NumPy-기반 순수 구현 CNN으로 6 × 6 손글씨 (1 | 2 | 3) 분류

Numerical Optimization 과제 보고서

이승우
고려대학교 세종캠퍼스 인공지능사이버보안학과
rainup4632@gmail.com

요 약

본 과제는 합성곱신경망(CNN)을 순수 NumPy로 구현해 강의 예제 SimpleCNN(Sigmoid + MaxPool + SGD)을 재현하고, 시드 10회 평균 정확도를 측정하는 데 목적이 있다. 동일 구조에서 옵티마이저만 RMSprop·Adam으로 교체한 S-RMS, S-Adam과, 풀링을 제거하고 ReLU를 사용한 EnhancedCNN (E-RMS, E-Adam)을 같은 조건(학습:검증 = 8:2, 에폭 300, 시드 0 - 9)으로 10회 학습해 비교하였다.
실험 결과 S-Adam이 99.1 % ± 1.0 %p로 가장 높았고, Baseline(S-SGD)은 96.3 % ± 0.5 pp, S-RMS

모델	활성화	풀링	옵티마이저	평균 정확도 ± 표준편차
S-SGD	Sigmoid	MaxPool 2*2	SGD(η = 0.9)	0.9625 ± 0.0051
S-RMSprop	Sigmoid	MaxPool 2*2	RMSprop(η = 0.001)	0.9333 ± 0.0280
S-Adam	Sigmoid	MaxPool 2*2	Adam(η = 0.01, β_1 = 0.7, β_2 = 0.99)	0.9906 ± 0.0098
E-RMSprop	ReLU	-	RMSprop(η = 0.001)	0.8313 ± 0.1667
E-Adam	ReLU	-	Adam(η = 0.01, β_1 = 0.7, β_2 = 0.99)	0.8271 ± 0.2191

는 93.3 % ± 2.8 %p였다. 풀링을 제거한 E-계열은 평균 83 % 수준에 그쳐, 6 × 6 입력에서는 MaxPool 층이 잡음 제거와 일반화에 실질적인 기여를 함을 확인하였다. 부록 A에는 Conv, Pool, ReLU, Softmax-CE 역전파 수식을 단계별로 전개하였고, 부록 B에는 전체 NumPy 코드와 실행 스크립트를 수록하였다. 전체 소스는 GitHub(https://github.com/seungwoo-AI/cnn-from-scratch)에 공개되어 있다.

1. 서론

이 보고서의 1차 목표는 합성곱신경망(CNN)을 외부 딥러닝 프레임워크 없이, 오로지 NumPy만으로 작성하여 강의 시간에 다룬 역전파 수식을 손으로 구현·검증하는 데 있다. Conv2D, MaxPool2D, Sigmoid, ReLU, Soft-max-CrossEntropy, 그리고 기본 최적화 기법인 SGD를 모두 직접 코딩하고, 체인룰을 따라 레이어별 기울기를 완전히 전개함으로써 “수식이 곧 코드”임을 확인한다. 실습 데이터는 6 × 6 흑백 손글씨 숫자(1·2·3) 96개이며, 강의 예제로 주어진 SimpleCNN(Sigmoid + MaxPool + SGD)을 정확히 재현해 난수 시드 10개로 학습·평가하였다. 여기까지가 과제 요구사항의 전부다.

이후 부분은 개인적인 추가 실험이다. 동일 구조에서 옵티마이저만 RMSprop · Adam으로 교체(S-RMS, S-Adam)하여 적응형 학습률이 수렴 속도와 초기값 민감도에 주는 영향을 살펴보고, 풀링 층을 제거하고 ReLU를 사용한 EnhancedCNN(E-RMS, E-Adam)을 설계해 MaxPool 유무가 작은 입력(6 × 6)에서 성능에 미치는 효과를 정량적으로 비교하였다.

보고서는 먼저 데이터셋과 NumPy 구현 과정을 설명한 뒤, 필수 실험(SimpleCNN + SGD) 결과를 제시한다. 이어서 선택적으로 수행한 네 가지 변형 모델의 성능을 간략히 비교하고, 풀링 층 및 옵티마이저 종류가 가져온 차이를 논의한다. 마지막으로 부록에 역전파 수식과 핵심 코드를 수록하여 수식 → 구현 → 실험 → 해석 흐름을 한눈에

확인할 수 있도록 구성하였다.

