# Deep Learning from Scratch

## 1장 신경망과 딥러닝 시작

인공지능 AI : 인간처럼 사고, 학습, 판단 가능한 시스템

기계학습 ML : 데이터 기반 예측 모델 구축, 지도 학습과 비지도 학습 개념 포함

딥러닝 DL : 다층 신경망 이용, 특징 자동 추출 및 표현 학습 가능

퍼셉트론 : 기본 인공 신경망 단위, 입력값 x와 가중치 w 내적 + 편향 b → 계단 함수로 출력 결정

수식 : $y = f(\sum w_i x_i + b)$, f : 활성화 함수

논리 연산 구현 : AND, OR 가능, XOR 불가

XOR 문제 : 선형 분리 불가, 단층 퍼셉트론 학습 불가 → 다층 퍼셉트론 필요

신경망 구성 요소 : 입력층, 은닉층, 출력층

활성화 함수 필요 : 단순 선형 조합으로 비선형 문제 해결 불가

활성화 함수 종류 : Sigmoid, tanh, ReLU

학습 과정 : 손실 함수 Loss Function 정의 → 경사하강법 Gradient Descent로 최적화

손실 함수 예 : 평균제곱오차 MSE, 교차엔트로피 Cross-Entropy

프로그래밍 환경 : Python + Numpy, 벡터/행렬 연산 기반

예제 : 단층 퍼셉트론 AND 구현, 가중치 초기화, 반복 학습 과정

순서 : 입력 벡터 x → 가중치 w, 편향 b → 출력 y 계산 → 오차 t-y → 가중치 갱신 반복

학습률 $\eta$ : 업데이트 크기 조절, 학습 안정성에 영향

데이터 전처리 필요성 : 입력 값 정규화, 학습 속도 향상

과적합/과소적합 문제 : 데이터 부족 또는 모델 복잡도 과다 → 일반화 성능 저하

다층 퍼셉트론 구조 필요 : 은닉층 추가 → 비선형 문제 해결 가능

장 마무리 : AI → ML → DL 흐름 이해, 퍼셉트론 정의와 한계 인지, 다층 신경망 필요성 숙지, Python 기반 구현 준비 완료

## 2장 퍼셉트론

퍼셉트론 학습 : 입력 → 가중치 내적 → 계단 함수 → 출력 → 오차 계산 → 가중치 갱신

가중치 갱신 수식 : $w_i \leftarrow w_i + \eta (t - y)x_i$, t : 정답

학습률 η : 한 단계 이동 크기 조절, 학습 속도 및 안정성에 영향

단층 퍼셉트론 한계 : XOR 문제, 선형 분리 불가 → 다층 퍼셉트론 필요

활성화 함수 발전 : 계단 함수 → Sigmoid

Sigmoid : 출력 0~1, 미분 가능, 확률 해석 가능, 포화 영역에서 기울기 소실 문제

tanh : 출력 -1~1, 평균 0 → 학습 안정성 향상

ReLU : 출력 0 이상 선형, 계산 효율 높음, 학습 빠름

Leaky ReLU : 음수 영역에도 작은 기울기 존재 → 죽은 뉴런 문제 완화

Python/Numpy 구현 : 입력 벡터 x, 가중치 w, 편향 b → 출력 y 계산 → 오차 t-y → 가중치 갱신 반복

학습 과정 시각화 : 손실 감소, 정확도 상승 확인

장 마무리 : 퍼셉트론 학습 알고리즘 이해, 가중치 갱신 방식 숙지, XOR 한계 확인, 활성화 함수 발전 필요성 인지, Python 구현 가능

# 3장 신경망

다층 퍼셉트론 MLP : 은닉층 추가 → 비선형 문제 해결 가능

구성 : 입력층 → 은닉층 → 출력층

각 층 : 선형 변환 + 활성화 함수 적용

활성화 함수 비교

Sigmoid : 출력 0~1, 미분 가능, 포화 영역에서 기울기 소실 문제

tanh : 출력 -1~1, 평균 0 → 학습 안정성 향상

ReLU : 음수 영역 0, 양수 선형 → 학습 효율 높음, 죽은 뉴런 문제 가능

순전파 forward propagation : 입력 → 은닉층 → 출력 계산 → 손실 계산

역전파 backpropagation : 연쇄법칙 chain rule 기반 → 기울기 계산 → 가중치 갱신

손실 함수

MSE : $L = 1/2 \sum (y - t)^2$

Cross-Entropy : $L = - \sum t_i \log y_i$

코드 구현 흐름 : forward → loss → backward → weight update 반복

Python/Numpy 기반 다층 신경망 구현 가능

장 마무리 : MLP 구조 이해, 순전파/역전파 개념 숙지, 활성화 함수 비교, Python 구현 가능

# 4장 손실 함수와 학습

손실 함수 Loss Function 심화

회귀 : MSE, MAE

분류 : Cross-Entropy, Softmax 적용

Softmax : 다중 클래스 확률 출력

수식 : y_i = e^{z_i} / ∑ e^{z_j}

Cross-Entropy : L = - ∑ t_i log y_i

경사하강법 Gradient Descent 심화

배치 Batch : 전체 데이터로 손실 계산 → 가중치 갱신, 안정적

확률적 Stochastic : 한 샘플씩 손실 계산 → 가중치 갱신, 진동 가능

미니배치 Mini-batch : 작은 배치 단위 → 속도와 안정성 균형, GPU 학습 최적화

가중치 초기화 중요 : 작은 무작위값 → 학습 안정화

Xavier, He 초기화 : 활성화 함수에 맞춰 분포 조정

최적화 기법

Momentum : 이전 기울기 활용, 관성 효과 → 학습 속도 향상

AdaGrad : 학습률 자동 조정, 희소 데이터 학습 유리

RMSProp : 지수이동평균 기반 학습률 조정 → 안정성 향상

Adam : Momentum + RMSProp 결합, 널리 사용

과적합 방지 : 정규화 L1, L2, Dropout → 학습 시 일부 뉴런 임의 제거 → 일반화 성능 향상

코드 흐름 : forward → loss 계산 → backward → optimizer 적용 → weight update 반복

학습 반복(epoch, batch 단위) → 손실 감소 → Accuracy 상승

장 마무리 : 손실 함수 종류와 의미 이해, 경사하강법 심화, 최적화 전략 적용 가능, 학습 안정성 확보

# 5장 활성화 함수

신경망에서 비선형성 도입 필수

단순 선형 변환만으로는 XOR 등 비선형 문제 해결 불가 → 활성화 함수 필요

주요 활성화 함수

Sigmoid : 수식 σ(x) = 1 / (1 + e^{-x})

