DEEP LEARNING
WITH PYTORCH

ABOUT AUTOGRAD

ABOUT AUTOGRAD

◆ 인공신경망 (ANN : Artificial Neural Network)

❖ 동작원리

[STEP1] 순전파 FORWARD PROPAGATION

- **입력층 ======> 출력층 방향** 계산 과정
- **신호와 가중치 곱한 값** 출력층까지 차례대로 계산
- 피쳐1*가중치1 + 피쳐2*가중치2 + ... + 피쳐n*가중치n + b

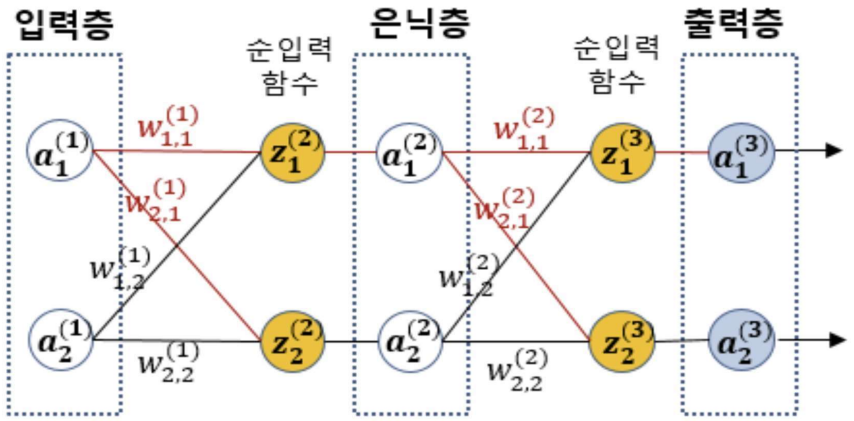
  ➔ 활성화함수AF(피쳐가중치합+b) ➔ 결과값

ABOUT AUTOGRAD

◆ 인공신경망 (ANN : Artificial Neural Network)

❖ 동작원리

[STEP1] 순전파 FORWARD PROPAGATION



ABOUT AUTOGRAD

◆ 인공신경망 (ANN : Artificial Neural Network)

❖ 동작원리

[STEP2] 역전파 BACKWARD PROPAGATION

- **입력층 <====== 출력층 방향** 계산 과정
- **오류에 대한 미분값, 학습률, 경사하강법 이용 최적화 값 입력층으로 전달**
- 현재 W, b - ((정답-예측값)의 미분값 * 학습률)

  ➔ 새로운 W, b 업데이트

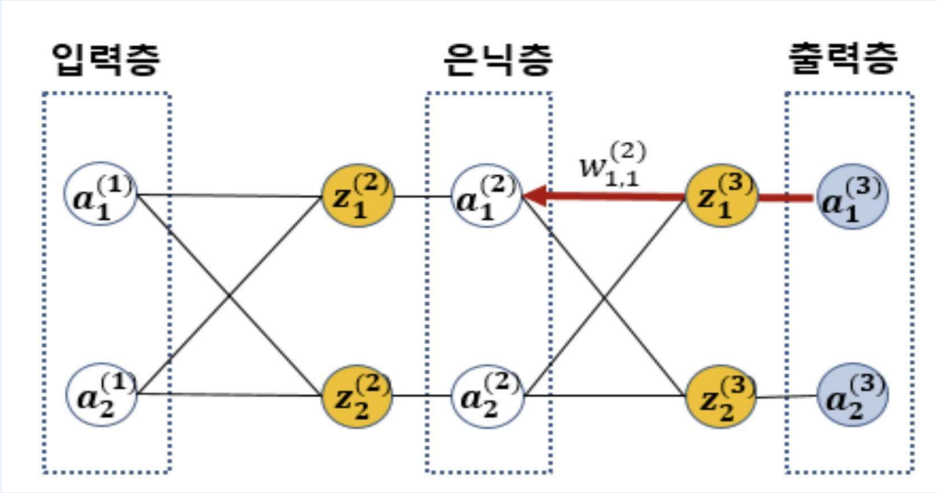
  ➔ 이전 층으로 전달

ABOUT AUTOGRAD

◆ 인공신경망 (ANN : Artificial Neural Network)

❖ 동작원리

[STEP2] 역전파 BACKWARD PROPAGATION

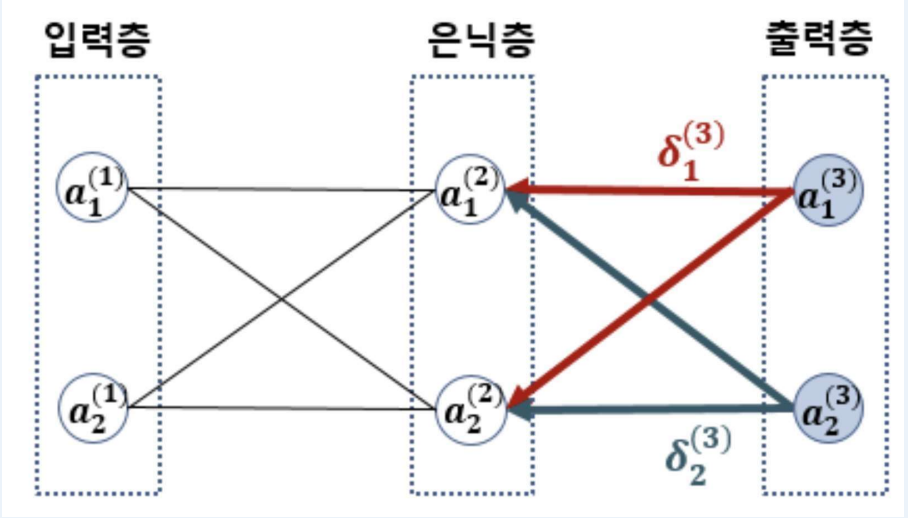


ABOUT AUTOGRAD

◆ 인공신경망 (ANN : Artificial Neural Network)