2. 방법

2.1 데이터셋

실험 데이터는 강의 자료로 제공된 Excel 파일 한 장(시트 이름 “Data”)에 들어 있다. 한 샘플은 6 × 6 흑백 픽셀 블록과 그 아래 3 셀짜리 원-핫(one-hot) 라벨로 이루어져 있다. 첫 번째 샘플은 L열(12열)부터 Q열(17열)까지, 두 번째 샘플은 그보다 6 칸 오른쪽(R ~ W열), 이런 식으로 총 96 개가 가로로 이어진다.

util_excel.py는 openpyxl로 값을 읽어 0 - 255를 0 - 1로 정규화한 뒤, 입력 X를 (96, 1, 6, 6) 배열, 라벨 y를 (96,) 정수 배열로 반환한다. 이 형식은 곧바로 Conv2D에 넣을 수 있는 N, C, H, W 배치 구조와 동일하다.

2.2 Conv 모듈

합성곱 계층은 각각 가중치와 편향

$$W \in R^{C_{out} * C_{in} * K * K},\ b \in R^{C_{out}}$$

을 갖는다. 순전파에서 한 출력 픽셀은

$$Z_{n,o,i,j} = \sum_{c=0}^{C_{in}-1} \sum_{u=0}^{K-1} \sum_{v=0}^{K-1} W_{o,c,u,v} X_{n,c,i+u,j+v} + b_o,$$

즉 입력 창($K \times K$)과 가중치의 곱·합에 편향을 더한 값으로 정의된다. 구현은 배치 → 출력 채널 → 입력채널 → 공간 i → 공간 j 순서의 다섯 겹 for-loop로 직접 계산하였다.

역전파 단계에서는 먼저 가중치 기울기를

$$\frac{\partial L}{\partial W_{o,c,u,v}} = \sum_{n,i,j} dZ_{n,o,i,j} X_{n,c,i+u,j+v},$$

다음으로 편향 기울기를

$$\frac{\partial L}{\partial b_{oc}} = \sum_{n,i,j} dZ_{n,oc,i,j},$$

마지막으로 입력기울기를

$$dX_{n,c,p,q} = \sum_{o,i,j} dZ_{n,o,i,j} W_{o,c,p-i,q-j}.$$

순서로 구한다.

가중치는 Xavier 초기화 $N(0, 1/\sqrt{C_{\in} K^2})$로 뿌려 초기 단계에서 기울기 폭주를 방지하였다. 한편, MaxPool 2×2는 순전파 때 각 풀링 창의 최대값 좌표를 저장하고 역전파 시 해당 좌표에만 상위 기울기를 전달하도록 구현하였다.

함수	순전파	역전파
Sigmoid	$\sigma(x) = 1/(1+e^{-x})$	$dX = dY \cdot \sigma(1-\sigma)$
ReLU	max(0,x)	$dX = dY \cdot 1[x > 0]$

표 1. Sigmoid·ReLU 순전파와 역전파 식

2.3 손실 함수와 활성화

손실은 소프트맥스 정규화 후 교차 엔트로피이다. 수치 안정화를 위해 로짓에서 각 행의 최대값을 빼고 지수를 취한다. 편미분은

$$\frac{\partial L}{\partial z_{n,i}} = \frac{1}{N}(p_{n,i} - 1[i = y_n])$$

한 줄로 표현된다.

2.4 최적화 알고리즘

모델	옵티마이저 하이퍼파라미터
S-SGD	SGD, η=0.90
S-RMS	RMSprop, η=0.001, ρ=0.9
S-Adam	Adam, η=0.01, β_1=0.7, β_2=0.99
E-RMS	RMSprop, η=0.001, ρ=0.9
E-Adam	Adam, η=0.01, β_1=0.7, β_2=0.99

표 2. 각 모델에 적용한 옵티마이저와 하이퍼파라미터

optimizer.py에는 SGD, Momentum, NAG, Adagrad, RMSprop, Adam의 여섯 가지 알고리즘을 모두 구현했으나, 본 실험에서는 비교의 초점을 명확히 하기 위해 SGD와 두 적응형 방법(RMSprop, Adam), 총 세 가지만 사용하였다. 학습에 사용한 설정은 표 2의 내용과 같다.

- SGD

$$p \leftarrow p - \eta g$$

- RMSprop

$$r_t = \rho r_{t-1} + (1-\rho) g_t^2 \mathrm{d}$$

- Adam

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t,$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$$

$$\hat{m} = \frac{m_t}{1-\beta_1^t},\ \hat{v}_t = \frac{v_t}{1-\beta_2^t},$$

$$p \leftarrow p - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

모델	연산흐름	파라미터 수
S-SGD	입력 1 × 6 × 6 → Conv 3 × 3 (1→3, Sigmoid) → 4 × 4 → MaxPool 2 × 2 → 2 × 2 → Conv 2 × 2 (3→3, Sigmoid) → 1 × 1 → Softmax	74
S-RMSprop	위와 동일한 SimpleCNN 구조, optimizer만 RMSprop	74
S-Adam	위와 동일한 SimpleCNN 구조, optimizer만 Adam	74
E-RMSprop	입력 1 × 6 × 6 → Conv 3 × 3 (1→4, ReLU) → 4 × 4 → Conv 3 × 3 (4→4, ReLU) → 2 × 2 → Conv 2 × 2 (4→3, ReLU) → 1 × 1 → Softmax	92
E-Adam	위 EnhancedCNN 구조, optimizer만 Adam	92

