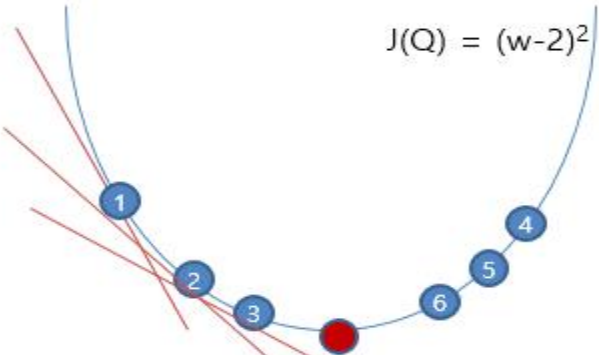


■ 서식 1 : 결과보고서 요약본

Project Based Learning 결과보고서 요약본

교과목명	국문	기계학습1		
	영문	Machine Learning1		
PBL 관련 능력단위 (능력단위코드)		PBL 관련 능력단위요소 (능력단위요소코드)		
기계학습 (YCS-2001020910_21v1)		신경망동작 이해하기		
		프레임워크 사용하기		
학년 반	2학년 1반	조원	이은우	
프로젝트 주제	MxNET 프레임워크를 이용한 선형회귀 객체지향 코드 분석하기			
지도교수	심효선	산업체 참가여부	<input type="checkbox"/> 유 <input checked="" type="checkbox"/> 무	
참여학생	이은우(202025016)			
작품 개요	<p>1. Reading the Dataset</p> <p>1) 기계학습 모델을 훈련하는 과정은 반복적으로 데이터를 적용하는 것으로 주로 미니배치 단위를 사용한다.</p> <p>2) get_dataloader()함수를 SyntheticRegressionData 클래스에 등록한다.</p> <p>3) 미니배치 크기로 (feature, label) 데이터를 묶는다.</p> <p>4) 훈련 과정에서는 랜덤하게 미니배치 데이터를 가져온다.</p> <p>5) 검증 과정에서는 정해진 순서대로 미니배치 데이터를 가져온다.</p> <p>2. 밑바닥에서 선형회귀 구현하기</p> <p>1) 모델 정의</p> <p>2) 손실함수 정의</p> <p>3) 미니배치 스토캐스틱 경사하강법 최적화 알고리즘 정의</p> <p>4) 1), 2), 3)을 모두 합친 훈련함수 정의</p> <p>5) 앞절에서 만든 데이터셋을 모델에 적용하기</p> <p>3. 최적화 알고리즘 정의 (미니배치 SGD)</p> <p>Defining the Optimization Algorithm</p> <p>1) 미니배치의 데이터를 가지고 손실함수의 미분값을 계산</p> <p>2) 손실함수의 값을 줄이는 방향으로 패러미터를 업데이트. 이때 학습률 lr 을 적용</p> <p>3) 학습률 lr 은 초기에 0.01 같은 작은 값을 사용하지만, 모델이 커질수록 튜닝이 필요</p> <p>사용 수식 : <code>param -= self.lr * param.gra</code></p>			

<p>작품 구조도</p>	<ol style="list-style-type: none"> 1. Reading the Dataset <ul style="list-style-type: none"> - 클래스의 용도와 어떻게 사용하는지에 대하여 설명한다. (예제 코드 명시) - 클래스가 가지고 있는 함수들에 대하여 입력, 동작, 출력에 대하여 설명한다. 2. 밑바닥에서 선형회귀 구현하기 <ul style="list-style-type: none"> - 클래스의 용도와 어떻게 사용하는지에 대하여 설명한다. (예제 코드 명시) - 클래스가 가지고 있는 함수들에 대하여 입력, 동작, 출력에 대하여 설명한다. 3. 최적화 알고리즘 정의 (미니배치 SGD) <ul style="list-style-type: none"> - 클래스의 용도와 어떻게 사용하는지에 대하여 설명한다 (예제 코드 명시) - 클래스가 가지고 있는 함수들에 대하여 입력, 동작, 출력에 대하여 설명한다.
<p>관련 이론</p>	<p>스토캐스틱 그레디언트 디센트</p> <p>다층퍼셉트론 모델에서 패러미터의 최적값을 구해나가는 방법 중 하나이다</p> 
<p>결과물 제작 (문제점 개선사항)</p>	<p>1) 2-oo_design.ipynb 코드 분석 (객체지향 설계)</p> <p>1-1) add_to_class 함수의 용도와 어떻게 사용하는지에 대하여 설명 하시오</p> <pre>def add_to_class(Class): #@save """Register functions as methods in created class.""" def wrapper(obj): setattr(Class, obj.__name__, obj) return wrapper class A: def __init__(self): self.b = 1 @add_to_class(A) def do(self): print('Class attribute "b" is', self.b) a.do()</pre> <p>함수와 클래스를 연결시켜주는데에 사용된다.</p>