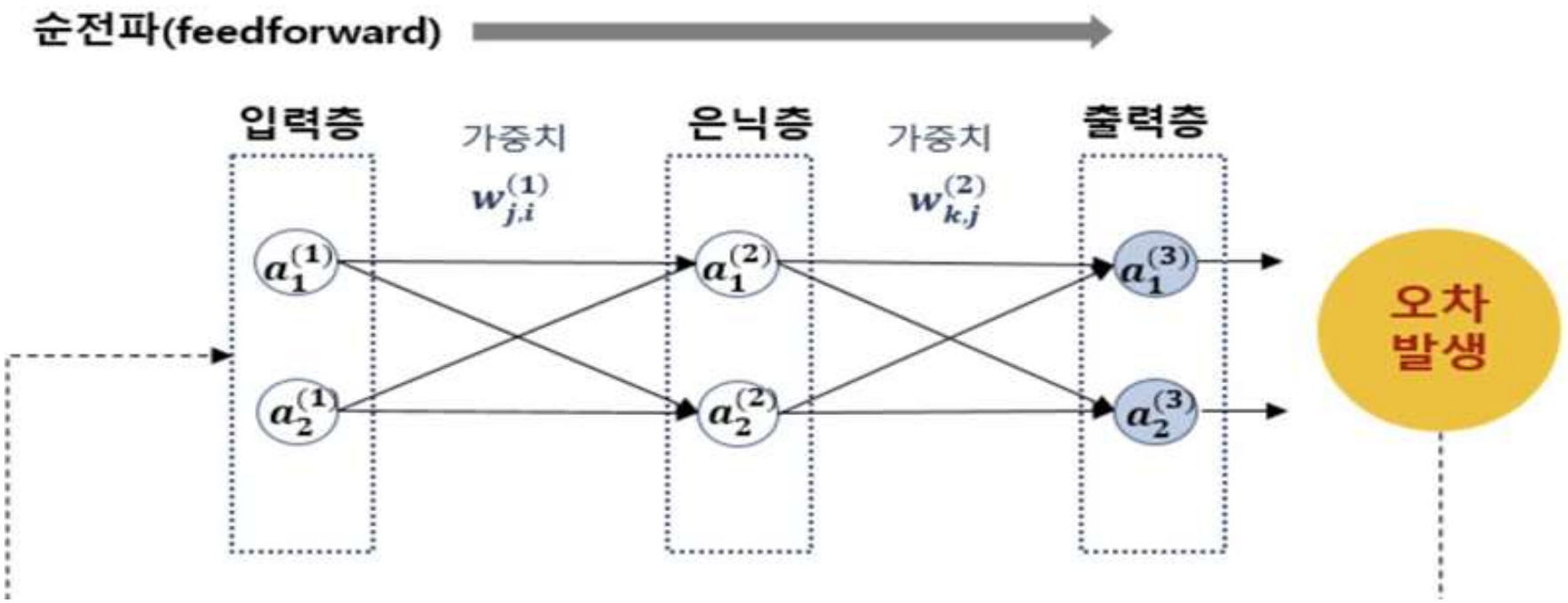
❖ 동작원리

[STEP2] 역전파 BACKWARD PROPAGATION



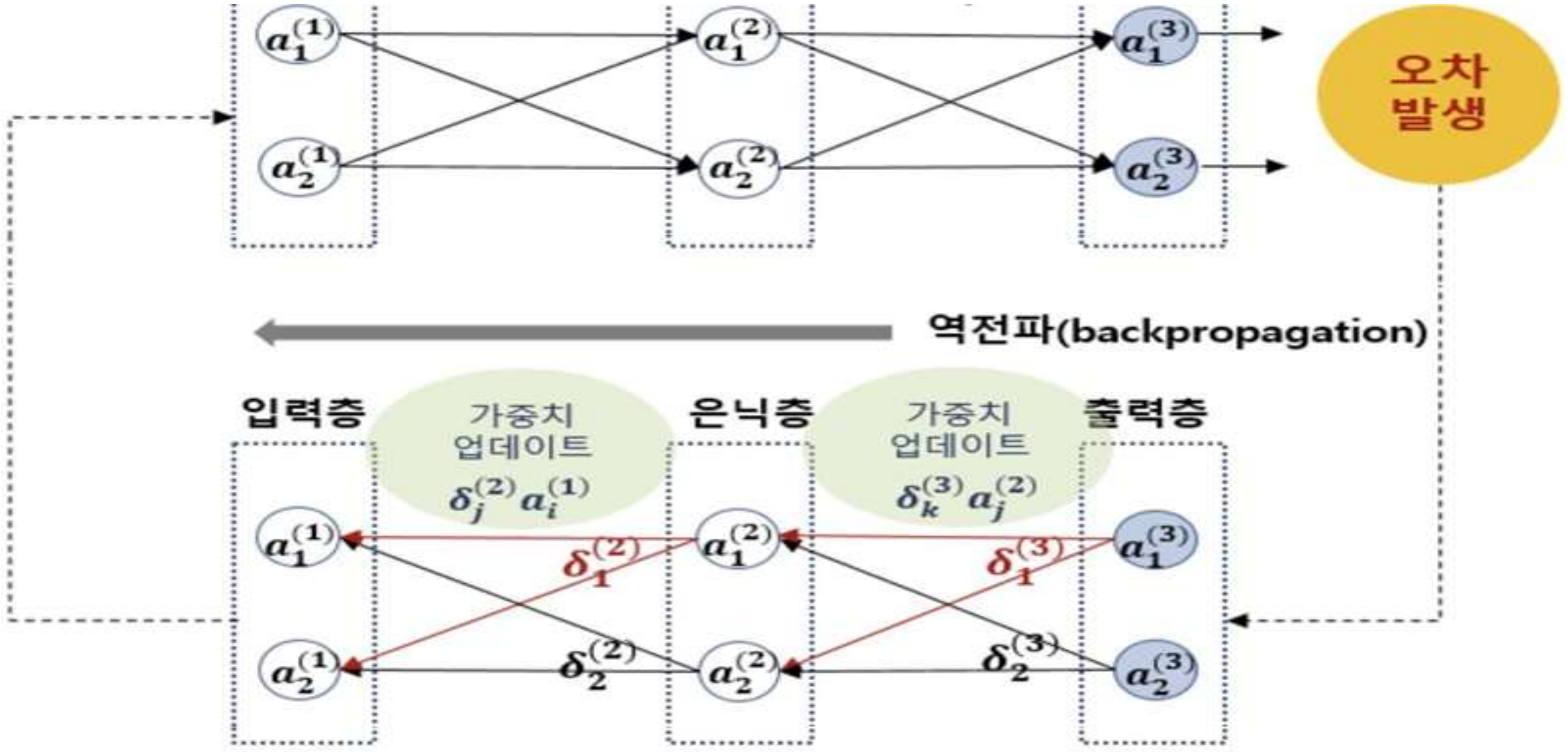
ABOUT AUTOGRAD

◆ 인공신경망 (ANN : Artificial Neural Network)



ABOUT AUTOGRAD

◆ 인공신경망 (ANN : Artificial Neural Network)



ABOUT AUTOGRAD

◆ 오류역전파 알고리즘 (Backward Propagation)