표 3. S-SGD·S-RMS·S-Adam·E-RMS·E-Adam 다섯 모델의 연산 흐름과 파라미터 규모

모델	시드1	시드2	시드3	시드4	시드5	시드6	시드7	시드8	시드9	시드10
S-SGD	0.958	0.958	0.969	0.969	0.958	0.969	0.969	0.958	0.958	0.958
S-RMSprop	0.917	0.958	0.906	0.927	0.979	0.875	0.948	0.927	0.948	0.948
S-Adam	1.0	0.990	1.0	0.990	0.969	0.990	1.0	0.990	1.0	0.979
E-RMSprop	1.0	0.667	0.990	0.656	0.667	0.667	0.667	1.0	1.0	1.0
E-Adam	0.990	0.333	0.969	1.0	0.667	0.990	1.0	0.667	0.990	0.667

표 4. 다섯 모델에 대한 시드 1 ~ 10별 최종 정확도 (학습 : 검증 = 8 : 2, Epoch 300)

2.5 모델 구조

표 1은 이번 실험에 사용한 다섯 모델(S-SGD, S-RMS, S-Adam, E-RMS, E-Adam) 의 연산 흐름과 파라미터 규모를 한눈에 보여 준다. Simple 계열(S-*)은 강의안에서 제시된 SimpleCNN 구조를 그대로 따르며, 6 × 6 입력에 3 × 3 합성곱-Sigmoid - 맥스풀(2 × 2)-합성곱-Sigmoid를 차례로 적용해 해상도를 6 → 4 → 2 → 1로 줄인다. 이때 학습 파라미터는 가중치 68 개와 편향 6 개, 합계 74 개에 불과하다. 세 모델(S-SGD, S-RMS, S-Adam)은 네트워크가 완전히 동일하고, 표 1에 표시된 대로 옵티마이저만 각각 SGD, RMSprop, Adam으로 달리하여 적응형 학습률이 수렴 속도와 초기값 민감도에 미치는 영향을 관찰한다.

Enhanced 계열(E-*)은 맥스풀을 제거하고 3 × 3→3×3→2×2 세 단계 합성곱을 ReLU와 함께 연속 적용한다(표 1 가운데 부분 참조). 채널 수를 1→4→4→3으로 확장·축소하기 때문에 파라미터 수는 92 개로 소폭 증가한다. 두 모델(E-RMS, E-Adam)은 구조가 동일하지만 옵티마이저만 바꿔 풀링이 없는 상황에서 RMSprop과 Adam이 각각 어떻게 작동하는지 비교하도록 설계했다.

모든 모델은 학습 : 검증을 8 : 2로 층화 분할하고, 에폭 300, 난수 시드 0 - 9(총 10 회)라는 동일 조건으로 학습해 평균 정확도와 표준편차를 비교한다. 표 1의 '파라미터 수' 열을 보면, 구조가 단순한 SimpleCNN은 74 개로 매우 가볍고, 풀링을 제거한 EnhancedCNN은 92 개로 약간 늘어났음을 확인할 수 있다.

3. 실험 결과 및 분석

모든 실험은 학습·검증 비율을 8:2로 고정하고, 에폭 300, 배치 크기는 전체 샘플(96)을 사용하는 방식으로 진행하였다. 난수 시드 0부터 9까지 총 10회 반복 학습한 뒤, 최종 정확도의 평균과 표준

모델	평균 정확도 ± 표준편차
S-SGD	0.9625 ± 0.0051
S-RMSprop	0.9333 ± 0.0280
S-Adam	0.9906 ± 0.0098
E-RMSprop	0.8313 ± 0.1667
E-Adam	0.8271 ± 0.2191

표 5. 다섯 모델의 10회 평균 정확도와 표준편차

편차를 계산하였다. 실험 결과는 표 5에 정리되어 있다.

SimpleCNN 계열 중 S-Adam은 평균 정확도 99.06% ± 0.98%p로 가장 우수했으며, 표준편차도 가장 작아 초기 가중치에 대한 민감도가 낮았다. 베이스라인인 S-SGD는 96.25% ± 0.51%p로 안정적인 성능을 보였고, RMSprop을 사용한 S-RMSprop은 오히려 93.33% ± 2.80%p로 낮은 성능을 기록하였다. 이는 적응형 옵티마이저가 항상 성능을 보장하지는 않는다는 점을 보여준다.