위 코드를 해석하자면 do() 함수를 add_to_class 함수를 이용하여 클래스 A에 연결을 시켜준다.

그렇게 되면 do 함수는 클래스 A에 있는 인스턴스 변수 b를 불러와 a.do()를 통해 b를 출력하게 된다.

1-2) HyperParameters 클래스, save_hyperparameters 함수의 용도와 어떻게 사용하는지에 대하여 설명하시오 (예제 코드 명시)

```
class B(d2l.HyperParameters):
    def __init__(self, a, b, c):
        self.save_hyperparameters(ignore=['c'])
        print('self.a =', self.a, 'self.b =', self.b)
        print('There is no self.c =', not hasattr(self, 'c'))
b = B(a=1, b=2, c=3)
```

HyperParameters 클래스는 여러 변수(파라미터)를 관리하는데에 사용된다.

save_hyperparameters 함수는 명시적인 저장을 하지 않고도 인스턴스 변수가 생성하여 값을 저장시키는데에 사용된다.

위 코드를 해석하자면 클래스 B는 HyperParameters 클래스를 상속받고, self.save_hyperparameters를 호출한다. b = B(a=1, b=2, c=3)를 통해 변수를 선언하고, 여기서 변수 a, b는 저장되 ignore=['c']를 이용해 변수 c는 무시하고 저장되지 않게 옵션을 넣어준다.

1-3) ProgressBoard 클래스의 용도와 어떻게 사용하는지에 대하여 설명하시오 (예제 코드 명시)

```
class ProgressBoard(d2l.HyperParameters): #@save
    """The board that plots data points in animation."""
    def __init__(self, xlabel=None, ylabel=None, xlim=None,
                 ylim=None, xscale='linear', yscale='linear',
                 ls=['-', '--', '-.', ':'], colors=['C0', 'C1', 'C2', 'C3'],
                 fig=None, axes=None, figsize=(3.5, 2.5), display=True):
        self.save_hyperparameters()
    def draw(self, x, y, label, every_n=1):
        raise NotImplemented

board = d2l.ProgressBoard('x')
for x in np.arange(0, 10, 0.1):
    board.draw(x, np.sin(x), 'sin', every_n=2)
    board.draw(x, np.cos(x), 'cos', every_n=10)
```

ProgressBoard는 학습(계산) 과정을 시각화 해주는 즉, 모델 그래프로 시각화할 때 사용된다.

위 코드에서는 먼저 클래스 ProgressBoard는 HyperParameters 클래스를 상속받고, x축과 y축의 레이블, x축과 y축의 범위, x축과 y축의 스케일의 인스턴스 변수(파라미터)를 생성하고 save_hyperparameters 함수를 호출해 전달받은 인스턴스 변수를 저장한다.

draw() 함수를 이용하여 계산의 과정을 그래프로 만들고

ProgressBoard 객체인 board를 생성하여 0에서 10까지 x의 값을 0.1간격으로 반복하며 sin, cos 함수의 그래프를 그린다. 그리고 every_n 변수값에 따라

특정 간격으로 그래프를 그린다.