→1986년 제안된 효율적 최적화 알고리즘

→ 경사하강법 이용 파라미터(w, b) 업데이트하며 학습 진행

→ 각 Layer에서 손실함수 미분값 계산에 어려움을 해결한 알고리즘

→ 미분학의 '체인툴 ' 착안 ➔ 연쇄법칙

→ 출력층에서 입력층로 오류를 전달하며 파라미터(w, b) 업데이트

ABOUT AUTOGRAD

◆ 가중치/절편 최적화

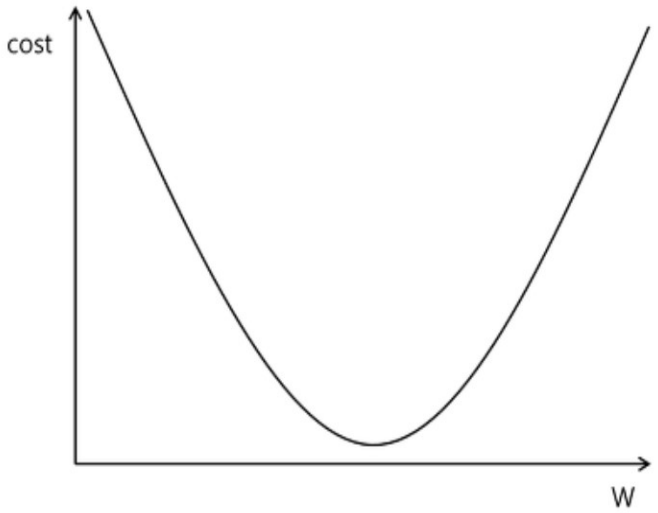
❖ **잔차/오차와 가중치, 절편 관계**

→ **잔차(Residual)** : 표본 회귀식과 관측치 차이

→ **오차(Error)** : 모집단의 회귀식과 관측치 차이

→ **최적화(Optimizer)**

잔차/오차 최소화 하는 것



ABOUT AUTOGRAD

◆ 가중치/절편 최적화

❖ 목적함수(Objective Function)

→ 최소화 또는 최대화 하고 싶어하는 함수

→ 최소화 함수

- 비용함수(cost function) : 전체 데이터 셋 대한 **예측값과 실제값 차이 측정 함수**
- 손실함수 (loss function) : 한 개 데이터 셋 대한 **예측값과 실제값 차이 측정 함수**
- 오류함수 (error function) : 딥러닝에서 손실 함수의 다른 말

- 최대화 함수
  - MLE, KL-Divergence

ABOUT AUTOGRAD

◆ 가중치/절편 최적화

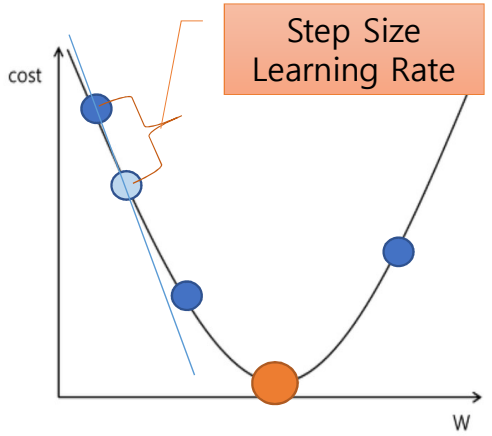
❖ **최적화 방법 - 경사하강법(Gradient Descent)**

→ 수학적 기법 중 하나

→ **손실함수 값을 최소화하기 위해 기울기를 이용하는 사용**

→ **step-size 간격**으로 수행, 보통 0.1~0.001 속도가 적당

→ 학습 데이터 수, 학습률에 따른 다양한 방법 존재



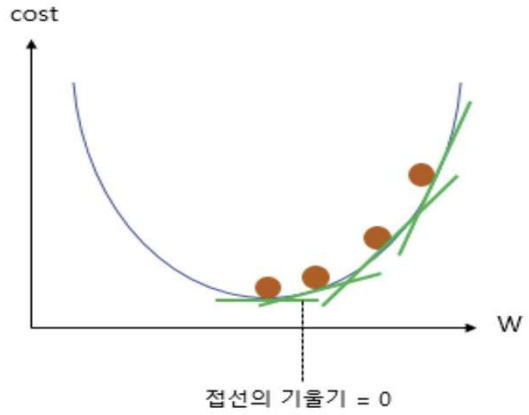
ABOUT AUTOGRAD

◆ 가중치/절편 최적화

❖ 최적화 방법 - 경사하강법(Gradient Descent)

→ 새로운 W, b 계산 : 현제 W - (오류/잔차/차이값 미분한 값 * step-size)

→ 접선의 기울기 즉, 미분



ABOUT AUTOGRAD

◆ 가중치/절편 최적화

❖ 최적화 방법 - 경사하강법(Gradient Descent)

① 처음 시작점(Initial starting point)은 **랜덤으로 시작**하여 **시작점의 기울기,▽f(x)** 구하여 **기울기의 경사정도(절대값)과 음 또는 양의 방향인지 확인**

② **Learning rate/step size**라 불리는 r의 값을 임의적으로 수정하여 **이동하고자 하는 거리만큼 점 이동**

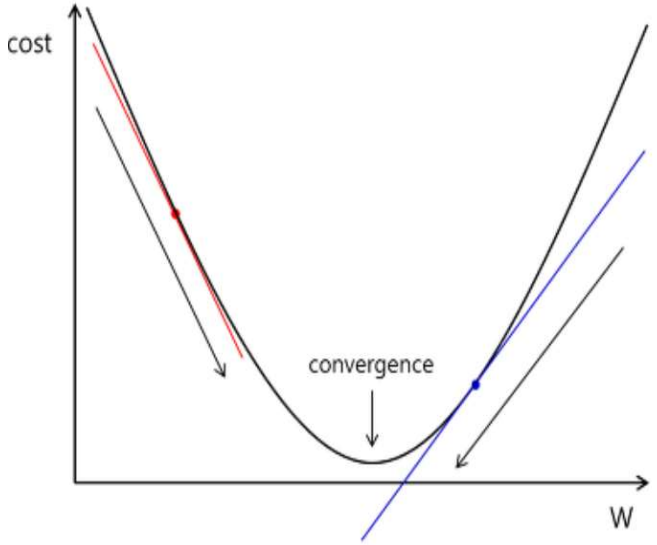
② 업데이트된 점에서 **기울기를 구하여 0의 수렴하는지 확인**

③ **기울기가 0의 수렴하지 않으면 1번으로 돌아가서 반복실행**

ABOUT AUTOGRAD

◆ 가중치/절편 최적화

❖ 최적화 방법 - 경사하강법(Gradient Descent)



ABOUT AUTOGRAD

◆ 가중치/절편 최적화

❖ 최적화 방법 - 경사하강법(Gradient Descent)

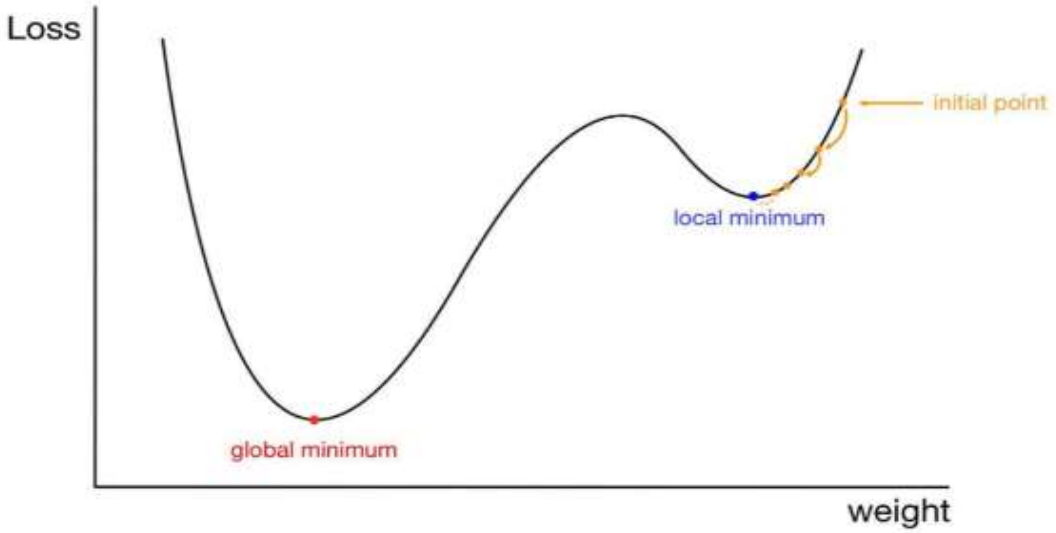
Local Minimum

실제 최적화 지점이 아닌 지점

최저점을 제외한 나머지 지점들

Global Minimum

최저점



ABOUT AUTOGRAD

◆ 가중치/절편 최적화

❖ 최적화 방법 - 경사하강법(Gradient Descent)

