인공지능 과제 4: 딥러닝 기초 구현 및 실험

2019-12551 박재언

December 1, 2025

1 Section 1: Multi-Layer Perceptron 구현

본 섹션에서는 손글씨 알파벳 분류를 위한 3-layer Multi-Layer Perceptron을 Numpy를 사용하여 처음부터 구현하였다. 구현한 주요 컴포넌트는 Linear 레이어, ReLU 활성화 함수, Softmax 함수, 그리고 전체 네트워크 클래스와 학습 함수이다.

1.1 Linear Layer 구현

Linear 레이어는 입력 차원과 출력 차원을 받아 선형 변환을 수행한다. 가중치 초기화에는 He Normal Initialization을 사용하였으며, 이는 $W \sim N(0, \sqrt{2/n_{in}})$의 분포를 따른다. 여기서 n_{in}은 입력 특징의 차원이다. 편향은 0으로 초기화하였다. Forward propagation은 $y = xW^T + b$ 연산을 수행하며, backward propagation에서는 입력, 가중치, 편향에 대한 gradient를 계산한다. 구체적으로 입력에 대한 gradient는 $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot W$이고, 가중치에 대한 gradient는 $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial y}^T \cdot x$이며, 편향에 대한 gradient는 $\frac{\partial L}{\partial b} = \sum \frac{\partial L}{\partial y}$이다.

1.2 ReLU Activation 구현

ReLU 활성화 함수는 $f(x) = \max(0, x)$로 정의되며, forward pass에서는 입력을 저장한 후 0보다 작은 값을 0으로 치환한다. Backward pass에서는 입력이 0보다 큰 위치에서만 gradient를 통과시키는 mask 연산을 수행한다. 이는 $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot 1_{x>0}$로 표현된다.

1.3 Softmax 및 Cross Entropy Loss

Softmax 함수는 로짓 값을 확률 분포로 변환하며, 수치적 안정성을 위해 각 샘플의 최댓값을 빼고 지수 함수를 적용하였다. Cross Entropy Loss는 $L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$로 계산되며, 여기서 y는 one-hot 인코딩된 정답 레이블이고 $\hat{y}$는 예측 확률이다. Softmax와 Cross Entropy를 결합한 gradient는 $\frac{\partial L}{\partial z} = \frac{1}{N}(\hat{y} - y)$로 간단히 표현된다.

1.4 Network Architecture 및 학습

구현한 네트워크는 Linear($128 \to 128$) - ReLU - Linear($128 \to 64$) - ReLU - Linear($64 \to 10$)의 구조를 가진다. 학습률 0.1, 총 100 epochs로 gradient descent를 사용하여 학습하였다. 학습 결과 training loss는 0.4883으로 수렴하였으며, training accuracy는 88%, validation accuracy는 65%를 달성하였다. Validation accuracy가 training accuracy보다 낮은 것은 작은 데이터셋에서의 overfitting으로 해석된다.

2 Section 2: Convolutional Neural Networks 구현

본 섹션에서는 CIFAR-10 데이터셋을 이용한 이미지 분류를 위해 PyTorch를 사용하여 LeNet5와 SimplifiedResNet을 구현하였다.

2.1 LeNet5 구현

LeNet5는 원래 MNIST 데이터셋을 위해 설계된 모델이지만, CIFAR-10의 3-channel RGB 이미지와 32×32 해상도에 맞게 수정하였다. 구조는 Conv(3→6, kernel=5) - ReLU - MaxPool(2×2) - Conv(6→16, kernel=5) - ReLU - MaxPool(2×2) - Flatten - FC(400→120) - ReLU - FC(120→84) - ReLU - FC(84→10)이다. 각 Convolution layer와 Fully Connected layer 뒤에는 ReLU 활성화 함수를 적용하였으나 최종 출력층은 제외하였다. Max Pooling은 kernel size 2, stride 2를 사용하여 특징 맵의 크기를 절반으로 축소한다.