1-4) 모델의 기본이 되는 Module클래스의 용도와 어떻게 사용하는지에 대하여 설명하시오 (예제 코드 명시)

```
class Module(nn.Block, d2l.HyperParameters): #@save
    """The base class of models."""
    def __init__(self, plot_train_per_epoch=2, plot_valid_per_epoch=1):
        super().__init__()
        self.save_hyperparameters()
        self.board = ProgressBoard()
    def loss(self, y_hat, y):
        raise NotImplementedError
    def forward(self, X):
        assert hasattr(self, 'net'), 'Neural network is defined'
        return self.net(X)

    def plot(self, key, value, train):
        """Plot a point in animation."""
        assert hasattr(self, 'trainer'), 'Trainer is not initied'
        self.board.xlabel = 'epoch'
        if train:
            x = self.trainer.train_batch_idx
            self.trainer.num_train_batches
            n = self.trainer.num_train_batches
            self.plot_train_per_epoch
        else:
            x = self.trainer.epoch + 1
            n = self.trainer.num_val_batches / \
            self.plot_valid_per_epoch
            self.board.draw(x, value.asnumpy(), ('train_' if train else 'val_') +
key, every_n=int(n))

    def training_step(self, batch):
        l = self.loss(self(*batch[:-1]), batch[-1])
        self.plot('loss', l, train=True)
        return l
    def validation_step(self, batch):
        l = self.loss(self(*batch[:-1]), batch[-1])
        self.plot('loss', l, train=False)
```

Module 클래스는 모든 모델의 기본이 되는 클래스이며 최소한 3개의 함수를 이용하여 사용된다.

위 코드에선 Block 클래스와 HyperParameters클래스를 상속받아 Module클래스를 사용하고, 최소 3개의 함수를 필요로 한다. 여기서 학습에 필요한 파라미터를 저장하기 위해 __init__ 함수와 손실함수를 계산하기 위해 loss함수를 사용, 배치 데이터를 입력받아 손실값을 출력하는 훈련단계의 training_step함수, 배치데이터를 입력받아 손실함수를 이용해 손실값을 그래프로 출력하는 검증단계의 validation_step함수, 전방전파 연산을 수행 하기위해 forward 함수를 이용하였다. 즉, 이 함수들의 기능을 이용해 모델의 학습

과 검증과정을 도와준다.

1-5) 모델의 기본이 되는 Module클래스가 가지고 있는 함수들에 대하여 입력, 동작, 출력에 대하여 설명하시오

가) `__init__` 함수

- 입력 : `plot_tarin_per_epoch`, `plot_valid_per_epoch` 변수를 입력
- 동작 : `save_hyperparameters()` 함수를 이용하여 변수를 저장, `ProgressBoard()` 객체를 생성해 `board` 변수에 저장
- 출력 : 반환 값이 없고 따로 출력이 없음[생성자 함수]

나) `loss` 함수

- 입력 : 예측값인 `y_hat`과 실제값 `y`를 입력
- 동작 : 구현되어있지 않음
- 출력 : 구현되어있지않아 결과 없음

다) `forward` 함수

- 입력 : 훈련 데이터(`X,Y`)의 특징데이터인 `X`를 입력
- 동작 : 훈련집합의 특징 `X`를 모델에 넣어 전방전과 연산을 수행하고 $y=(x)$ 의 출력값을 계산한다. `self` 객체에 `net` 속성이 있는지 확인후, 없으면 `AssertionError`를 발생시킴
- 출력 : `net` 속성을 이용해 입력데이터 `x`를 전달하고 그 결과를 반환함.

라) `plot` 함수

- 입력 : 시각화할 레이블[`key`], 시각화할 값[`value`], 훈련데이터인지, 검증데이터인지를 나타내는 값[`train`]을 입력
- 동작 : `self` 객체에 `trainer` 속성이 있는지 확인하고, 만약 없으면 `'Trainer is not inited'`라는 메시지와 함께 에러를 발생시키고, `X`축 레이블을 `epoch`로 설정함, 그리고 훈련데이터인지 검증데이터인지를 구분하여 나눈다. 여기서 훈련데이터인 경우 `X`축 좌표를 계산하고 배치의 위치를 비율로 나타냄. 만약 훈련데이터가 아닌 검증데이터인 경우 `epoch`값에 1을 더하고, 한 `epoch` 내에서 몇 개의 검증 배치마다 플롯을 갱신할지 결정함
- 출력 : 출력값은 없다.[그래프를 그리기만 함]

마) `training_step` 함수

- 입력 : `batch` 데이터를 입력
- 동작 : 1번수에 손실함수를 이용하여 오차를 계산하고 미분한 값을 저장한 후, 계산된 오차값을 훈련된 데이터로 명시하고, 그래프로 시각화한다.
- 출력 : 함수를 끝내고 미분값이 저장된 1결과값을 반환함.