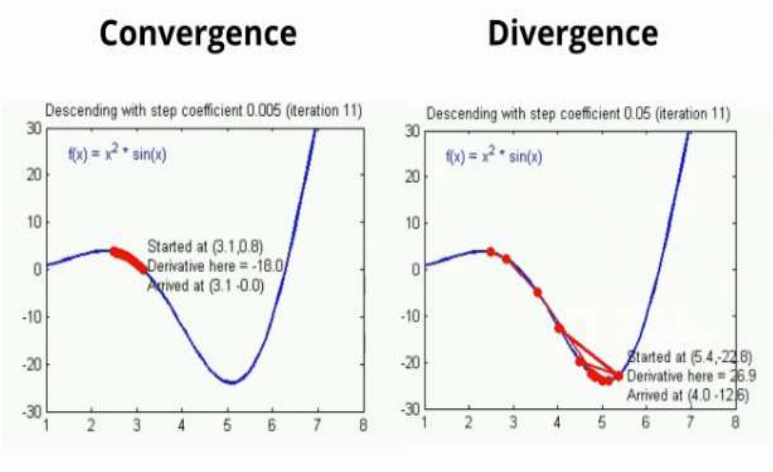
Local Minima Problem

Global minimum 아닌 Local minimum에서 멈추는 현상

사실상 고차원공간에서 발생하기 힘든 현상

➔ **Momentum, Adagrad, Adam 등의 회피 최적화기법**

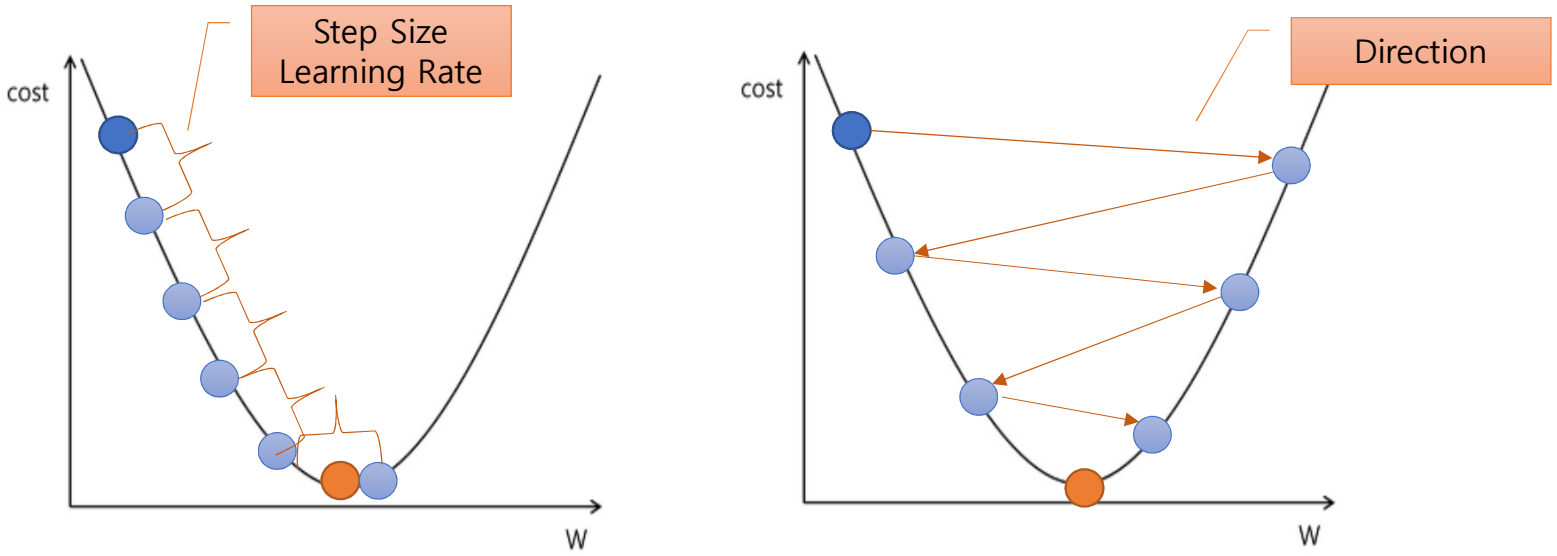
➔ **Learning Rate 적절히 조절**



ABOUT AUTOGRAD

◆ 가중치/절편 최적화

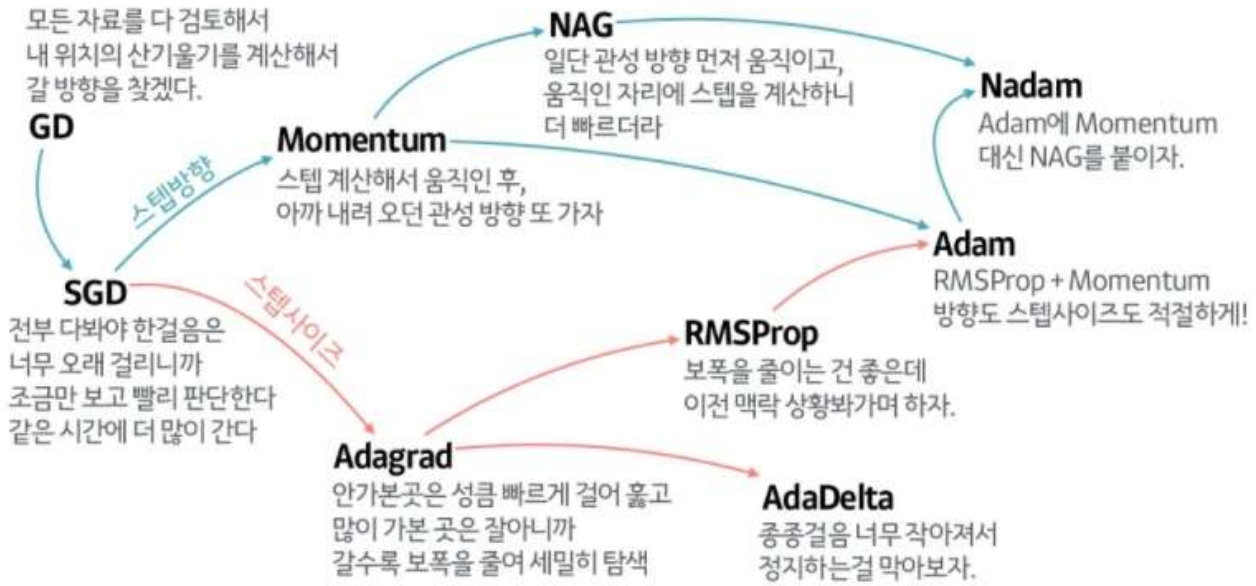
❖ 최적화 방법 - 경사하강법(Gradient Descent)