맥스풀링을 제거한 EnhancedCNN 계열은 성능이 전반적으로 낮았다. E-RMSprop과 E-Adam은 각각 평균 83.13%, 82.71%에 그쳤고, 표준편차도 0.17~0.22 수준으로 시드에 따라 정확도 편차가 컸다. 6×6 입력처럼 해상도가 작은 경우, 맥스풀은 오히려 잡음을 줄여 일반화 성능을 높이는 데 도움이 될 수 있다는 점을 시사한다.

시드별 정확도는 표 4에 제시되어 있으며, 성능 일관성 차이를 잘 보여준다. S-Adam은 모든 시드에서 97% 이상을 기록한 반면, E-Adam은 일부 시드에서 정확도가 30%대로 급락하는 등 구조적 불안정성이 뚜렷하게 나타났다.

모델별 손실 곡선은 그림 1에 시각화되어 있다. 모든 모델은 초기 50 epoch 이내에 손실이 급격히 감소하고 이후 완만하게 수렴하였다. S-SGD는 진동 폭이 크고 수렴 속도가 느리며, S-RMSprop은 초반 손실 감소는 빠르지만 불안정한 흔들림이 있었다. S-Adam은 가장 매끄럽고 안정적인 수렴 곡선을 보였고, 정확도와도 일관된 결과를 나타냈다.

EnhancedCNN 계열은 손실이 안정적으로 감소했음에도 검증 정확도가 오르지 않아 구조적 한계가 드러났다. 이는 단순히 손실이 낮다고 해서 높은 정확도가 보장되지는 않으며, 수렴의 안정성과 네트워크 구조가 함께 작용한다는 점을 보여준다.

종합하면, 실험 결과는 옵티마이저, 구조적 설계(MaxPool 유무), 손실 수렴 특성이 모델 성능과 일반화 능력에 미치는 영향을 잘 드러낸다. 단순한 수치 비교를 넘어 학습 과정과 구조적 특성에 대한 해석이 가능했다는 점에서 의미 있는 분석이었다.