바) `validation_step` 함수

- 입력 : `batch` 데이터를 입력
- 동작 : 1번수에 손실함수를 이용하여 오차를 계산하고 미분한 값을 저장한 후, 계산된 오차값을 훈련 데이터가 아님[검증 데이터임]을 명시하고, 그래프로 시각화 한다.
- 출력 : 반환 값이 없어 출력이 없음

사) `configure_optimizers` 함수

- 입력 : 입력값이 존재 하지 않음

- 동작 : 구현되어있지 않음
- 출력 : 구현되어 있지 않아 결과 없음

1-6) 데이터를 처리하는 기본 클래스인 DataModule의 용도와 어떻게 사용하는지에 대하여 설명하시오 (예제 코드 명시)

```
class DataModule(d2l.HyperParameters): #@save
    """The base class of data."""
    def __init__(self, root='../data', num_workers=4):
        self.save_hyperparameters()
    def get_dataloader(self, train): #미니배치, 서플링
        raise NotImplementedError
    def train_dataloader(self): #훈련 데이터 => 서플링 0 > 훈련 0 > 모델을 개선
        return self.get_dataloader(train=True) #훈련 0
        #iterator값을 넘겨줌
    def val_dataloader(self): #검증 데이터 => 서플링 X > 훈련 X > 오차계산
        return self.get_dataloader(train=False) #훈련 X
```

데이터를 다운로드하고 가공한 후에 배치데이터를 제공하는 방식으로 이용되며 데이터로딩, 배치 생성등을 간편하게 관리한다. train_dataloader

1-7) 데이터를 처리하는 기본 클래스인 DataModule이 가지고 있는 함수들에 대하여 입력, 동작, 출력에 대하여 설명하시오

가) __init__ 함수

- 입력 : 데이터가 저장될 기본 디렉터리 경로를 '../data'로 지정하고, 데이터 로딩에 사용할 작업자의 수 'num_workers'를 입력으로 한다.
- 동작 : 입력받은 하이퍼 파라미터를 저장한다.
- 출력 : 출력값 없음[생성자 함수]

나) get_dataloader 함수

- 입력 : 학습 데이터인지 판별하는 값[train]을 입력받음
- 동작 : 구현되어있지 않음
- 출력 : 출력값또한 구현되어있지 않음

다) train_dataloader 함수

- 입력 : 입력값 없음
- 동작 : "train=True" 패러미터를 전달하여 학습 데이터를 가져온다.
- 출력 : 학습 데이터로더 객체를 반환함

라) val_dataloader 함수

- 입력 : 입력값 없음
- 동작 : "train=False" 패러미터를 전달하여 검증 데이터를 가져온다.
- 출력 : 검증 데이터로더 객체를 반환함.

2) 3-synthetic_regression_data.ipynb 코드 분석 (회귀데이터 데이터생성)

2-1) 회귀 데이터를 생성하는 클래스인 SyntheticRegressionData의 용도와 어떻게 사용하는지에 대하여 설명하시오 (예제 코드 명시)

```
class SyntheticRegressionData(d2l.DataModule): #@save
    """Synthetic data for linear regression."""
    def __init__(self, w, b, noise=0.01, num_train=1000, num_val=1000,
                 batch_size=32):
```

```

super().__init__()
self.save_hyperparameters()
n = num_train + num_val
self.X = np.random.randn(n, len(w))
noise = np.random.randn(n, 1) * noise
self.y = np.dot(self.X, w.reshape((-1, 1))) + b + noise
data = SyntheticRegressionData(w=np.array([2, -3.4]), b=4.2)

@d2l.add_to_class(SyntheticRegressionData)
def get_dataloader(self, train):
    if train:
        indices = list(range(0, self.num_train))
        # The examples are read in random order
        random.shuffle(indices)
    else:
        indices = list(range(self.num_train, self.num_train+self.num_val))
    for i in range(0, len(indices), self.batch_size):
        batch_indices = np.array(indices[i: i+self.batch_size])
        yield self.X[batch_indices], self.y[batch_indices]
xyiter = iter(data.train_dataloader())
X, y = next(xyiter)
print('X shape:', X.shape, '\ny shape:', y.shape)
Xy = next(xyiter)
print('X:', Xy[0], '\ny:', Xy[1])