ABOUT AUTOGRAD

◆ 가중치/절편 최적화

❖ 다양한 경사하강법



ABOUT AUTOGRAD

◆ 가중치/절편 최적화

❖ 다양한 경사하강법(Gradient Descent)

Batch Gradient Descent	전체 데이터 학습 후 검증 및 업데이트 진행, 많은 시간 및 계산량 소요
SGD (Stochastic Gradien Descent)	일부 학습 데이터(mini-batch) 선택 후 진행 BGD에 비해 다소 부정확, 속도 빠름
Momentum	업데이트 시 이전 값과 비교하여 같은 방향으로 업데이트 진행 -> 관성
AdaGrad **(Adaptive Gradient)**	변수들 update 시 각각의 변수마다 lr 즉 step size 다르게 설 정 갱신 정도 약해져 전혀 갱신되지 않는 경우 발생
RMSProp / AdaDelta	AdaGrad의 단점 보완, 과거 기울기는 조금 반영 + 최신 기울기 많이 반영
Adam **(Adaptive Moment Estimation)**	과거 미분값 계속 가중평균 내면서 효율적 업데이 트 RMSProp + Momentum 방식 결합

ABOUT AUTOGRAD

◆ torch.autograd

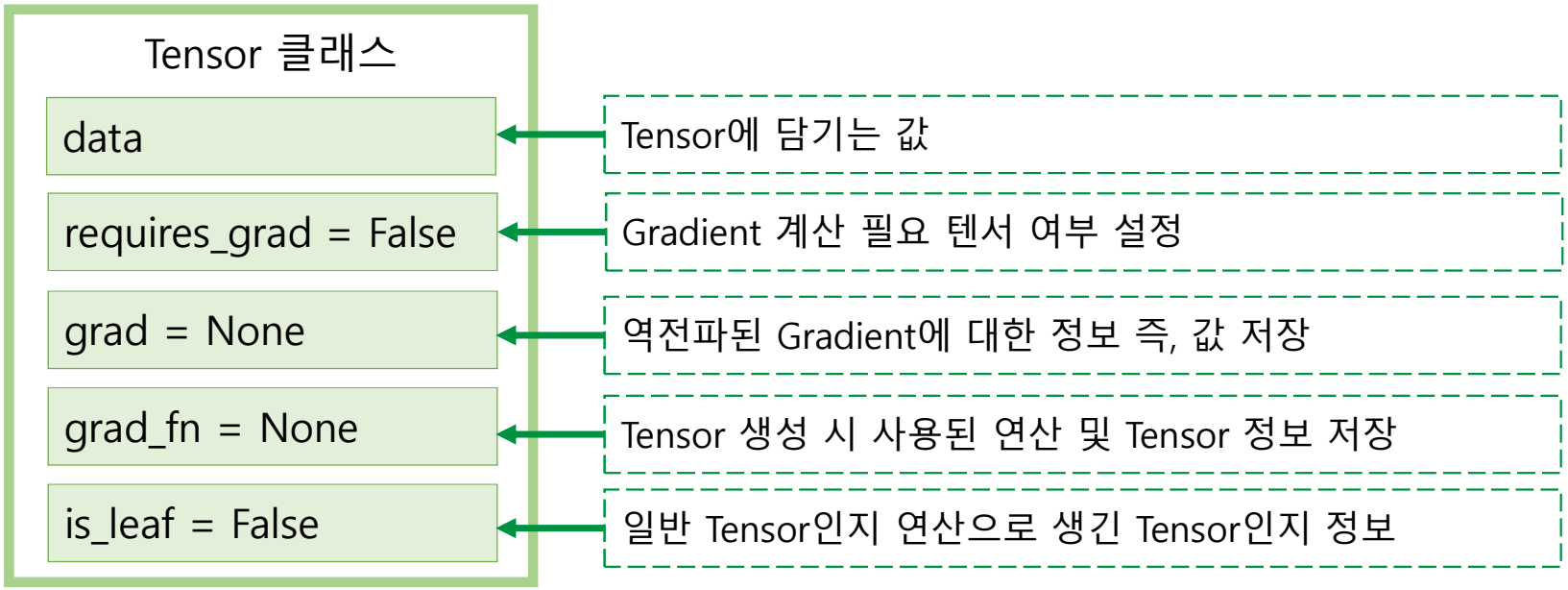
❖ **자동미분 기능**

- 모델 복잡해질수록 **경사 하강법을 넘파이 등으로 직접 코딩하는 것은 까다로움**
- **파이토치**에서는 이런 수고를 하지 않도록 **자동 미분(Autograd) 지원**