출력 범위 0~1, 확률 해석 가능

장점 : 미분 가능, 확률적 출력

단점 : 출력 포화 → 기울기 소실(vanishing gradient) 문제, 학습 느림

tanh : 수식 tanh(x) = (e^x - e^{-x}) / (e^x + e^{-x})

출력 범위 -1~1, 평균 0 → 학습 안정성 향상

장점 : Sigmoid보다 중앙 집중, 학습 빠름

단점 : 큰 입력값에서 포화 발생 → 기울기 소실 가능

ReLU : 수식 f(x) = max(0, x)

장점 : 단순, 계산 효율 높음, 학습 빠름

단점 : 음수 영역 0 → 죽은 뉴런 문제 발생 가능

Leaky ReLU : 수식 f(x) = max(0.01x, x)

음수 영역에도 작은 기울기 존재 → 죽은 뉴런 문제 완화

Softmax : 수식 y_i = e^{z_i} / ∑ e^{z_j}

다중 클래스 확률 출력, Cross-Entropy와 함께 사용

출력층에서 주로 사용, 전체 합 1 → 확률 해석 가능

학습과 활성화 함수 관계

활성화 함수 선택 → 학습 속도 및 안정성에 직접 영향

은닉층 ReLU, 출력층 Sigmoid/Softmax 조합 일반적

순전파 : 활성화 함수 적용 → 출력 계산

역전파 : 활성화 함수 미분값 활용 → 기울기 전달

코드 흐름

입력 → Affine Layer → Activation Layer → 출력 → Loss

Loss → Activation Backward → Affine Backward → Weight Update

장 마무리 : 비선형 문제 해결, 활성화 함수 종류 및 특징 숙지, 순전파/역전파 적용 이해

# 6장 학습 알고리즘과 최적화

경사하강법 Gradient Descent 심화

- 배치 Batch GD : 전체 데이터로 손실 계산 → 가중치 갱신, 안정적
- 확률적 Stochastic GD : 한 샘플씩 손실 계산 → 가중치 갱신, 진동 가능
- 미니배치 Mini-batch GD : 작은 배치 단위 → 속도와 안정성 균형, GPU 학습 최적화

학습률 Learning Rate

- 너무 크면 발산, 너무 작으면 학습 느림
- 학습률 스케줄링: 학습 진행에 따라 감소

최적화 기법

Momentum : 이전 기울기 활용 → 관성 효과, 학습 속도 향상

AdaGrad : 학습률 자동 조정, 희소 데이터 학습 유리

RMSProp : 지수이동평균 기반 학습률 조정 → 안정성 향상

Adam : Momentum + RMSProp 결합, 가장 널리 사용

가중치 초기화

작은 무작위값 → 학습 시작 안정화

Xavier, He 초기화 → 활성화 함수에 맞춰 분포 조정

과적합 방지

정규화 L1, L2

Dropout : 학습 시 일부 뉴런 임의 제거 → 일반화 성능 향상

코드 흐름

forward → loss 계산 → backward → optimizer 적용 → weight update 반복

학습 반복(epoch, batch 단위) → 손실 감소 → Accuracy 상승

장 마무리 : 학습 알고리즘 심화, 최적화 전략 이해, 과적합 방지 방법 숙지, 학습 안정성 확보

# 7장 신경망 구현

Numpy 기반 신경망 구현

레이어 단위 구조화 : Affine Layer, Activation Layer 분리

## 레이어 구조

Affine Layer : 선형 변환 y = Wx + b

- Forward : 입력 → 가중치 곱 + 편향 → 출력
- Backward : 손실에 대한 기울기 계산 → 가중치/편향 업데이트

Activation Layer : ReLU, Sigmoid 등

- Forward : 입력 → 활성화 함수 적용 → 출력
- Backward : 출력에 대한 기울기 전달 → Affine Layer 연결

Softmax + Cross-Entropy Layer

- Forward : Softmax → Cross-Entropy 손실 계산
- Backward : 손실 미분값 계산 → 출력층 기울기 전달

## 학습 루프

데이터 로드 → 신경망 초기화 → 학습 반복

배치 처리, 손실 기록, 정확도 계산

모듈화 : 레이어 클래스 재사용, 확장성 확보

코드 흐름

for epoch :

for batch in data :