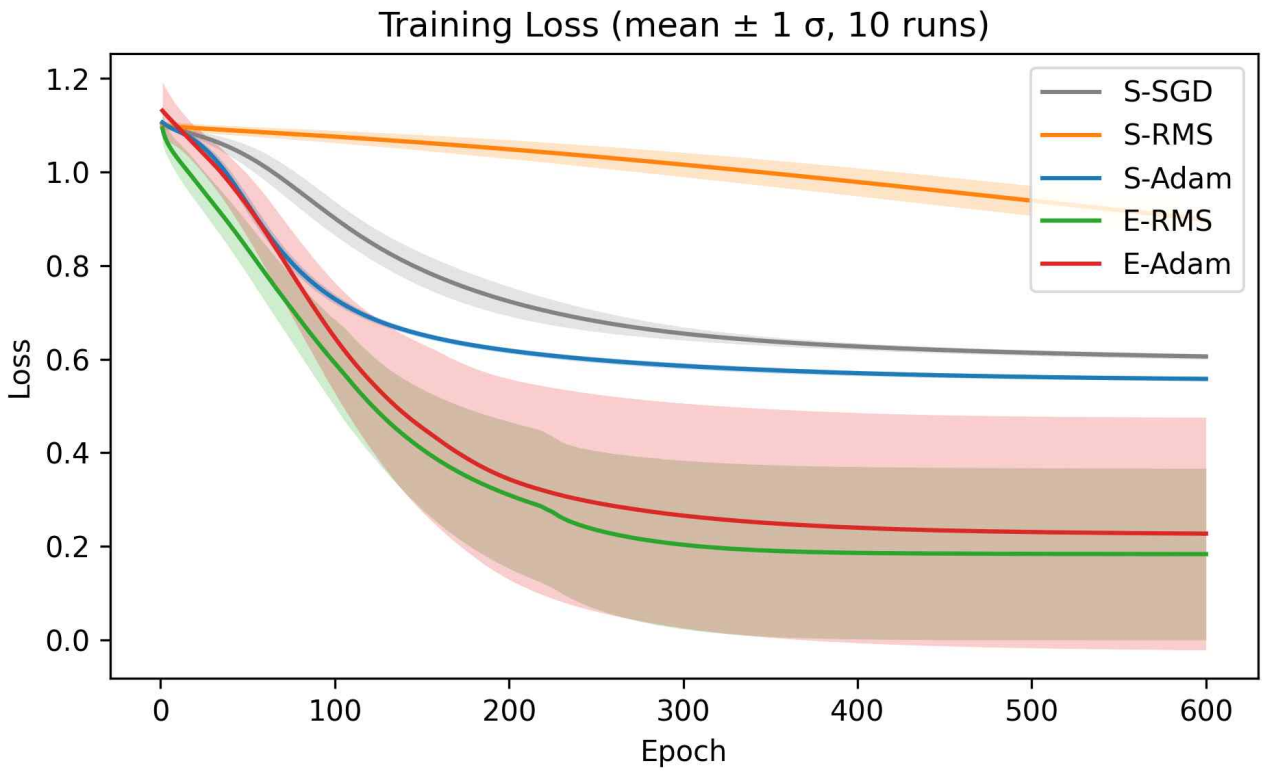

그림 1. 6 × 6 손글씨 데이터셋 학습 결과: SimpleCNN을 600 epoch 돌린 10회 반복 실험에서 얻은 훈련 손실(평균과 표준편차 음영). 다섯 가지 최적화 설정(S-SGD, S-RMS, S-Adam, E-RMS, E-Adam)을 비교하였다.

4. 결론 및 향후 과제

NumPy만으로 구현한 SimpleCNN 베이스라인은 평균 96% 이상의 정확도를 달성하며, 순수 파이썬 코드로도 충분한 분류 성능을 구현할 수 있음을 확인하였다. 동일한 구조에서 옵티마이저만 Adam으로 교체한 S-Adam은 평균 99%에 도달했고, 표준편차도 가장 낮아 안정적인 학습 특성을 보였다. 반면, 맥스풀링을 제거한 EnhancedCNN은 평균 정확도가 약 83% 수준으로 떨어지고 시드에 따른 편차도 커져, 작은 입력에서도 풀링이 일반화에 실질적으로 기여함을 확인할 수 있었다.

특히 손실 곡선의 진동 여부가 정확도에 큰 영향을 미쳤다는 점이 인상적이었다. 실제로 손실이 빠르게 줄더라도 진동이 큰 모델(S-SGD, S-RMSprop)은 수렴이 불안정했고, 반면 S-Adam처럼 손실이 부드럽고 안정적으로 감소한 경우 정확도도 가장 높았다. 이를 통해 loss의 절대값보다도 수렴 안정성이 모델 성능에 더 중요하며, 또한 6×6과 같은 작은 입력에서도 MaxPool이 유의미하게 작동할 수 있다는 점을 실험적으로 검증할 수 있었다.

이번 실험은 데이터셋이 96개로 매우 작고, 검증 샘플이 20개 미만이라는 점에서 한계가 있다. 또한 합성곱 연산을 5중 for-loop로 구현해 실행 속도가 느리고 GPU 가속을 활용하지 못했다. 향후에는 손글씨 샘플을 추가 수집하고 회전이나 잡음 삽입을 통한 데이터 증강을 적용할 계획이다. 또한, 배치 정규화와 드롭아웃을 도입하여 과적합을 완화하고, 옵티마이저 측면에서는 Adagrad나 Nesterov Momentum 등 다양한 방식과의 비교 실험을 진행할 예정이다.

마지막으로, 본 모델을 28×28 크기의 MNIST나 32×32 CIFAR-10과 같은 더 큰 이미지 데이터셋에 확장하고, Cython·Numba·CuPy 기반으로 핵심 연산을 가속화하여 CPU와 GPU 환경 모두에서 학습 속도를 개선하는 방향으로 발전시킬 계획이다.

5. 느낀점

직접 NumPy로 구현하며 CNN이 실제로 어떻게 작동하는지를 처음부터 끝까지 확인할 수 있었다. 코드를 짜는 동안 수식이 연산 흐름과 정확히 일치하는 걸 보며, 그동안 추상적으로만 알고 있던 딥러닝 구조가 구체적으로 와닿았다.

처음에는 6×6 입력에 MaxPool이 오히려 정보 손실만 일으킬 거라 생각했지만, 실험 결과는 정반대였다. 풀링을 포함한 SimpleCNN이 성능과 안정성 모두에서 더 우세했고, 작은 입력일수록 불필요한 세부 정보를 줄이는 과정이 오히려 효과적이라는 걸 알게 됐다.

또 하나 느낀 건, loss 값 자체보다 그 변화의 안정성이 정확도에 더 큰 영향을 준다는 점이다. S-Adam처럼 손실이 매끄럽게 수렴한 모델이 실제 정확도도 가장 높았다. 단순한 수치보다 그 흐름과 패턴을 읽는 것이 더 중요하다는 걸 실감했다.

아이러니하게도, 공부하고 이해가 깊어질수록 신경망이라는 구조가 더 복잡하고 어렵게 느껴진다. 처음엔 단순한 블록처럼 보였던 것들이 이제는 각 연산마다 가정과 제약, 한계가 달라 보인다. 점점 더 많은 걸 이해하게 되면서 동시에 더 많이 의심하게 되는 과정이라는 걸 느낀다.