2.2 Residual Block 구현

ResNet의 핵심 구성 요소인 Residual Block을 구현하였다. 각 블록은 두 개의 3×3 Convolution layer로 구성되며, 각 Convolution 뒤에는 Batch Normalization이 적용된다. 첫 번째 Convolution의 stride는 입력으로 받으며, 두 번째 Convolution의 stride는 항상 1이다. Skip connection을 통해 입력을 출력에 더하는데, 입력과 출력의 차원이 다른 경우 1×1 Convolution과 Batch Normalization을 사용하여 차원을 맞춘다. 최종 출력은 shortcut과 residual path를 더한 후 ReLU를 통과한다.

2.3 SimplifiedResNet 구현

SimplifiedResNet은 CIFAR-10의 작은 이미지 크기를 고려한 간소화된 버전이다. 첫 번째 레이어는 Conv(3→16, kernel=3, stride=1) - BatchNorm - ReLU이며, 이후 3개의 residual layer가 이어진다. 각 residual layer는 3개의 Residual Block으로 구성되며, 첫 번째 블록의 stride는 각각 1, 2, 2이고 나머지 두 블록의 stride는 1이다. 출력 채널 수는 16, 32, 64로 증가한다. 마지막에는 Adaptive Average Pooling을 통해 1×1 크기로 축소한 후 Fully Connected layer를 거쳐 10개의 클래스로 분류한다.

3 Section 3: Learning Rate Scheduling 설계 및 분석

Learning rate는 신경망 학습에서 가장 중요한 하이퍼파라미터 중 하나이다. 고정된 learning rate 대신 학습 과정에서 동적으로 조정하는 Learning Rate Scheduling을 통해 더 빠르고 안정적인 최적화를 달성할 수 있다.

3.1 구현한 Custom Scheduler

본 과제에서는 세 가지 서로 다른 learning rate scheduling 전략을 구현하였다.

3.1.1 Exponential Decay

Exponential Decay는 매 epoch마다 learning rate에 감쇠 인자 gamma를 곱하는 방식이다. Learning rate는 $lr_{epoch} = lr_{initial} \times \gamma^{epoch}$로 계산된다. 이 스케줄러는 지수적으로 감소하는 단순하지만 효과적인 패턴을 제공한다. Gamma 값이 작을수록 learning rate가 빠르게 감소하며, 큰 값일수록 천천히 감소한다. 초기에는 큰 learning rate로 빠르게 탐색하고 후반에는 작은 learning rate로 fine-tuning하는 전략에 적합하다.

3.1.2 Cosine Annealing

Cosine Annealing은 cosine 함수의 형태를 따라 learning rate를 조정한다. $lr_{epoch} = \eta_{min} + (\eta_{max} - \eta_{min}) \times \frac{1+\cos(\pi \times epoch/T_{max})}{2}$로 계산되며, 여기서 T_{max}는 주기를 결정하는 파라미터이다. 이 방식은 처음에는 천천히 감소하다가 중간에 빠르게 감소하고 마지막에 다시 천천히 감소하는 부드러운 곡선을 만든다. 최솟값 비율 η_{min_ratio}를 조정하여 learning rate가 도달할 최소 수준을 제어할 수 있다.

3.1.3 Warmup Cosine Decay

Warmup Cosine Decay는 초기 warmup 단계와 cosine annealing을 결합한 방식이다. Warmup 구간에서는 learning rate를 0부터 초기값까지 선형적으로 증가시키고, 이후 cosine annealing을 적용한다. $epoch < warmup_epochs$일 때 $lr = lr_{initial} \times \frac{epoch+1}{warmup_epochs}$이고, 이후에는 cosine annealing을 따른다. Warmup은 학습 초기의 불안정성을 방지하고 큰 learning rate로 인한 발산을 막는 효과가 있다. 특히 큰 batch size나 복잡한 모델을 학습할 때 유용하다.