forward pass

loss 계산

backward pass

optimizer 적용 → 가중치 갱신

학습 완료 후 평가

장 마무리 : Numpy 신경망 구현, Layer 단위 구조화, forward/backward 구현, 학습 루프 완성

# 8장 MNIST 실습과 정리

MNIST 데이터셋 : 손글씨 숫자 이미지, 28x28 픽셀 → 784차원 입력 벡터

데이터 전처리 : 정규화 0~1, 학습/검증 데이터 분리

### 신경망 구성

입력층 : 784

은닉층 : 50~100, ReLU

출력층 : 10, Softmax

손실 함수 : Cross-Entropy

### 학습

미니배치 SGD

Epoch 반복 → 손실 감소, 정확도 상승

학습률 조정 : 0.1~0.01

### 성능 평가

학습 손실, 검증 손실 기록

학습 곡선 시각화 → 과적합 여부 확인

Accuracy : 학습 정확도, 검증 정확도 비교

### 실습 결과

단층 퍼셉트론 → 다층 신경망 → MNIST 학습 과정 완성

순전파, 역전파, 활성화 함수, 손실 함수, 최적화 기법 전 과정 적용

장 마무리 : MNIST 실습 완료, 신경망 학습 과정 완전 이해, 구현 능력 확보, 다음 단계 준비 완료

## 1장 신경망 심화

다층 퍼셉트론 MLP : 은닉층 다중 구성, 비선형 문제 해결

입력층, 다수 은닉층, 출력층 구조

각 층 : 선형 변환 + 활성화 함수 적용

순전파 forward propagation : 입력 → 은닉층 → 출력 계산 → 손실 계산

역전파 backpropagation : 연쇄법칙 chain rule 기반 → 기울기 계산 → 가중치 갱신

손실 함수 Loss Function

MSE : 회귀용, L = 1/2 ∑ (y - t)^2

Cross-Entropy : 분류용, L = - ∑ t_i log y_i

Softmax : 다중 클래스 확률 출력, Cross-Entropy와 함께 사용

활성화 함수 Activation Function

Sigmoid : 출력 0~1, 미분 가능, 포화 영역 기울기 소실

tanh : 출력 -1~1, 평균 0 → 학습 안정성 향상

ReLU : 음수 0, 양수 선형 → 학습 효율 높음

Leaky ReLU : 음수 영역 작은 기울기 존재 → 죽은 뉴런 문제 완화

Python/Numpy 구현 : 입력 → Affine Layer → Activation Layer → 출력 → Loss

Loss → Backward → Weight Update 반복

장 마무리 : MLP 구조 심화, 순전파/역전파 숙지, 활성화 함수 비교, Python 구현 가능

# 2장 학습 심화

경사하강법 Gradient Descent 심화

배치 Batch GD : 전체 데이터 → 가중치 갱신, 안정적

확률적 Stochastic GD : 한 샘플 → 가중치 갱신, 진동 가능

미니배치 Mini-batch GD : 작은 배치 → 속도와 안정성 균형, GPU 학습 최적화

학습률 Learning Rate

크기 중요 : 너무 크면 발산, 너무 작으면 학습 느림

학습률 스케줄링 : 진행에 따라 감소

최적화 Optimizer

Momentum : 이전 기울기 활용 → 관성 효과, 속도 향상

AdaGrad : 학습률 자동 조정, 희소 데이터 유리

RMSProp : 지수이동평균 기반 학습률 → 안정성 향상

Adam : Momentum + RMSProp 결합, 널리 사용

가중치 초기화 : 작은 무작위값, Xavier, He 초기화 → 활성화 함수 최적화

과적합 방지 : 정규화 L1/L2, Dropout → 일반화 성능 향상

Python/Numpy 학습 루프

Forward → Loss 계산 → Backward → Optimizer 적용 → Weight Update 반복

Epoch 단위 반복 → 손실 감소, Accuracy 상승

장 마무리 : 학습 알고리즘 심화, 최적화 전략 숙지, 과적합 방지, 안정적 학습 가능

## 3장 신경망 구조와 레이어

레이어 단위 신경망 구현

Affine Layer : 선형 변환 y = Wx + b

Activation Layer : ReLU, Sigmoid 등

Softmax + Cross-Entropy Layer : 다중 클래스 분류

Forward : 입력 → Affine → Activation → 출력 → Loss

Backward : Loss → Activation Backward → Affine Backward → Weight Update

모듈화 Layer 클래스 : 재사용, 확장성 확보

학습 루프 : 데이터 로드 → 신경망 초기화 → Forward/Backward → Optimizer 적용 → Epoch 반복

손실 기록, 정확도 계산, 학습 곡선 시각화

장 마무리 : 신경망 구조 이해, 레이어 단위 구현, 순전파/역전파 적용, Python 구현 가능

## 4장 MNIST와 데이터 처리

MNIST 데이터셋 : 손글씨 숫자 이미지, 28x28 픽셀 → 784차원 입력

데이터 전처리 : 정규화 0~1, 학습/검증 분리

배치 처리 : Mini-batch SGD 적용

신경망 구성 : 입력 784, 은닉층 50~100 ReLU, 출력 10 Softmax

손실 함수 : Cross-Entropy

학습 : Epoch 반복, 학습률 0.1~0.01, 손실 감소, Accuracy 상승

평가 : 학습 손실, 검증 손실 기록, 학습 곡선 시각화 → 과적합 여부 확인

실습 결과 : 단층 → 다층 신경망 학습 완료, 순전파/역전파/손실/활성화/최적화 전체 과정 적용

장 마무리 : MNIST 실습 완료, 신경망 학습 과정 완전 이해, Python 구현 능력 확보

# 5장 활성화 함수와 정규화

신경망 학습에서 비선형성 확보 필수, 단순 선형 변환만으로는 XOR 등 문제 해결 불가

활성화 함수 Activation Function : 은닉층 및 출력층 비선형성 도입, 기울기 전파 영향

주요 활성화 함수

Sigmoid : σ(x) = 1 / (1 + e^{-x})

- 출력 범위 0~1 → 확률 해석 가능
- 장점 : 미분 가능, 확률적 출력
- 단점 : 입력값 크면 포화 → 기울기 소실(vanishing gradient) 문제, 학습 느림

tanh : tanh(x) = (e^x - e^{-x}) / (e^x + e^{-x})

- 출력 범위 -1~1, 평균 0 → 학습 안정성 향상
- 장점 : Sigmoid보다 중앙 집중, 학습 빠름
- 단점 : 큰 입력값 포화 → 기울기 소실 가능