부록 A - 역전파(Backpropagation) 수식전개

A.0 사전 준비 - 체인룰 복습

두 함수가 차례로 연결되어 있을 때,

$$u = g(x),\ y = f(u)$$

전체 미분은

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

CNN에서도 "위에서 온 기울기($\frac{\partial L}{\partial \text{출력}}$)×내 층의 미분($\frac{\partial \text{출력}}{\partial \text{입력}}$)"을 계속 곱하는 것뿐이다.

A.1 Softmax + Cross-Entropy 손실

1) 순전파

로짓(미가공 점수) $z_{n,i}$

→ 확률 $p_{n,i}$:

$$p_{n,i} = \frac{e^{z_{n,i}}}{\sum_{k=1}^{C} e^{z_{n,k}}}$$

원-핫 라벨 y_n (정답 인덱스). 미니배치 크기를 N이라고 하면 손실은

$$L = -\frac{1}{N}\sum_{n=1}^{N}\log p_{n,y_n}$$

2) 역전파

$$\frac{\partial L}{\partial z_{n,i}} = \frac{1}{N}(p_{n,i} - 1[i = y_n])$$

즉 "예측 확률 - 정답 원-핫"을 N으로 나누면 Softmax와 Cross-Entropy를 한 번에 미분한 결과가 된다. 여기서부터 다음 레이어로 기울기가 내려간다.

A.2 ReLU

순전파: a=max(0,z)

역전파는 두 경우로 나뉜다.

$z > 0$이면 기울기 그대로 통과

$z \le 0$이면 기울기 0

수식으로는

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \times 1[z > 0]$$

코드에서는 $mask = (z > 0)$를 미리 저장했다가 $dZ = dA * mask$로 한 줄에 처리한다.

A.3 MaxPool 2×2 (stride 2)

1) 순전파

각 2×2 창에서 최댓값 하나만 남겨서 특징맵 크기를 절반으로 줄인다. 동시에 "어느 좌표가 최댓값인지"를 기억해 둔다.

2) 역전파

최댓값 위치로만 기울기를 보내면 끝. 예를 들어 (i, j)창에서 (u^*, v^*)가 최댓값이었다면

$$\left(\frac{\partial L}{\partial A}\right)_{2i+u^*} = \left(\frac{\partial L}{\partial P}\right)_{i,j}$$

창 안의 나머지 세 칸에는 0이 들어간다.

A.4 Conv2D (다중 채널 포함)

이해를 돕기 위해 1채널 입력, 1필터(커널 크기 $K \times K$)부터 시작 후, 마지막에 다중 채널, 다중 필터로 일반화한다.

A.4.1 1채널 → 1필터 예시

1) 순전파

$$Z_{i,j} = \sum_{u=0}^{K-1}\sum_{v=0}^{K-1} W_{u,v} X_{i+u,j+v} + b$$

출력 공간 인덱스는 $i = 0...(H-K)$, $j = 0...(W-K)$

2) 역전파

(a) 편향 b

출력 모든 위치의 기울기를 더하면 된다.

$$\frac{\partial L}{\partial b} = \sum_{i,j}\frac{\partial L}{\partial Z_{i,j}}$$ (코드: $db = dZ.\sum()$)

(b) 가중치 W

각 커널 좌표(u,v)마다 "해당 커널과 겹친 입력값"×dZ를 합한다.

$$\frac{\partial L}{\partial W_{u,v}} = \sum_{i,j}\frac{\partial L}{\partial Z_{i,j}} \cdot X_{i+u,j+v}$$

코드에서는 u·v·i·j 네 겹 for-loop로 누적하면 된다.

(c) 입력 X

입력 한 픽셀은 출력 여러 위치에 겹쳐 쓰인다. 모든 겹치는 곳의 dZ 값과 대응 W를 더해 주면 입력 기울기가 된다.

조건: $0 \le p - i < K$, $0 \le q - j < K$

구현은 W를 "뒤집어서" 같은 2중 합으로 계산한다.

A.4.2 다중 채널 · 다중 필터 일반화

- 입력채널: C_{in}
- 출력 채널(필터 개수): C_{out}

순전파는

$$Z_{n,o,i,j} = \sum_{c=0}^{C_{in}-1}\sum_{u,v} W_{o,c,u,v} X_{n,c,i+u,j+v} + b_0$$

역전파 결과:

-편향

$$\partial L/\partial b_0 = \sum_{n,i,j} dZ_{n,o,i,j}$$

-가중치

$$\frac{\partial L}{\partial W_{o,c,u,v}} = \sum_{n,i,j} dZ_{n,o,i,j} X_{n,c,i+u,j+v}$$

-입력

$$dX_{n,c,p,q} = \sum_{o,i,j} dZ_{n,o,i,j} W_{o,c,p-i,q-j}$$

코드 구조는 5중(또는 6중) for-loop가 필요하지만, 인덱스가 늘어났을 뿐 계산 원리는 1채널과 동일하다.