ABOUT AUTOGRAD

◆ torch.autograd

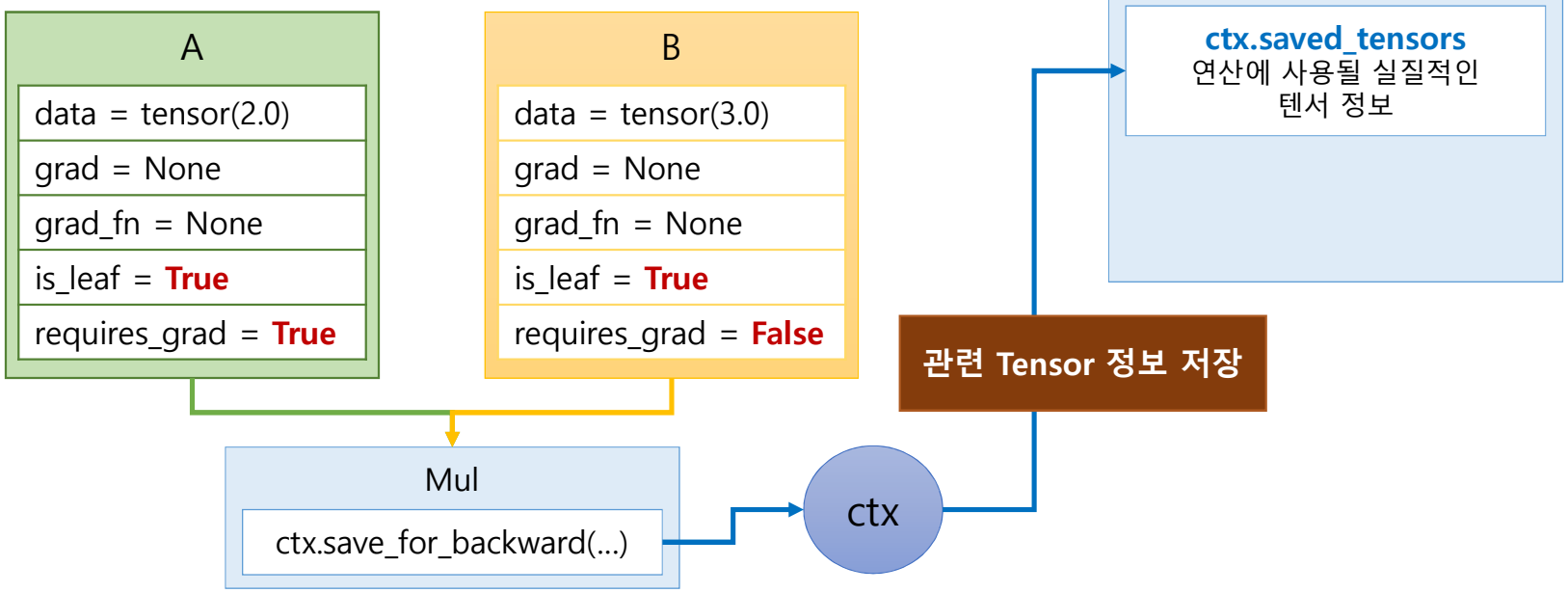
❖ **자동미분 관련 Tensor 속성**



ABOUT AUTOGRAD

◆ torch.autograd

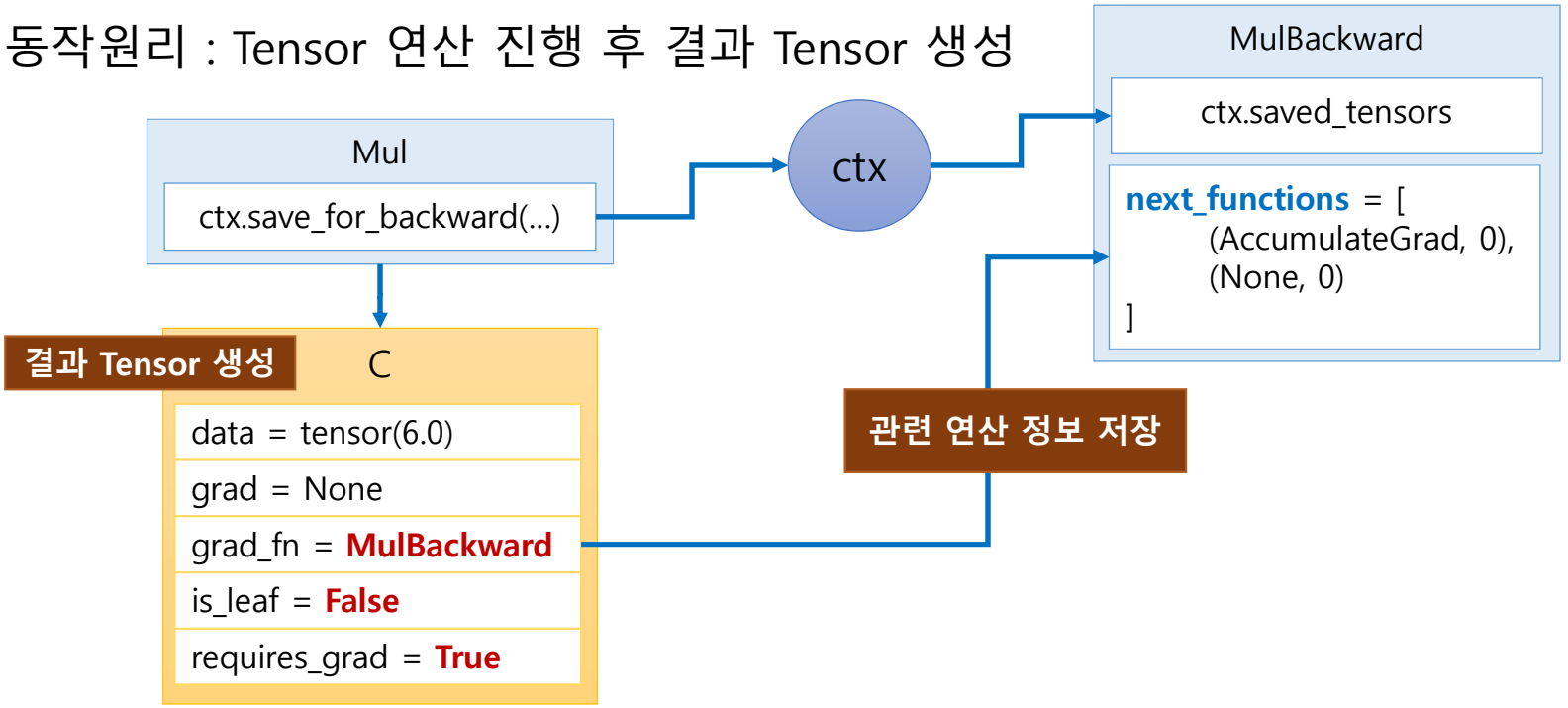
❖ 동작원리 : Tensor 연산 진행 후 결과 Tensor 생성



ABOUT AUTOGRAD

◆ torch.autograd

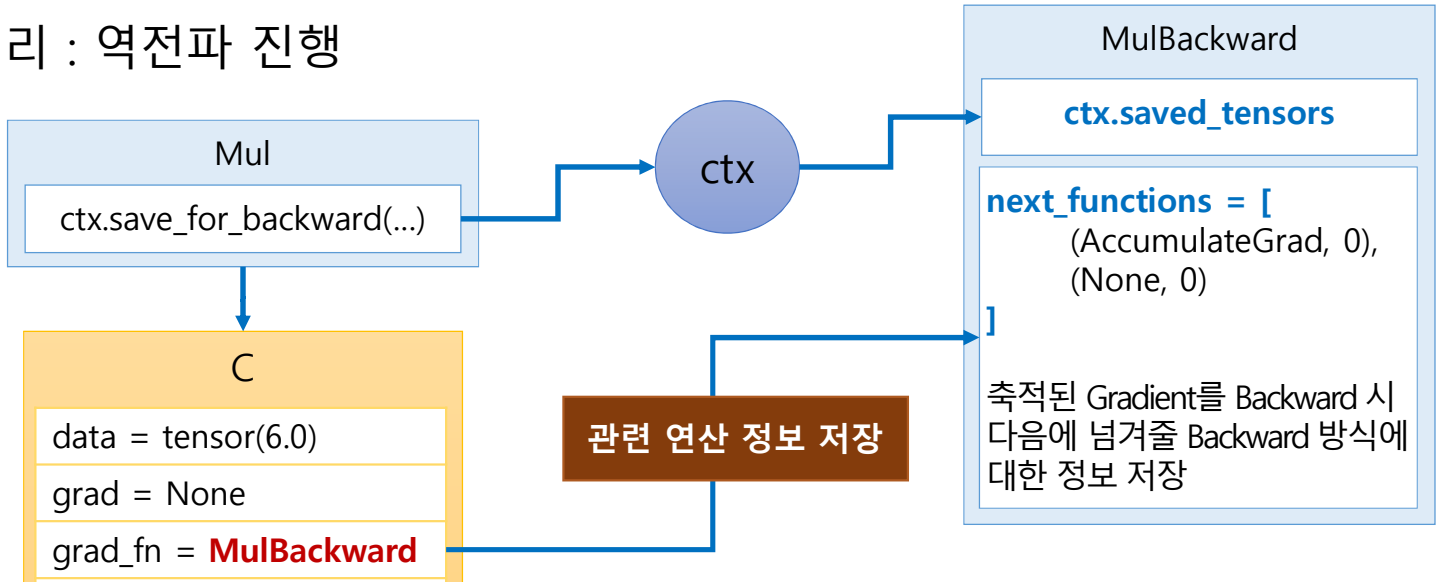
❖ 동작원리 : Tensor 연산 진행 후 결과 Tensor 생성



ABOUT AUTOGRAD

◆ torch.autograd

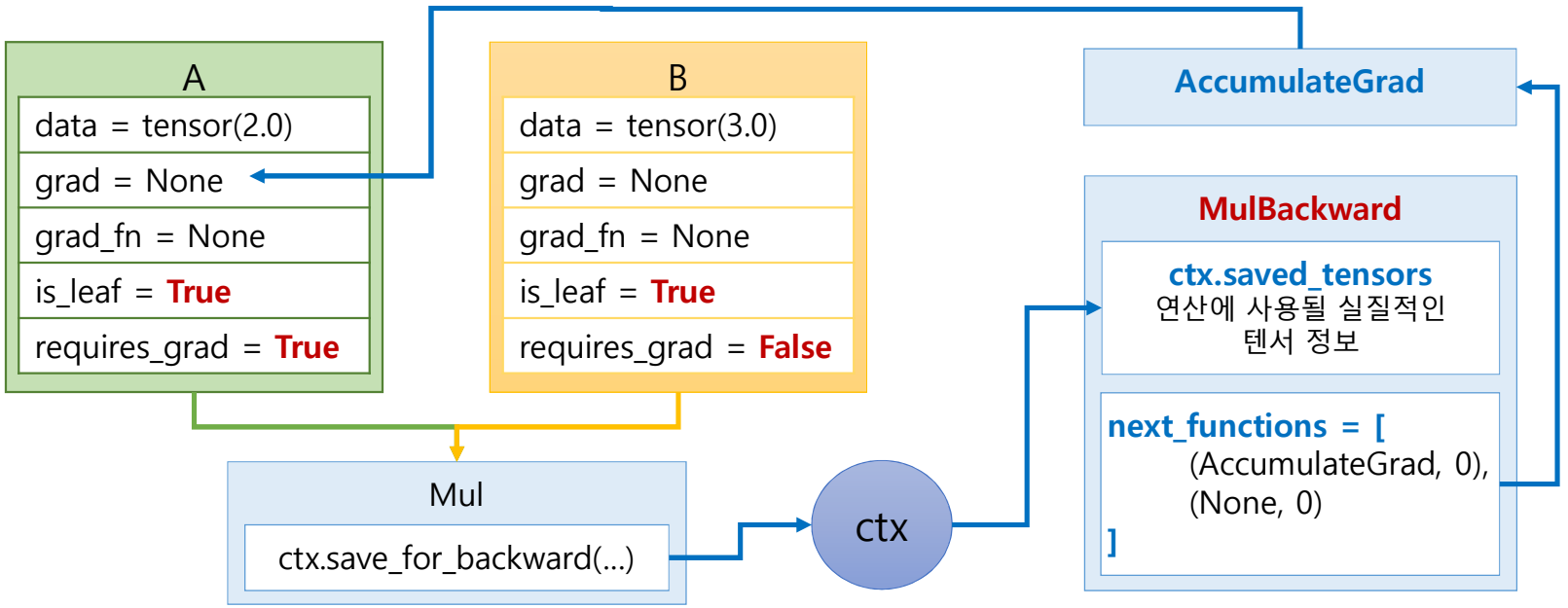
❖ 동작원리 : 역전파 진행



ABOUT AUTOGRAD

◆ torch.autograd

❖ 동작원리 : 역전파 진행



ABOUT AUTOGRAD

◆ torch.autograd

❖ **데이터 업데이트 조건**

- .requires_grad = True
- .is_leaf = True

ABOUT AUTOGRAD

◆ torch.autograd

❖ 자동미분 설정 ➔ requires_grad = True 설정된 Tensor 업데이트 진행됨

```
# 역전파 진행으로, 가중치/절편 업데이트
loss.backward()

for name, param in linear_model.named_parameters():
        print(name, param)
```

ABOUT AUTOGRAD

◆ torch.autograd

❖ 자동미분 해제 ➔ torch.no_grad() , detach()

```
with torch.no_grad():
      z = torch.matmul(x, w)+b
print(z.requires_grad)
```

```
z = torch.matmul(x, w)+b
z_det = z.detach()
print(z_det.requires_grad)
```

ABOUT AUTOGRAD