ReLU : f(x) = max(0, x)

- 장점 : 단순, 계산 효율 높음, 학습 빠름
- 단점 : 음수 영역 0 → 죽은 뉴런 문제 발생 가능

Leaky ReLU : f(x) = max(0.01x, x)

- 음수 영역에도 작은 기울기 존재 → 죽은 뉴런 문제 완화

Softmax : y_i = e^{z_i} / ∑ e^{z_j}

- 다중 클래스 확률 출력, Cross-Entropy와 결합
- 출력층에서 주로 사용, 전체 합 1 → 확률 해석 가능

정규화 Batch Normalization

- 각 미니배치 입력 정규화 → 평균 0, 분산 1
- 학습 안정성, 속도 향상, 과적합 감소

- 순전파 : 입력 → 정규화 → scale & shift → 다음 레이어
- 역전파 : 기울기 전파 → 가중치 업데이트

Python 구현 흐름 :

- Forward : 입력 → Affine → Activation → BatchNorm → 출력
- Loss 계산
- Backward : Loss → Activation Backward → BatchNorm Backward → Affine Backward → Weight Update

장 마무리 : 활성화 함수 비교, BatchNorm 이해, 순전파/역전파 적용, 학습 안정성 확보

# 6장 학습 알고리즘과 최적화

경사하강법 Gradient Descent : 손실 최소화 핵심

- Batch GD : 전체 데이터 → 가중치 갱신, 안정적
- Stochastic GD : 한 샘플 → 가중치 갱신, 진동 가능
- Mini-batch GD : 작은 배치 → 속도와 안정성 균형, GPU 최적화

학습률 Learning Rate

- 너무 크면 발산, 너무 작으면 학습 느림
- 학습률 스케줄링 : 진행에 따라 감소, 최적화

최적화 Optimizer

- Momentum : 이전 기울기 활용 → 관성 효과, 속도 향상
- AdaGrad : 학습률 자동 조정, 희소 데이터 유리
- RMSProp : 지수이동평균 기반 학습률 → 안정성 향상
- Adam : Momentum + RMSProp 결합, 널리 사용, 대부분 실무 표준

가중치 초기화

- 작은 무작위값 → 학습 시작 안정화
- Xavier : Sigmoid, tanh 적합
- He : ReLU 계열 적합

과적합 방지

- 정규화 : L1, L2
- Dropout : 학습 시 일부 뉴런 임의 제거 → 일반화 성능 향상

Python 학습 루프

- Forward → Loss 계산 → Backward → Optimizer 적용 → Weight Update 반복
- Epoch 단위 반복 → 손실 감소, Accuracy 상승
- 배치 단위 학습 → GPU 활용 가능, 학습 속도 향상

장 마무리 : 학습 알고리즘 심화, 최적화 기법 이해, 과적합 방지 전략 숙지, 학습 안정성 확보

# 7장 신경망 구현과 모듈화

Layer 단위 신경망 구현 : Affine Layer, Activation Layer, BatchNorm, Softmax+CrossEntropy

Forward : 입력 → 각 Layer → 출력 → Loss

Backward : Loss → 각 Layer Backward → Weight Update

Affine Layer : y = Wx + b

- Forward : 입력 → 가중치 곱 + 편향 → 출력
- Backward : Loss → 기울기 계산 → 가중치/편향 갱신

Activation Layer : ReLU, Sigmoid, Leaky ReLU 등

- Forward : 입력 → 활성화 적용 → 출력
- Backward : 출력 기울기 전달 → Affine Layer 연결

Softmax + Cross-Entropy Layer

- Forward : Softmax → Cross-Entropy 계산
- Backward : Loss 미분값 출력층 전달

BatchNorm Layer

- Forward : 입력 정규화 → scale & shift → 출력
- Backward : Loss 기울기 전파 → 가중치/편향 갱신

학습 루프

- 데이터 로드 → 신경망 초기화 → Forward/Backward → Optimizer 적용 → Epoch 반복

- 손실 기록, 정확도 계산, 학습 곡선 시각화

장 마무리 : Numpy 기반 Layer 단위 신경망 구현, 모듈화, 순전파/역전파 적용, 학습 루프 완성, Python 구현 능력 확보

# 8장 MNIST 심화 실습

MNIST 데이터셋 : 손글씨 숫자 이미지, 28x28 → 784차원 입력

데이터 전처리 : 정규화 0~1, 학습/검증 분리, Mini-batch 적용

신경망 구성

- 입력 784
- 은닉층 50~100 ReLU, BatchNorm 적용
- 출력 10 Softmax
- 손실 : Cross-Entropy

학습

- Mini-batch SGD + Adam
- Epoch 반복 → 손실 감소, Accuracy 상승
- 학습률 스케줄링 적용 → 안정적 학습

평가

- 학습 손실, 검증 손실 기록
- Accuracy : 학습 정확도, 검증 정확도
- 학습 곡선 시각화 → 과적합 여부 확인

실습 결과

- 단층 → 다층 → BatchNorm → 최적화 기법 적용 완료
- 순전파, 역전파, 활성화 함수, 손실 함수, 최적화 전체 과정 적용
- Python/Numpy 기반 신경망 구현 완전 이해, 학습 안정화, 실무 적용 가능

장 마무리 : MNIST 심화 실습 완료, 신경망 구현·학습 과정 완전 숙지, Python 기반 실무 능력 확보

# 제1고지 미분 자동 계산

### 1단계 상자로서의 변수

- Variable : 값(value), 기울기(grad), 부모 함수 참조(creator) 저장
- 계산 과정 추적 → 순전파 Forward 및 역전파 Backward 준비
- 핵심 : 변수 단위로 데이터와 미분 정보 관리, 계산 그래프 구성

