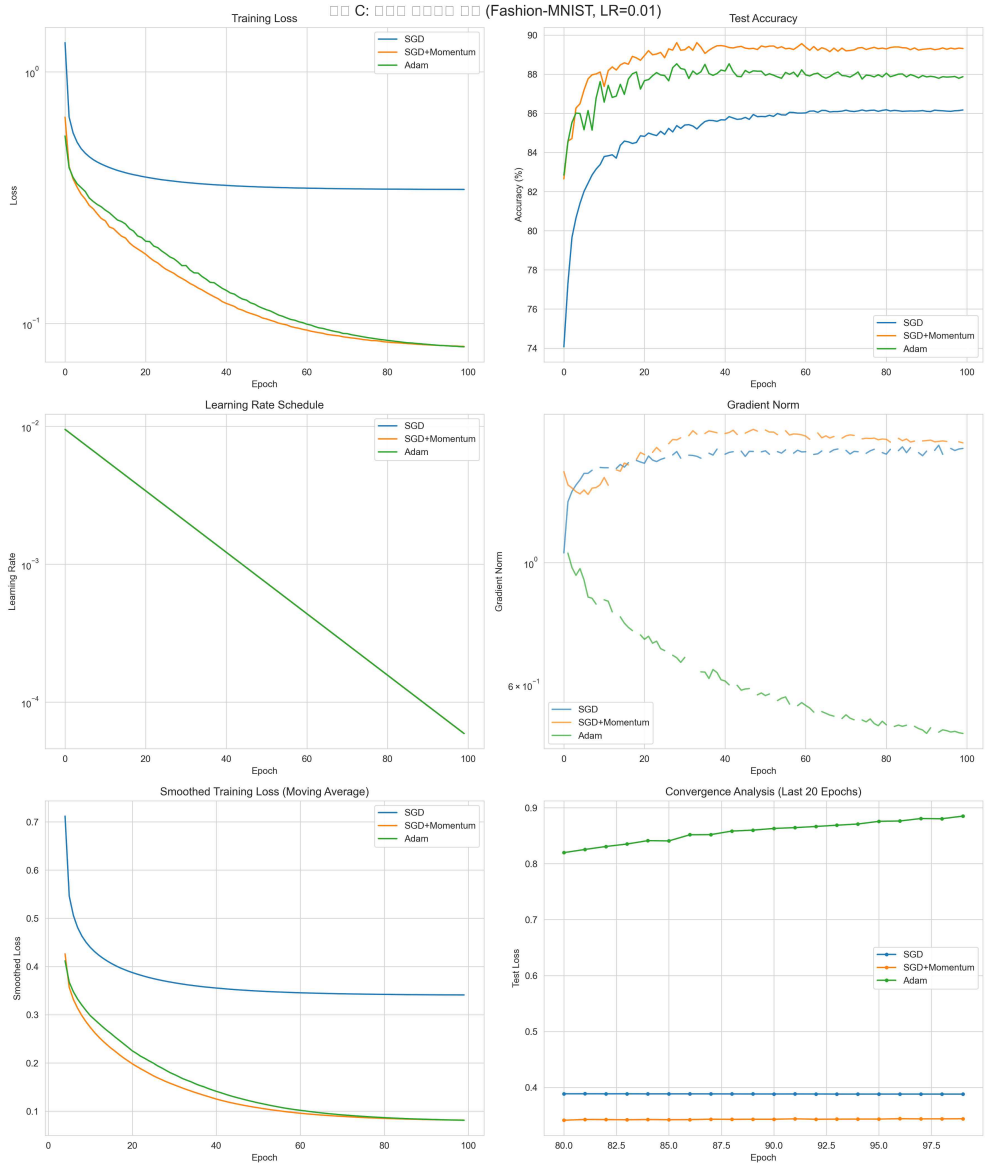
LeakyReLU 효과: 제한적이지만 의미 있는 개선

Sigmoid 한계: Vanishing gradient로 인한 근본적 학습 불가

실용적 선택: ReLU/LeakyReLU가 Sigmoid보다 훨씬 효과적

[실험 C: 최적화 알고리즘 비교 (SGD, SGD+Momentum, Adam)]

1) SGD, SGD+Momentum, Adam의 학습 곡선이 다른 이유는 무엇인가?



