1. 선형회귀

w = torch.zeros(1, requires_grad = True): 가중치 선언

b = torch.zeros(1, requires_grad = True): 편향 선언

optimizer.zero_grad(): 기울기 0으로 초기화

loss.backward(): 손실 함수를 미분하여 기울기 계산

optimizer.step(): 가중치와 편향을 업데이트

epoch_num = n: 원하는 만큼 경사 하강법을 반복. 학습 진행 횟수

epoch: 전체 훈련 데이터가 학습에 한 번 사용된 주기

loss = torch.mean((hypothesis - y_train) ** 2): 평균 제곱 오차 선언

optimizer = optim.SGD([w, b], lr = 0.01): 경사하강법 구현

loss = f.mse_loss(prediction, y_train): 파이토치에서 제공하는 평균 제곱 오차 함수

2. 이진분류

* 파이토치의 nn.Linear 와 nn.Sigmoid로 로지스틱 회귀를 구현

```
model = nn.Sequential(
    nn.Linear(2, 1), # input_dim = 2, output_dim = 1
    nn.Sigmoid() # 출력은 시그모이드 함수를 거친다.
)
```

3. 퍼셉트론

```
# GPU가 사용가능한 여부 파악 test code
device = 'cuda' if torch.cuda.is_available() else 'cpu'

torch.manual_seed(777)

if device == 'cuda' :
    torch.cuda.manual_seed_all(777)

# 이제 다층 퍼셉트론을 설계합니다. 아래는 입력층 , 은닉층1 은닉층2 은닉층3 출력층 을 가지는 은닉층이 3개인 인공 신경망
```

```
# GPU
model = nn. Sequential(
    nn.Linear(2, 10, bias = True), # input_layer = 2 hidden_layer1 = 10
    nn.Sigmoid(),
    nn.Linear(10, 10, bias = True), # hidden_layer1 = 10 hidden_layer2 = 10
    nn.Sigmoid(),
    nn.Linear(10, 10, bias = True), # hidden_layer2 = 10 hidden_layer3 = 10
    nn.Sigmoid(),
    nn.Linear(10, 1, bias = True), # hidden_layer3 output_layer = 1
    # 우리가 사용할 Loss가 BCELoss 이므로 마지막 레이어를 시그모이드 함수를 적용
    nn.Sigmoid(),
).to(device)
```

4. 합성곱 신경망

합성곱층과 풀링 선언하기

```
# 첫번째 합성곱 층 구현. 1채널 짜리를 입력받아서 32채널을 뽑아내는데 커널 사이즈는 3이고 패딩은 1
conv1 = nn.conv2d(1, 32, 3, padding=1)
print(conv1)
conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
# 두번째 합성곱 층 구현. 32채널 짜리를 입력받아서 64채널을 뽑아내는데 커널 사이즈는 3이고 패딩은 1
conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
```

```
print(conv2)
conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
# 이제 맥스풀링 구현. 정수 하나를 인자로 넣으면 커널 사이즈와 스트라이드가 둘 다 해당값으로 지정됨
pool = nn.maxPool2d(2)
print(pool)
maxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

# 배치정규화 크기를 32로 적용
bn1 = torch.nn.BatchNorm1d(32)
bn2 = torch.nn.BatchNorm1d(32)

5. 객체 탐지 모델 분류기법

# transform은 이미지 데이터를 변형시키기 위한 함수
# torchvision의 데이터 출력은 0에서1의 범위이기 때문에 -1에서 1의 범위로 정규화된 텐서로 변환
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```