### 2단계 변수를 낳는 함수

- Function 클래스 : Variable 입력 → Variable 출력 생성
- Forward : 입력 계산 → 출력 반환
- Backward : 출력 기울기 → 입력 기울기 계산
- 연산 단위를 함수 객체로 구조화 → 자동 미분 기반

### 3단계 함수 연결

- 계산 그래프 Graph 구성 : 각 Variable의 creator 연결 → 역전파 경로 확보
- 복합 함수 처리 가능 → 체인룰 적용
- 순전파 Forward : 입력 → 함수 → 출력
- 역전파 Backward : 출력.backward() → 그래프 따라 기울기 전파

### 4단계 수치 미분

- 작은 변화량 h → 기울기 근사
- 수치 미분 : (f(x+h) - f(x-h)) / 2h
- 검증용 → 자동 역전파 구현 정확도 확인

### 5단계 역전파 이론

- 체인룰 기반 → 복합 함수 미분 가능
- 입력 Variable → 출력 Variable 기울기 전파
- 기울기 계산 흐름 이해 → Backpropagation 핵심

### 6단계 수동 역전파

- 각 함수 별 Backward 구현

- 단순 함수 → 수동 기울기 계산
- 복합 함수 → 체인룰 적용, 기울기 전달

**7단계 역전파 자동화**

- Function 객체 재귀 호출 → 반복적 기울기 계산
- Variable.backward() 호출 시 그래프 따라 역전파
- 코드 예시 : `x = Variable(1.0); y = x ** 2 + 3 * x; y.backward()`

**8단계 재귀에서 반복문으로**

- 재귀 호출 → 스택 과부하 위험 → 반복문으로 변환
- 계산 그래프 후위 순회 → 안정적 Backward 수행
- 메모리 효율 개선

**9단계 함수를 더 편리하게**

- Wrapper 함수 : Variable 자동 생성, 연산 간소화
- 여러 입력 처리 및 복합 연산 자동화
- 코드 직관성 향상 → `f(x, y, z)`처럼 자연스러운 호출 가능

**10단계 테스트**

- 수치 미분과 역전파 결과 비교 → 정확도 검증
- 단순 함수, 합성 함수 테스트
- 구현 안정성 확보 → 이후 고차 미분, 신경망 기반 확장 준비

**제1고지 목표**

- Variable, Function 기반 계산 구조 완전 이해
- 수동 → 자동 역전파 구현 순서 숙지
- 계산 그래프, Forward/Backward 흐름 이해
- Python/Numpy 기반 구현 능력 확보

# 제2고지 자연스러운 코드로

**가변 길이 인수와 함수 구현**

- Forward 단계에서 *args 활용 → 입력 Variable 개수 동적 처리 가능

- 함수 호출 시 입력 갯수에 따라 자동 계산, 코드 간결화
- 역전파 Backward에서도 동일 방식 적용 → 다수 입력 Variable 기울기 자동 전파
- 코드 예시 : `forward(*inputs)` → 여러 입력 처리, `backward()` → 각 입력 기울기 계산

**같은 변수 반복 사용과 계산 그래프**

- 동일 Variable를 여러 연산에서 반복 사용 가능 → 그래프 공유
- 중간 결과 재사용 가능, 메모리 효율 증가
- 순환 구조 순전파, 역전파 시 그래프 순환 주의
- 메모리 관리 필요 → 순환 참조 문제 해결, Python GC 활용

**복잡한 계산 그래프 처리**

- Forward 시 각 연산을 Function 객체로 기록 → Backward에서 기울기 계산
- 그래프 크기 커도 재사용과 참조 관리로 효율적
- 메모리 절약 모드 구현 → 불필요한 중간 값 삭제, 학습 속도 유지

**연산자 오버로드 Operator Overloading**

- Variable 객체 간 자연스러운 연산 가능
- +, -, *, /, ** 등 사용 → 계산 그래프 자동 구성
- 예시 : `z = x * y + x ** 2` → Forward 기록, Backward 자동 연결
- 연산과 Backward 자동 연결 → 코드 직관적, 가독성 향상

**패키지화와 재사용성 강화**

- Function, Variable, Layer를 모듈화
- 반복적 사용 가능한 연산 단위 구성
- 복잡한 함수 미분도 자동 처리 가능 → Layer 단위 구현과 호환

**메모리 관리와 최적화**

- 순환 참조 방지, 필요 없는 중간 값 제거
- 학습 속도 유지, GPU/CPU 메모리 효율화
- Variable와 Function 참조 정리 → Garbage Collector 활용

**제2고지 목표**

- 계산 그래프 복잡도 증가에도 자연스러운 코드 유지

- 메모리 효율화 및 연산자 직관화
- 반복적 계산과 다양한 입력 처리 가능
- 확장 가능한 프레임워크 기반 구조 이해

# 제3고지 고차 미분 계산

**계산 그래프 시각화**

- Forward/Backward 구조 시각화 → 연산 흐름 이해
- 변수와 함수 간 의존 관계 확인 가능
- 복잡한 고차 미분 그래프도 분석 가능 → 디버깅과 최적화 용이

**테일러 급수 기반 미분**

- 함수 근사 → 작은 변화량 기반 고차 미분 계산
- 1차, 2차 이상 차수 미분 계산 가능
- 최적화 문제 적용 → 함수 최소/최대 계산

**고차 미분 High-order Derivative**

- 여러 차수 미분 계산 → 2차, 3차 이상
- sin, cos, exp, 복합 함수 적용 가능
- Forward → Variable → Function 연결 → 역전파 반복
- 자동 역전파로 고차 미분 구현, 중간 변수 자동 관리

**최적화 Optimization**

- 뉴턴 방법 Newton Method : 2차 미분 활용 → 수동 계산 및 자동 계산 가능
- 함수 최적화 : 최소값, 최대값 찾기
- Backward 자동화 → 고차 미분과 최적화 통합 가능