A.5 RMSprop 업데이트 공식

1차 모멘텀 대신 제곱 기울기의 지수 평균 r_t을 누적한다.

$$r_t = \rho r_{t-1} + (1-\rho) g_t^2$$

지수 평균된 r_t로 분모를 스케일링하여 파라미터를 갱신한다.

$$\theta_t = \theta_{t-1} - \eta \frac{g_t}{\sqrt{r_t} + \varepsilon}$$

ρ는 보통 0.9, ε은 1.0×10^{-8}을 쓴다.

A.6 Adam 업데이트 공식

1차 모멘텀(평균) m_t와 2차 모멘텀(분산) v_t를 지수 평균으로 누적한다.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t,$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

편향 보정 후 파라미터를 갱신한다.

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t},$$

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}.$$

β_1은 보통 0.9, β_2는 0.999, ε은 $1.0e^{-8}$을 쓴다.

부록 B - 핵심코드

핵심 로직이 한눈에 보이도록 각 파일에서 가장 중요한 부분만 발췌했습니다. 전체 파일은 GitHub/ZIP 첨부본에서 확인할 수 있습니다.

B.1 utils/util_excel.py - 데이터 로더

구분	내용
함수	load_excel_dataset(path, sheet_name='Data')
핵심 흐름	① 엑셀 열기 ② 샘플 96개 순회 ③ 픽셀 6×6, 라벨 3칸 읽기 ④ X를 (96, 1, 6, 6) float32, y를 (96,) int64 로 리턴
주의	열 인덱스 오프셋: 첫 샘플 L열 (열 번호 12) → col_start = 12 + i*6

표 6. util_excel.py - 6 × 6 손글씨 데이터 로더의 핵심 흐름

B.2 nn/conv.py — 합성곱 · 맥스풀 레이어

1) Conv2D

-멤버

W(가중치), b(편향), dW, db, 내부 저장용 x

-forward(x)

입력 배치 X 를 5중 for-loop(배치→출력채널→입력채널→공간 i→공간 j)로 순회하며 창(window)·가중치 곱의 합을 계산, bias 추가 결과 shape = (N, C_out, H_out, W_out)

-backward(d_out)

위와 동일한 인덱스 체계를 역으로 사용해

① 편향 기울기 db : d_out 전부 더하기

② 가중치 기울기 dW : d_out × 입력창 누적

③ 입력 기울기 dx : d_out × W 누적

최종 dx 를 반환해 전 레이어로 넘겨 준다

2) MaxPool2D (kernel=2, stride=2)

-forward

2×2 창에서 최댓값과 그 좌표(딕셔너리 max_idx)를 저장

-backward

위에서 받은 d_out을 저장 좌표에만 대입 → dx 완성

B.3 nn/activation.py - 활성화 · 손실

함수	forward 출력	backward 인자	동작
relu(x)	(y, mask)	(d_out, mask)	
sigmoid(x)	(s,s)	(d_out, s)	d_out*s*(1-s)
softmax_cross_entropy (logits, labels)	(loss, d_logits)	-	안정화 (shift) softmax, CE 손실, 로짓 기울기 계산

표 7. activation.py - Sigmoid·ReLU·Softmax-CE 함수의 입출력과 동작

B.4 nn/optimizer.py — 최적화

클래스	내부 상태	step() 동작
SGD	lr	p -= lr * grad
Momentum	v(속도)	v = gamma*v + lr*grad; p -= v
NAG	v	예측 위치에서 grad 측정 후 동일 방식
Adagrad	r(누적 제곱)	p -= lr*grad / (sqrt(r)+eps)
RMSprop	r(지수 평균)	위와 유사, ρ 사용
Adam	m,v,t	편향 보정 후 p -= lr * m̂ / (√v̂ + eps)

표 8. optimizer.py - 구현된 여섯 가지 최적화 알고리즘의 내부 상태와 step() 동작

모델	순전파 단계	파라미터 등록(optimizer용)	해당 실험명
SimpleCNN	Conv1 → Sigmoid → MaxPool → Conv2 → Sigmoid → Flatten	[(c1.W,c1.dW),(c1.b,c1.db),(c2.W,c2.dW),(c2.b,c2.db)]	S-SGD, S-RMSprop, S-Adam
EnhancedCNN	Conv1 → ReLU → Conv2 → ReLU → Conv3 → ReLU → Flatten	[(c1.W,c1.dW),(c1.b,c1.db),(c2.W,c2.dW),(c2.b,c2.db),(c3.W,c3.dW),(c3.b,c3.db)]	E-RMSprop, E-Adam

표 9. models/model.py - 두 가지 CNN 구조와 옵티마이저용 파라미터 등록 항목

B.5 models/model.py — 두 가지 CNN

표 9는 두 가지 CNN 구조와 옵티마이저용 파라미터 등록 방식을 한눈에 보여 준다. 먼저 SimpleCNN은 “Conv1 → Sigmoid → MaxPool → Conv2 → Sigmoid → Flatten” 순으로 이루어진다. 두 합성곱 층의 가중치·편향을 차례로 모아 [(c1.W, c1.dW), (c1.b, c1.db), (c2.W, c2.dW), (c2.b, c2.db)] 네 쌍을 옵티마이저에 넘긴다. 이 동일한 구조에 옵티마이저만 달리 적용해 S-SGD, S-RMSprop, S-Adam 세 가지 실험을 수행하였다.