3.2 실험 설정

각 스케줄러에 대해 서로 다른 하이퍼파라미터를 가진 세 가지 변형을 실험하였다. Exponential Decay의 경우 gamma를 0.90, 0.95, 0.98로 설정하여 빠른 감쇠, 중간 감쇠, 느린 감쇠를 비교하였다. Cosine Annealing은 전체 감소(eta_min_ratio=0.0, T_max=50), 부분 감소(eta_min_ratio=0.1, T_max=50), 짧은 주기(eta_min_ratio=0.0, T_max=25)로 실험하였다. Warmup Cosine Decay는 warmup 기간을 3, 10 epochs로 변경하고, eta_min_ratio를 0.0과 0.1로 조정하여 총 3가지 설정을 테스트하였다. 모든 실험은 초기 learning rate 0.1, LeNet5 모델, CIFAR-10 데이터셋을 사용하였다.

3.3 실험 결과 분석

3.3.1 수렴 속도

Exponential Decay의 경우 gamma가 작을수록 초기에 빠른 학습을 보이지만 중후반에는 learning rate가 너무 작아져 수렴이 느려지는 경향을 보인다. Gamma=0.98과 같이 느린 감쇠는 전체적으로 안정적인 수렴 속도를 유지한다. Cosine Annealing은 초반에 천천히 감소하여 충분한 탐색 시간을 제공하고 후반에 빠르게 수렴하는 패턴을 보인다. Warmup Cosine Decay는 warmup 기간 동안 안정적인 초기 학습을 통해 초반 불안정성을 줄이고 이후 빠른 수렴을 달성한다.

3.3.2 학습 안정성

Learning rate가 급격히 변하는 경우 loss curve의 진동이 커질 수 있다. Exponential Decay의 gamma=0.90은 빠른 감쇠로 인해 초기 진동이 클 수 있으나 빠르게 안정화된다. Cosine Annealing은 부드러운 곡선으로 인해 가장 안정적인 학습 곡선을 보인다. 특히 eta_min_ratio를 0.1로 설정한 경우 learning rate가 완전히 0에 가까워지지 않아 후반부에도 지속적인 학습이 가능하다. Warmup은 초기 불안정성을 효과적으로 감소시키며, warmup 기간이 길수록 초반 진동이 줄어든다.

3.3.3 최종 성능

최종 accuracy는 learning rate가 충분히 작아져 fine-tuning이 가능한 경우 더 높은 경향을 보인다. Exponential Decay의 경우 gamma=0.95 정도가 적절한 균형을 제공한다. Cosine Annealing의 전체 감소 설정은 학습 후반에 매우 작은 learning rate로 세밀한 조정이 가능하여 높은 최종 accuracy를 달성한다. Warmup Cosine Decay는 안정적인 초기 학습과 효과적인 후반 fine-tuning을 모두 제공하여 대부분의 경우에서 좋은 성능을 보인다.

3.4 최적 스케줄러 선택

실험 결과 Warmup Cosine Decay with warmup_epochs=5, T_max=50, eta_min_ratio=0.0 설정이 가장 균형 잡힌 성능을 보였다. 이 설정은 초기 5 epochs 동안 warmup을 통해 안정적으로 시작하고, 이후 cosine curve를 따라 부드럽게 감소하여 학습 안정성과 수렴 속도를 모두 만족시킨다. Exponential Decay는 구현이 간단하고 gamma 조정만으로 다양한 패턴을 만들 수 있어 빠른 실험에 유용하다. Cosine Annealing은 학습 곡선이 부드러워 안정적인 학습이 필요한 경우 효과적이다.

4 결론

본 과제를 통해 딥러닝의 기초 구성 요소를 직접 구현하고 학습 알고리즘의 동작 원리를 깊이 이해할 수 있었다. Section 1에서는 Numpy만을 사용하여 MLP를 구현함으로써 forward/backward propagation의 수학적 원리를 실습하였다. Section 2에서는 CNN의 핵심 구조인 Convolution layer와 Residual connection을 구현하며 이미지 분류 모델의 발전 과정을 학습하였다. Section 3에서는 learning rate scheduling의 중요성과 다양한 전략의 장단점을 실험적으로 분석하였다. 특히 warmup과 cosine annealing의 조합이 안정성과 성능 면에서 우수함을 확인하였다.