**코드 구현 흐름**

- Forward : 입력 Variable → Function 적용 → 출력
- Backward : 출력.backward() → 1차/2차 기울기 계산
- 계산 그래프 기반 자동화 → 고차 미분 연산 반복 가능
- 시각화 : 그래프 출력 → 함수 의존 관계 확인 및 분석

**실습 예제 포인트**

- sin(x), exp(x) 등 함수 고차 미분
- 뉴턴 방법 적용 → 최적화 문제 해결
- 계산 그래프 시각화 → 복잡한 미분 구조 이해
- Python/Numpy 기반 자동화 구현

**제3고지 목표**

- 고차 미분 구현 및 시각화
- 최적화 문제 해결 능력 확보
- 계산 그래프 구조 이해 및 자동화 능력
- 다양한 수학적 연산과 고차 미분 활용 가능

# 제4고지 신경망 만들기

**텐서 연산과 기초 함수**

- Tensor : 다차원 배열, Numpy 대체
- 형상 변환 : reshape, transpose 등 → 연산 전/후 차원 맞춤
- 합계 함수 : sum, mean → 배치 평균, 손실 계산
- 브로드캐스트 Broadcast : 차원 자동 맞춤, 연산 효율화
- 행렬 곱 MatMul : 선형 연산 핵심, Forward/Backward 구현

**선형 회귀와 신경망 구조**

- 선형 회귀 Linear : y = Wx + b, Forward/Backward 구현
- 다층 퍼셉트론 MLP : 은닉층 추가 → 비선형성 확보
- 활성화 함수 : ReLU, Sigmoid, tanh 적용
- Layer 단위 설계 : Layer → LayerList → 계층 구조 구성
- Optimizer : 매개변수 갱신, SGD/Adam 적용

**출력층과 손실 함수**

- Softmax : 다중 클래스 확률 출력
- Cross-Entropy : 분류 문제 손실 함수

- Softmax + Cross-Entropy : 안정적 확률 출력 및 기울기 계산

**데이터 처리와 학습 루프**

- Dataset 클래스 : 데이터 전처리, 학습/검증 분리
- DataLoader : 미니배치 생성, 반복 학습 지원
- Forward → Loss → Backward → Optimizer → Weight Update 반복
- Epoch 단위 학습 : 손실 감소, Accuracy 상승
- 학습 곡선 시각화 : 과적합 여부 확인

**MNIST 실습 포인트**

- 입력 : 28x28 이미지 → 784 차원 Vector 변환
- 은닉층 : 50~100 ReLU, BatchNorm 적용
- 출력층 : Softmax 10차원, 손실 : Cross-Entropy
- 학습 : Mini-batch SGD + Adam, Epoch 반복
- 평가 : 학습 손실, 검증 손실, Accuracy 기록, 시각화

**제4고지 목표**

- Tensor 기반 신경망 구현 능력 확보
- Layer 구조 이해, Optimizer 적용
- 실제 데이터 학습 및 평가 가능
- Python/Numpy 기반 실습 완료, 실전 신경망 구현 능력 확보

# 제5고지 DeZero의 도전

**GPU 지원**

- 연산 속도 향상, 대규모 데이터 처리 가능
- Variable/GPU Tensor 호환 → 연산 자동 전환
- Forward/Backward 연산 GPU 적용 → 학습 시간 단축
- 코드 예시 : `x.to_gpu()` → GPU 연산 적용

**모델 저장 및 읽어오기**

- 학습 완료 모델 저장 → `save_model()`

- 학습 재개 및 테스트 → `load_model()`
- Layer 단위, 전체 모델 구조 유지
- 학습 완료 후 평가, 추론에 활용 가능

**드롭아웃과 테스트 모드**

- Dropout : 학습 시 일부 뉴런 임의 제거 → 과적합 방지
- 테스트 모드 : 전체 뉴런 사용 → 정확한 예측 가능
- Forward 시 학습/테스트 모드 구분 → 안정적 성능 확보

**CNN 메커니즘**

- Convolution : Feature Map 생성, 커널 기반 연산
- Pooling : MaxPooling, AveragePooling → 차원 축소, 특징 강조
- conv2d 함수와 pooling 함수 구현
- 대표 모델 VGG16 구조 이해 : Convolution + Pooling + Fully Connected

**RNN과 LSTM**

- 시계열 데이터 처리 → 순환 구조 기반
- RNN : 이전 상태 반영, 연속 데이터 학습 가능
- LSTM : 장기 의존성 처리, Forget Gate, Input Gate, Output Gate 활용
- DataLoader와 연계, 시계열 미니배치 학습 가능

**실습 포인트**

- GPU 적용 → 연산 속도 향상, 대규모 학습 가능
- CNN/VGG16 → 이미지 분류 문제 해결
- RNN/LSTM → 시계열 예측 및 NLP 문제 활용
- Dropout 적용 → 학습/테스트 안정화
- 모델 저장/로드 → 학습 결과 재사용 및 평가

**제5고지 목표**

- 신경망 기능 확장, GPU 활용 능력 확보
- CNN, RNN, LSTM 구조 이해 및 구현 가능
- 실전 데이터 학습, 평가, 최적화 능력 확보

- DeZero 프레임워크 기반 실전 신경망 구축 능력 완전 확보

# CHAPTER 1 밴디트 문제

### 머신러닝 분류와 강화 학습

- 지도학습 : 입력 → 출력, 레이블 기반 학습
- 비지도학습 : 데이터 구조, 패턴 발견
- 강화학습 : 에이전트가 환경과 상호작용 → 보상 최대화
- 핵심 : 행동(Action), 상태(State), 보상(Reward) 정의
- 학습 목표 : 장기 보상 최대화 정책(policy) 학습

### 밴디트 문제

- K-armed bandit : K개의 슬롯머신, 각 행동에 기대 보상 존재
- 탐험(Exploration) vs 이용(Exploitation) 딜레마
- 목표 : 제한된 시도 내에서 최대 누적 보상 획득