◆ 손실함수(Loss Function)

❖ torch.nn.functional

```
l1_loss

mse_loss

margin_ranking_loss

multilabel_margin_loss

multilabel_soft_margin_loss
```

❖ torch.nn.Loss Class

```
nn.L1Loss

nn.MSELoss

nn.CrossEntropyLoss

nn.CTCLoss

nn.NLLLoss

nn.PoissonNLLLoss
```

ABOUT AUTOGRAD

◆ 손실함수(Loss Function)

❖ torch.nn.functional / torch.nn

유형		손실함수	
회귀		torch.nn.**MSELoss**	평균 제곱 오차(Mean Squared Error)
		torch.nn.**L1Loss**	평균 절대 오차(Mean Absolute Error)
분류	다중	torch.nn.**CrossEntropyLoss**	교차 엔트로피 손실(Cross Entropy Loss)
	이진	torch.nn.**BCELoss**	이진 교차 엔트로피 손실(Binary Cross Entropy Loss)

ABOUT AUTOGRAD

◆ 가중치/절편 최적화 클래스

❖ torch.optim 클래스

- W, b 업데이트 처리 클래스
- 필수 매개변수 : 모델 파라미터(model.parameters()와 학습률(lr)

torch.optim.**SGD**	가장 기본적인 최적화 알고리즘 각각의 파라미터에 대해 학습률(learning rate) 곱한 값 사용 가중치 업데이트
torch.optim.**Adam**	학습률을 각 파라미터마다 적응적으로 조절하는 최적화 알고리즘 현재 그래디언트와 이전 그래디언트의 지수 가중 평균 이용 가중치 업데이트
torch.optim.**RMSprop**	그래디언트의 제곱값의 이동 평균을 이용하여 학습률 조절하는 최적화 알고리즘 Adam과 유사한 방식으로 학습률 조절
torch.optim.**Adagrad**	각 파라미터에 대한 학습률을 조절하는 최적화 알고리즘 이전 그래디언트 제곱의 누적 값 사용하여 학습률 조절