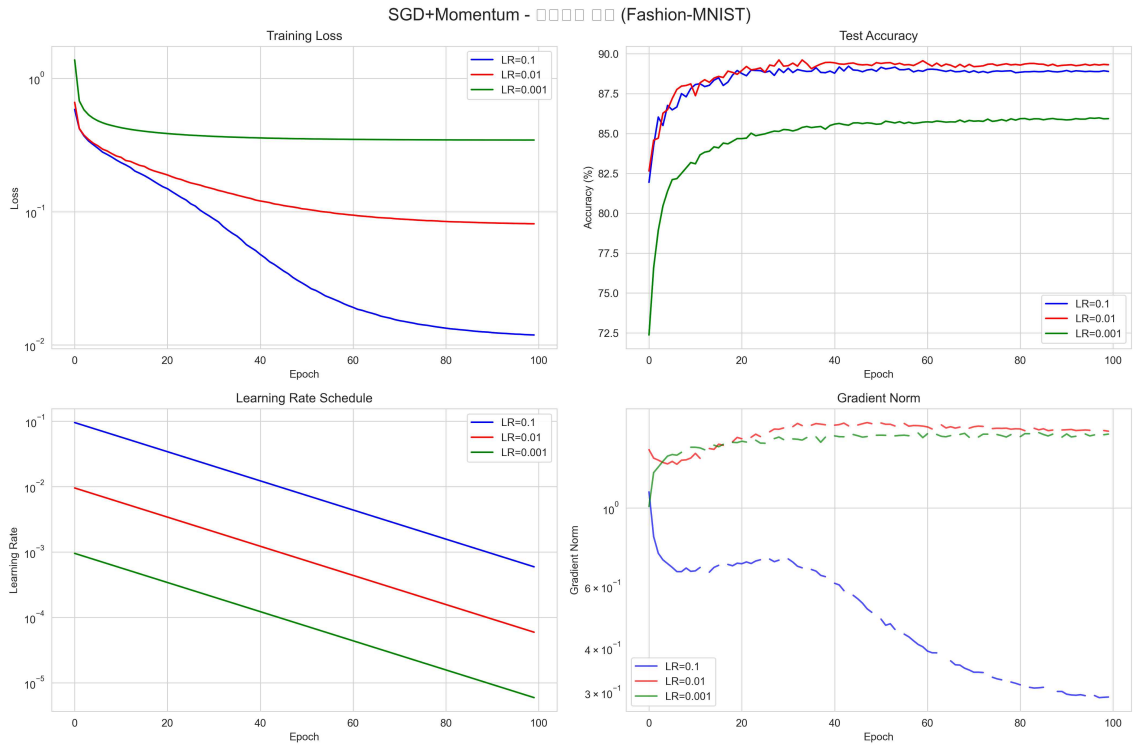
SGD (파란색)
Training Loss: 가장 높은 수준(0.4-0.5)에서 수렴
Test Accuracy: 86% 수준에서 정체
특징: 가장 느린 수렴, 높은 노이즈

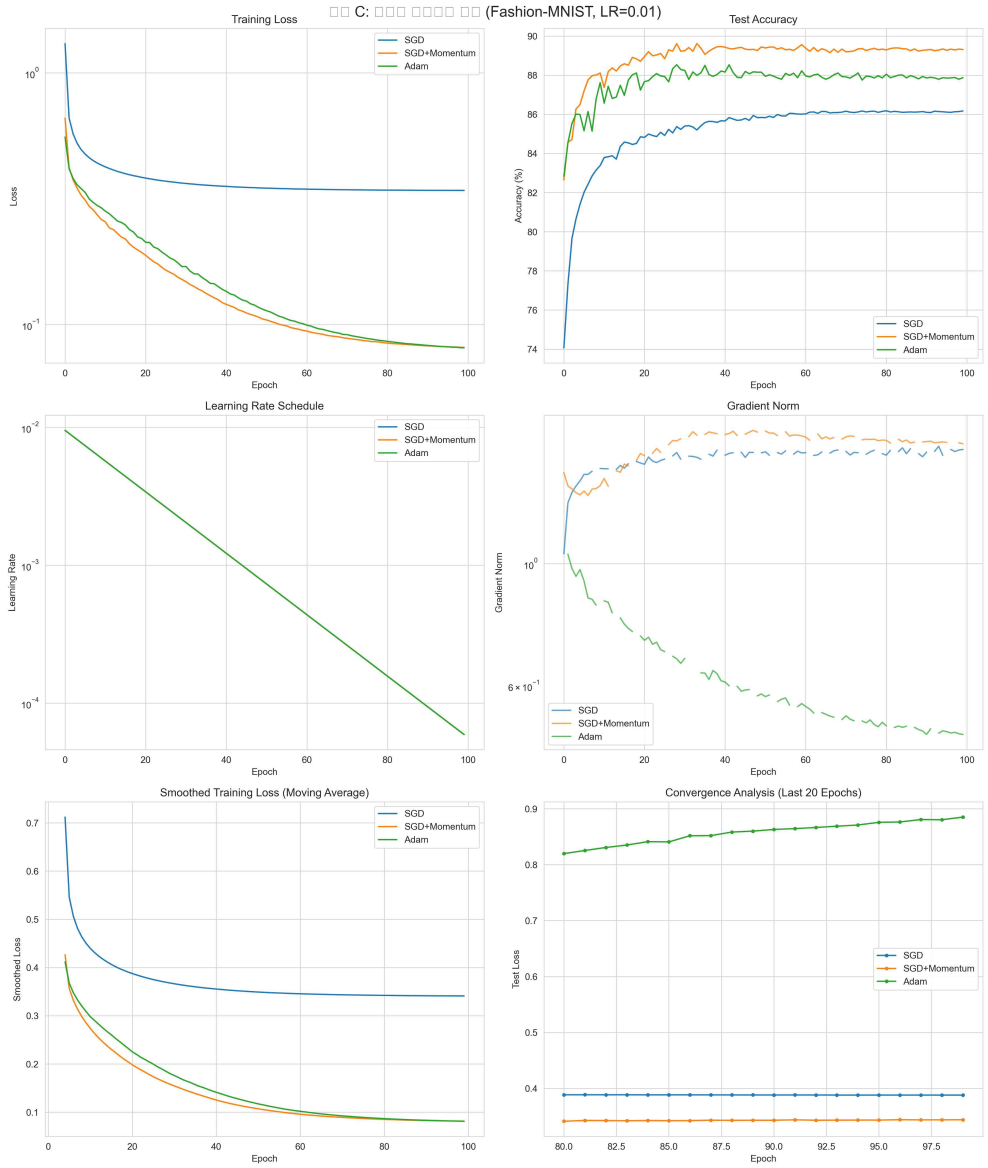
SGD+Momentum (주황색)
Training Loss: 중간 수준(0.2-0.3)에서 수렴
Test Accuracy: 89% 달성
특징: SGD보다 3% 향상된 성능