### 밴디트 알고리즘

- ε-greedy : 확률 ε로 랜덤 행동 → 나머지 확률로 최적 행동
- Softmax : 행동 선택 확률을 보상 기대값 기반으로 분포화
- Upper Confidence Bound(UCB) : 불확실성 고려, 적은 시도된 행동 우선 선택

### 밴디트 알고리즘 구현

- 변수 : 각 행동별 평균 보상 Q(a), 선택 횟수 N(a)
- 행동 선택 : ε-greedy 혹은 Softmax
- 보상 관찰 → Q(a) 업데이트 : `Q(a) ← Q(a) + α (r - Q(a))`
- 반복 → 누적 보상 계산

### 비정상 문제

- 보상 분포 변화 Non-stationary 문제

- 이동 평균 적용 : 최근 보상 기반 Q(a) 업데이트
- α : 학습률 조절 → 빠른 적응 가능

**정리**

- 단일 행동 선택 문제 이해
- 탐험과 이용 균형, 보상 업데이트 방법 학습
- Python/Numpy 기반 시뮬레이션 가능

# CHAPTER 2 마르코프 결정 과정

**마르코프 결정 과정(MDP) 정의**

- 상태 s ∈ S, 행동 a ∈ A, 보상 r ∈ R, 상태 전이 확률 P(s'|s,a)
- Markov Property : 현재 상태만 고려, 과거 정보 불필요
- 정책 π(a|s) : 상태에서 행동 선택 확률
- 목표 : 장기 할인 보상 최대화

**환경과 에이전트를 수식으로**

- 에이전트 → 행동 선택
- 환경 → 상태 전이, 보상 반환
- 상태 가치 함수 V(s) : 장기 기대 보상
- 행동 가치 함수 Q(s,a) : 행동 선택 시 기대 보상

**MDP 예제**

- Gridworld : 각 칸 상태, 움직임 행동, 보상 정의
- 정책 평가 : 주어진 π에 따른 V(s) 계산
- 최적 정책 : Q(s,a) 최대화 → 행동 선택

**정리**

- MDP로 강화학습 문제 수식화
- 상태, 행동, 보상, 정책 구조 이해
- 강화학습 알고리즘 기반 구조 마련

# CHAPTER 3 벨만 방정식

### 벨만 방정식 도출

- V(s) = E[r + γ V(s')] : 상태 가치 함수 재귀 정의
- 정책 π 기반 기대값 계산 → Policy Evaluation

### 벨만 방정식의 예

- Gridworld, 간단한 보상 환경 적용
- Forward 계산 → 각 상태 가치 반복 업데이트

### 행동 가치 함수(Q 함수)와 벨만 방정식

- Q(s,a) = E[r + γ Σ_a' π(a'|s') Q(s',a')]
- Q 함수 활용 → 최적 행동 선택 가능
- 정책 평가와 개선 반복

### 벨만 최적 방정식

- 최적 정책 π* : max_a Q(s,a) 선택
- Value Iteration 기반 → 반복 계산
- 수식 : V*(s) = max_a E[r + γ V*(s')]

### 정리

- 가치 함수, Q 함수, 최적 정책 개념 확립
- Bellman backup, recursive 구조 이해
- DP 기반 강화학습 알고리즘 토대 마련

# CHAPTER 4 동적 프로그래밍

### 동적 프로그래밍과 정책 평가

- 반복적 Bellman backup → Policy Evaluation
- π 기반 V(s) 반복 계산, 수렴 확인
- 구현 : Python/Numpy, 상태 공간 반복

### 정책 반복법(Policy Iteration)

- 정책 평가 → 정책 개선 반복

- 수렴 후 최적 정책 획득
- 코드 흐름 : 반복 루프 내 정책 업데이트

**가치 반복법(Value Iteration)**

- 정책 평가 + 정책 개선 통합
- V(s) 변화 확인 → Convergence → 최적 정책 결정

**정리**

- DP 기반 정책 최적화 이해
- Bellman backup 반복 구조, Python 구현 가능
- 강화학습 최적 정책 계산 능력 확보

# CHAPTER 5 몬테카를로법

**몬테카를로법 기초**

- 확률적 시뮬레이션 기반, 샘플링으로 기대값 추정
- 정책 평가와 최적화 가능
- 핵심 : 여러 에피소드 샘플 → 평균 보상 계산
- 수식 : ( V(s) = E[G_t | S_t = s] ), G_t : 누적 보상

**몬테카를로법으로 정책 평가하기**

- 주어진 정책 π 실행 → 에피소드 생성
- 각 상태 s의 누적 보상 G_t 기록 → V(s) 업데이트
- First-visit MC : 처음 방문 시 보상만 사용
- Every-visit MC : 모든 방문 시 보상 사용

**몬테카를로법 구현**

- Python/Numpy 기반
- 에피소드 반복 생성 → 상태 방문 기록, 누적 보상 계산
- 학습률 α 적용 가능 : 점진적 평균 업데이트

**몬테카를로법으로 정책 제어하기**

- ε-greedy + MC → 정책 개선 가능

- 상태-행동 가치 Q(s,a) 계산
- 정책 개선 : π(s) = argmax_a Q(s,a)
- 수렴 반복 → 최적 정책 획득

**오프-정책과 중요도 샘플링**

- 행동 정책과 대상 정책 분리
- Off-policy 학습 : 중요도 샘플링 적용
- 가중치 w = π_target(a|s)/π_behavior(a|s) → 보상 조정

**정리**

- 샘플 기반 정책 평가와 개선 가능
- On-policy, Off-policy 학습 이해
- Python 구현으로 에피소드 기반 강화학습 실습 가능