다음으로 EnhancedCNN은 맥스풀을 완전히 제거하고 “Conv1 → ReLU → Conv2 → ReLU → Conv3 → ReLU → Flatten”으로 구성된다. 세 합성곱 층이므로 파라미터 등록 항목도 늘어나 [(c1.W, c1.dW), (c1.b, c1.db), (c2.W, c2.dW), (c2.b, c2.db), (c3.W, c3.dW), (c3.b, c3.db)] 여섯 쌍을 전달한다. 이 구조에는 RMSprop과 Adam을 적용해 E-RMSprop과 E-Adam 두 실험을 진행하였다. 표 9의 ‘해당 실험명’ 열에 적힌 약어(S-, E-)는 본문 전반에서 사용되는 모델 표기와 정확히 대응하므로, 구조와 실험 조건을 빠르게 대조할 수 있다.

B.6 train.py — 학습 스크립트

실험용 스크립트는 단일 조합 확인용과 일괄 실행용 두 형태로 구성되어 있다. train.py, train_rmsprop.py, train_adam.py처럼 접미사가 없는 파일들은 한 모델-옵티마이저 조합을 빠르게 검증하기 위한 것이다. 스크립트가 시작되면 util_excel.py에서 6 × 6 손글씨 데이터를 읽어 $(N,1,6,6)$ 입력 배열과 $(N,)$ 정수 라벨을 얻는다. 이어서 난수 시드를 고정한 뒤 인덱스를 섞어 학습과 검증을 8 : 2로 분할하고, 모델(S-계열은 SimpleCNN, E-계열은 EnhancedCNN)을 생성한다. collect_params(model) 함수가 합성곱 레이어의 가중치·편향과 대응 기울기를 자동 수집해 옵티마이저(SGD, RMSprop, Adam)에 전달하며, 300 epoch 동안 순전파·손실 계산·역전파·파라미터 갱신을 반복한다. 열 에폭마다 손실이 출력되고, 마지막 단계에서 학습·검증 정확도가 계산되어 화면에 표시된다. 이러한 구조 덕분에 학습률이나 모멘텀 계수를 수정해도 파일 하나만 실행하면 결과를 즉시 확인할 수 있다.

train_all.py는 보고서에 제시된 다섯 모델(S-SGD, S-RMSprop, S-Adam, E-RMSprop, E-Adam)을 시드 0부터 9까지 열 번씩 자동으로 학습해 결과를 한 번에 집계하도록 작성되었다. 스크립트 내부의 실험 목록을 차례로 순회하면서 run_once()가 각 조합을 학습하고 정확도를 반환하면, 메인 루프가 그것을 배열에 저장한다. 열 번의 반복이 끝나면 numpy를 이용해 평균과 표준편차를 계산하고, pandas 데이터프레임으로 Run 1 ~ Run 10, Mean, Std를 포함한 표를 터미널에 출력한다. 이 표가 곧 본문에 사용된 표 5와 동일한 값이며, loss_record 변수에는 에폭별 손실 곡선이 보존되어 필요하면 그래프 작성이나 로그 파일 저장에 활용할 수 있다.

B.7 파일 간 흐름 요약

util_excel.py가 엑셀 원본을 읽어 $(N,1,6,6)$ 배치 입력과 $(N,)$ 라벨을 반환하면, train_all.py는 시드를 고정해 model.forward()로 순전파를 실행한다. models/model.py 안에서는 합성곱→활성화 → (선택적) 맥스풀 계층이 순차적으로 계산되어 로짓이 만들어지고, Softmax - Cross-Entropy 모듈이 이를 확률과 손실, 그리고 로짓 기울기 d_logits로 변환한다. 같은 모델 객체에서 backward()를 호출하면 각 Conv 레이어의 dW, db가 채워지고, 이후 optimizer.step() — SGD, RMSprop, Adam 중 해당 조합이 이 기울

기를 사용해 파라미터를 업데이트한다. 이 과정을 300 epoch 반복한 뒤 정확도를 계산하고, 시드 0 부터 9까지 10 회 반복 결과의 평균과 표준편차를 구해 보고서 표 5에 제시한 성능 지표로 출력한다.