```

DataModule을 사용한 클래스이며, 회귀 분석 모델을 훈련하기 위해 인위적으로 생성된 데이터를 생성하는 클래스이다. 사용 방법은 `__init__` 함수에서 모델의 매개변수와 학습률, 훈련데이터와 검증데이터, 배치데이터의 크기를 지정하고, `get_dataloader` 함수를 통해 학습용, 검증용 데이터의 iterator을 반환하여 배치단위로 데이터를 불러온다.

2-2) 회귀 데이터를 생성하는 클래스인 SyntheticRegressionData이 가지고 있는 함수들에 대하여 입력, 동작, 출력에 대하여 설명하시오

가) `__init__` 함수

- 입력 : w, b[모델의 매개변수], noise[학습률], num_train, num_val[훈련 데이터와 검증 데이터의 개수], batch_size[배치데이터의 크기]를 입력
- 동작 : d2l.DataModule 클래스를 상속받고, self.save_hyperparameter()을 이용해 변수를 저장하고, 데이터의 총 샘플 수를 n변수에 저장한다. 그리고 n개의 샘플과 len(w)의 특징을 이용하여 임의의 난수들로 이루어진 행렬 X를 생성한다. n개의 샘플에 대한 임의의 난수를 생성하고, 주어진 noise값을 곱하여 최종 noise(최종 학습률)을 생성한다. 그리고 입력데이터 x와 모델의 매개변수 w를 dot연산을 하고, b와 noise값을 더하여 y값을 생성한다. 마지막으로 주어진 w와 b값을 이용하여 SyntheticRegressionData클래스의 인스턴스 변수를 생성함.
- 출력 : 출력 값은 없다.

나) `get_dataloader` 함수

- 입력 : train 패러미터로 선별된 훈련데이터 혹은 검증데이터를 입력으로 받음
- 동작 : 훈련데이터일 경우 indices라는 리스트를 생성하고 인덱스값을 0부터 999까지[num_train] 저장한다. 그리고 임의로 셔플을 진행하고, 만약 검증데

	<p>이터일 경우 indices라는 리스트를 생성해 리스트의 인덱스 값을 1000[num_train]부터 1999까지[num_train+num_val] 저장한다. 다음에는 i의 범위를 0부터 1000까지[indices] 32단위로 증가한다. 이때 batch_indices 배열을 생성하는데, 범위를 [i : i+32(batch_size)]까지로 설정한다. 마지막으로 X변수와 Y변수의 값을 batch_indices의 값인 32개만큼 받는다.</p> <ul style="list-style-type: none"> 출력 : 그리고 xyiter변수에 iterator을 이용하여 train_dataloader()값을 반환 하고 next()를 통해 받는 값을 X와 Y에 저장한다.
기대 효과 및 활용방안	<p>기계학습에 사용되는 프레임워크인 MxNET을 사용 함으로써 데이터를 준비하고, 모델을 설계하고, 최적화 학습을 하는 과정을 쉽게 설계하고, 기계학습이 어떻게 이루어 지는지를 이해할 수 있다. 가장 단순한 모델을 만들고 학습하는 방법을 기초로 해서 영상, 언어, 인터넷 데이터, 생성형 분야의 복잡한 모델과 학습데이터를 다룰 수 있는 기술을 키울수 있다.</p>
<p>2024 년 6 월 14 일</p> <p style="text-align: right;">지도교수 심 효 선 (인)</p>	