# CHAPTER 6 TD법

**TD법으로 정책 평가하기**

- Temporal Difference(TD) : Monte Carlo + Dynamic Programming 통합
- V(s) 업데이트 : ( V(s) ← V(s) + α (r + γ V(s') - V(s)) )
- 장점 : 에피소드 끝까지 기다리지 않고 온라인 학습 가능

**SARSA**

- On-policy TD Control
- Q(s,a) 업데이트 : ( Q(s,a) ← Q(s,a) + α (r + γ Q(s',a') - Q(s,a)) )
- 에피소드 진행 중 다음 행동 선택 → 정책과 일치

**오프-정책 SARSA**

- 행동 정책 ≠ 대상 정책
- ε-greedy 등 탐험 정책 적용
- 중요도 샘플링 없이 행동 정책에 따른 기울기 적용

**Q 러닝**

- Off-policy TD Control

- Q(s,a) 업데이트 : ( Q(s,a) ← Q(s,a) + α (r + γ max_{a'} Q(s',a') - Q(s,a)) )
- 최적 정책 학습 가능, 탐험 정책과 독립

**분포 모델과 샘플 모델**

- 모델 기반 vs 모델 프리 강화학습
- TD법 : 모델 필요 없음 → 환경 상호작용 기반
- 샘플 모델 활용 → 시뮬레이션 반복, 학습 효율화

**정리**

- TD법 : 온라인 정책 평가와 제어
- SARSA, Q-learning 비교
- Python/Numpy 기반 TD 학습 구현 가능

# CHAPTER 7 신경망과 Q 러닝

**DeZero 기초**

- Variable, Function, 자동 미분 구조 활용
- Forward/Backward 구조 → Q-value 학습에 적용

**선형 회귀**

- 입력 상태 s → Q(s,a) 예측
- 손실 : MSE 또는 Huber Loss

**신경망**

- 은닉층 추가 → 비선형성 확보
- 활성화 함수 : ReLU, tanh 등
- Forward : s → Q(s,a) 출력
- Backward : 손실 기반 기울기 전파

**Q 러닝과 신경망**

- Deep Q-Network(DQN) : Q(s,a) → 신경망 예측
- 경험 재플레이(Experience Replay) → 샘플 균일 학습
- 타겟 네트워크(Target Network) → 학습 안정화

**정리**

- TD + 신경망 → DQN 기반 강화학습 이해
- Python/DeZero 기반 신경망 구현 가능
- 온라인 학습, 경험 재플레이, 타겟 네트워크 적용

# CHAPTER 8 DQN

**OpenAI Gym**

- 강화학습 환경 제공
- 상태 관찰, 행동 선택, 보상 반환
- Python API 활용 → 시뮬레이션 가능

**DQN의 핵심 기술**

- 신경망 Q-value 예측
- 경험 재플레이 → 데이터 샘플링 균일
- 타겟 네트워크 → 안정적 학습
- ε-greedy 정책 적용 → 탐험/이용 균형

**DQN과 아타리**

- Atari 게임 적용 사례
- 프레임 입력 → CNN 적용 → Q-value 출력
- 학습 곡선, 성능 평가

**DQN 확장**

- Double DQN : 과대 추정 문제 완화
- Dueling DQN : 가치와 이득 분리 → 학습 효율 개선
- Prioritized Replay : 중요 샘플 우선 학습

**정리**

- Deep Q-learning 구조 이해
- CNN + DQN 적용 → 실제 게임 환경 학습 가능
- Python/DeZero 기반 구현

# CHAPTER 9 정책 경사법

### 가장 간단한 정책 경사법

- 정책 $\pi_\theta(a|s)$ 직접 파라미터화
- 목표 : 누적 보상 기대값 최대화
- 파라미터 θ 업데이트 : Gradient Ascent

### REINFORCE

- Monte Carlo 기반 정책 경사법
- 업데이트 : ( $\theta \leftarrow \theta + \alpha\, G_t\, \nabla_\theta \log \pi_\theta(a_t|s_t)$ )
- 에피소드 단위 학습

### 베이스라인

- 분산 감소 → 가치 함수 V(s) 활용
- 업데이트 : ( $\theta \leftarrow \theta + \alpha\, (G_t - b(s))\, \nabla_\theta \log \pi_\theta(a_t|s_t)$ )
- 안정적 학습 가능

### 행위자-비평자(Actor-Critic)

- Actor : 정책 파라미터 θ
- Critic : 가치 함수 V(s)
- TD 기반 업데이트 → 온라인 학습 가능

### 정책 기반 기법 장점

- 연속 행동 공간 처리 가능
- 안정적 학습, 탐험 정책 직접 설계 가능

### 정리

- Policy Gradient, REINFORCE, Actor-Critic 이해
- Monte Carlo + TD 기반 파라미터 업데이트 구현 가능

# CHAPTER 10 한 걸음 더

**심층 강화 학습 알고리즘 분류**

- Value-based : DQN 계열
- Policy-based : 정책 경사법 계열
- Actor-Critic : 혼합형

**정책 경사법 계열의 고급 알고리즘**

- A2C, PPO, TRPO 등 → 안정적 학습, 분산 감소, 샘플 효율화

**DQN 계열의 고급 알고리즘**

- Double DQN, Dueling DQN, Prioritized Replay, Rainbow DQN
- 안정성과 성능 개선, 다양한 게임 환경 적용 가능

**사례 연구**

- Atari, MuJoCo 등 실제 환경 학습 사례
- 성능 분석, hyperparameter 중요성

**심층 강화 학습이 풀어야 할 숙제와 가능성**

- 탐험 효율화, 샘플 효율, 안정적 학습
- 연속 행동, 대규모 상태 공간
- 현실 세계 적용 가능성, 자율 시스템 확장

**정리**

- 강화학습 최신 기술 이해
- Policy Gradient, DQN 고급 기법 구조 숙지
- 실제 환경 적용 및 성능 분석 가능