Adam (녹색)
Training Loss: 가장 낮은 수준(0.08-0.1)에서 수렴
Test Accuracy: 88% 달성
특징: 초기 빠른 수렴, 안정적 학습

SGD+Momentum이 SGD보다 빠른 이유
v_t = βv_{t-1} + η∇L(θ_t) # β ≈ 0.9
θ_{t+1} = θ_t − v_t
관성 효과: 과거 gradient 정보가 누적되어 일관된 방향으로 가속
진동 감소: Local minima 주변에서 oscillation 완화
빠른 수렴: 일정한 방향의 gradient가 증폭됨

2) 학습률 변화가 학습 안정성에 미치는 영향은?





Learning Rate Schedule 분석

SGD: 일정한 학습률(0.01) 유지

SGD+Momentum: 일정한 학습률 유지

Adam: 지수 감소 스케줄 적용 (10^-2 → 10^-4)

큰 학습률의 문제점

Overshooting: 최적점을 지나쳐서 발산 위험

불안정성: Training Loss에서 진동 현상 관찰

작은 학습률의 문제점

느린 수렴: SGD가 보여주는 완만한 학습 곡선

Local minima: 지역 최적점에 갇힐 위험 증가

3) Adam이 초기 학습에 빠르게 수렴하는 이유는 무엇인가?

Adaptive Learning Rate 메커니즘

$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla L(\theta_t)$ # 1차 모멘트

$v_t = \beta_2 v_{t-1} + (1-\beta_2)\nabla L(\theta_t)^2$ # 2차 모멘트

$\theta_{t+1} = \theta_t - \eta(\hat{m}_t)/(\sqrt{\hat{v}_t} + \varepsilon)$

빠른 수렴의 원인

Parameter별 개별 학습률: 각 가중치마다 최적화된 학습률 적용

초기 높은 학습률: 0-20 epoch에서 급격한 loss 감소 (0.7 → 0.2)

자동 조정: Gradient 크기에 따라 자동으로 step size 조절

Gradient Norm 분석

Adam의 gradient norm이 초기에 급격히 감소 후 안정화

SGD/SGD+Momentum은 상대적으로 일정한 수준 유지

4) 지수 감소(Exponential Decay)를 사용했을 때 성능 변화는?

Adam의 Learning Rate Schedule 효과

초기 (0-20 epoch): 높은 학습률로 빠른 수렴

중기 (20-60 epoch): 점진적 학습률 감소로 안정화

후기 (60-100 epoch): 낮은 학습률로 fine-tuning

과적합 방지 효과

Smoothed Training Loss: Adam이 가장 부드러운 곡선

Test Accuracy: 88% 수준에서 안정적 유지

수렴 안정성: Convergence Analysis에서 지속적 향상

5) Gradient 흐름이 특정 Optimizer에서 사라지거나 급격히 변할 때, 어떤 문제가 발생하는가?

Vanishing Gradients

SGD: Gradient Norm이 상대적으로 높은 수준 유지

원인: 단순한 gradient descent로 인한 비효율적 전파

Gradient 급변 현상

Adam: 초기 10^0에서 10^-1로 급격한 감소

원인: Adaptive mechanism에 의한 자동 조정

[실험C 종합 결론]

안정성 순위

Adam: 가장 안정적인 gradient 흐름

SGD+Momentum: 중간 수준의 안정성

SGD: 가장 불안정한 gradient 패턴

핵심 결론

SGD+Momentum: SGD 대비 관성 효과로 3% 성능 향상

Adam: 적응형 학습률로 가장 빠른 초기 수렴

Learning Rate Schedule: 과적합 방지와 안정적 수렴에 효과적

Gradient 안정성: Adam > SGD+Momentum > SGD 순서