ABOUT AUTOGRAD

◆ 가중치/절편 최적화 클래스

❖ Model Parameters

▪ 층별 W, b 텐서들로 역전파 시 업데이트

model .parameters()	• 모델의 학습 가능한 파라미터들 반환 • 모델의 모든 레이어 및 모듈에서 정의된 파라미터들 포함 • 모델이 학습 중에 업데이트되는 값들 • 최적화 알고리즘 통해 업데이트되는 대상 • 학습 데이터와 손실 함수를 통해 계산된 그라디언트를 사용하여 업데이트
model .named_parameters()	• parameters()와 동일 기능 • 파라미터 이름과 값을 전달

ABOUT AUTOGRAD

◆ 가중치/절편 최적화

❖ torch.optim 클래스

```
# 가중치/절편 최적화 방법 설정
from torch.optim import Adam

# W, b 업데이트위한 모델 파라미터 설정
optimizer = Adam( linear_model.parameters(), lr=LR )
```

Adadelta

Adagrad

Adam

AdamW

SparseAdam

ABOUT AUTOGRAD

◆ 가중치/절편 최적화

❖ torch.optim 클래스

```
        # 가중치 기울기 0 초기화
        optimizer.zero_grad()

        # 학습 진행
        pre_y = linear_model(x)

        # 손실 계산
        loss = nn.MSELoss()(pre_y, y.reshape(-1,1))

        # 역전파 진행
        loss.backward()

        # 가중치/절편 업데이트
        optimizer.step()
```

MODEL TRAIN & TEST

MODEL TRAIN & TEST

◆ 학습 과정

❖ 주제 및 목표 선정

MODEL TRAIN & TEST

◆ 학습

❖ 데이터 분석

- **피쳐와 타겟 선정**
- **학습 방법 선정**
  - → 지도 학습 : 문제/속성/피쳐 + 정답/라벨/타겟
  - → 비지도학습 : 문제
  - → 강화 학습 : 문제 + 피드백(상과 벌)
- **학습 종류 선정 : 회귀 / 분류**
  - → 회귀(Regression) : 수치값 예측
  - → 분류(Classification) : 데이터 그룹/묶음 나누기

MODEL TRAIN & TEST

◆ 학습

❖ 데이터 분석

▪ 구현 방법 선정

MODEL TRAIN & TEST

◆ 학습

❖ 데이터셋 분리

- **학습용 데이터셋** : 학습 전용 데이터 셋
- **테스트용 데이터셋** : 학습 후 성능평가 위한 데이터 셋
- **검증용 데이터셋** : 모델 성능 개선 체크용 데이터 셋

MODEL TRAIN & TEST

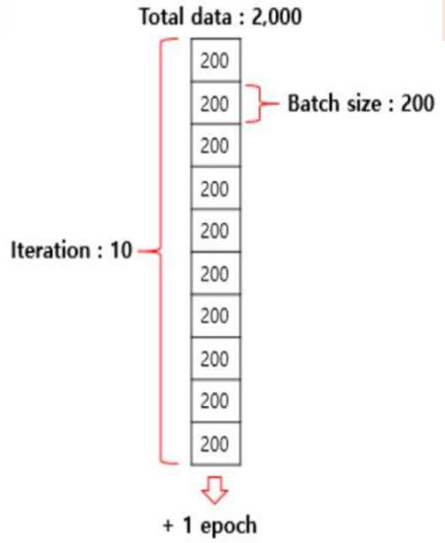
◆ 학습

❖ 학습용 데이터셋 분리

- **딥러닝 대량 데이터 학습 시간/비용 많이 소요 부분 대책**

- **에포크(epochs)** : 처음부터 끝까지 학습하는 횟수
- **배치크기(batch size)** : 전체 데이터를 작은 단위로 나눈 크기
  2의 제곱수 크기
- **이터레이션(iteration)** : 에포크, 배치크기로 계산한 반복 횟수
  W,b 업데이터 횟수

- 예) 100개 데이터, 배치크기 20개, 에포크 10번



MODEL TRAIN & TEST

◆ 학습

❖ 학습용 데이터 준비

```
# DataFrame ===> Feature 추출
X = bostonDF.iloc[:, :13].values
Y = bostonDF['medv'].values

print(f'X : {type(X)}, {X.shape}\nY : {type(Y)}, {Y.shape}')
```

MODEL TRAIN & TEST

◆ 학습

❖ 학습 설정

```
# 학습 횟수 및 한번에 학습할 데이터 크기 설정
EPOCHS = 500
BATCH_SIZE = 100
LR = 0.001
```

MODEL TRAIN & TEST

◆ 학습

❖ 학습 설정

```
# 모델 인스턴스 생성
linear_model = nn.Sequential(nn.Linear(13, 10),
                             nn.ReLU(),
                             nn.Linear(10, 1))
```

MODEL TRAIN & TEST

◆ 학습

❖ 학습 설정

```
# 가중치/절편 최적화 방법 설정
from torch.optim import Adam

optimizer = Adam(linear_model.parameters(), lr=LR)
```

MODEL TRAIN & TEST

◆ 학습

❖ 학습 설정 - 모델 동작모드 설정

유형	설정 함수	
학 습 모 드	model.train()	* 모델 학습 전 호출 ➔ 훈련 데이터에 대해 학습 시작할 준비 - 정규화(regularization) 기법들 동작 설정 - 모델 파라미터 업데이트를 위해 역전파(backpropagation) 수행 - 그래디언트 계산 수행 - 최적화(optimizer) 알고리즘에 따라 모델 파라미터 업데이트

MODEL TRAIN & TEST

◆ 학습

❖ 학습 설정 - 모델 동작모드 설정

유형	설정 함수	
평 가 모 드	model.eval()	* 모델 평가/추론 모드 전환 * 테스트 데이터나 검증 데이터 사용하여 모델 평가할 때 사용 - 정규화(regularization) 기법들 비활성화 설정 - 드롭아웃 비활성화, 배치 정규화 이동 평균/이동 분산 업데이트 않됨 - 역전파 비활성화로 모델 파라미터 업데이트 않됨

MODEL TRAIN & TEST

◆ 학습

❖ 학습 진행

```
for epoch in range(EPOCHS):
    for i in range(len(X)//BATCH_SIZE):
        start = i*BATCH_SIZE
        end = start + BATCH_SIZE

        # ndarray ==> tensor변환
        x = torch.FloatTensor(X[start:end])
        y = torch.FloatTensor(Y[start:end])
```

MODEL TRAIN & TEST

◆ 학습

❖ 학습 진행

```
# 가중치 기울기 0 초기화
optimizer.zero_grad()

# 학습 진행
pre_y = linear_model(x)

# 손실 계산
loss = nn.MSELoss()(pre_y, y.reshape(-1,1))

# 역전파 진행
loss.backward()

# 가중치/절편 업데이트
optimizer.step()
```

MODEL TRAIN & TEST

◆ 학습

❖ 모델 평가

```
# 모델 성능 평가
pre = linear_model(torch.FloatTensor(X[0, :13]))

pre, Y[0]
```