제 포트폴리오는 이론과 예제 코드들로 구성되어 있습니다.
교수님께서 주신 PPT 자료와 직접 구매한 딥러닝 도서, 인터넷 블로그를 참고하여 작성하였습니다.
처음에는 가장 기본인 머신러닝과 딥러닝에 대해 다루었습니다.
AI의 시작 배경과 시대에 따른 관심도, 머신 러닝과 딥러닝의 이론적인 부분을 작성했습니다.
머신러닝은 크게 지도학습과 비지도 학습, 강화 학습으로 나뉘며, 딥러닝은 신경망을 사용해 구축하므로 사람이 개입할 필요가 없습니다.
다음으로는 텐서플로입니다.
여기서 가장 중요시했던 부분은 브로드캐스팅입니다.
가지고 있는 값을 이용하여 shape를 맞추는 작업을 브로드캐스팅이라고 하는데, 처음 배우는 개념이고, 코드를 보고 그림으로 바꾸지 않으면 계산이 잘 안되서 더 집중해서 다뤘던 파트입니다.
다음은 제 포트폴리오에서 가장 큰 범위를 차지한 MNIST입니다.
x_train에는 총 60000개의 28×28 크기의 이미지가 담겨 있으며, y_train에는 이 x_train의 60000개에 대한 0에서 9까지의 값이 담겨 있는 레이블 데이터셋입니다.
그리고 x_train과 y_train은 각각 10000개의 이미지와 레이블 데이터셋입니다.
먼저 x_train와 y_train을 통해 모델을 학습하고 난 뒤에, x_test, y_test를 이용해 학습된 모델의 정확도를 평가하게 됩니다.

```python
import tensorflow as tf

# mnist 모듈 준비
mnist = tf.keras.datasets.mnist

# MNIST 데이터셋을 훈련과 테스트 데이터로 로드하여 준비
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 샘플 값을 정수(0~255)에서 부동소수(0~1)로 변환
x_train, x_test = x_train / 255.0, x_test / 255.0

# 층을 차례대로 쌓아 tf.keras.Sequential 모델을 생성
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 선택
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 모델 요약 표시
model.summary()

# 모델을 훈련 데이터로 총 5번 훈련
model.fit(x_train, y_train, epochs=5)

# 모델을 테스트 데이터로 평가
model.evaluate(x_test, y_test)
```

다음 코드는 신경망 모델입니다.
총 4개의 레이어로 구성된 신경망인데, 첫 번째 레이어는 입력 이미지의 크기가 28×28이므로 이를 1차원 텐서로 펼치는 것이고, 두 번째 레이어는 첫 번째 레이어에서 제공되는 784개의 값을 입력받아 128개의 값으로 인코딩해 주는데, 활성함수로 ReLU를 사용하도록 했습니다.
두번째 레이어의 실제 연산은 첫 번째 레이어에서 제공받은 784개의 값을 784×128 행렬과 곱하고 편향값을 더하여 얻은 128개의 출력값을 다시 ReLU 함수에 입력해 얻은 128개의 출력입니다.
세 번째는 128개의 뉴런 중 무작위로 0.2가 의미하는 20%를 다음 레이어의 입력에서 무시합니다.
이렇게 20% 정도가 무시된 값이 4번째 레이어에 입력되어 총 10개의 값을 출력하는데, 여기서 사용되는 활성화 함수는 Softmax가 사용되었습니다.
Softmax는 마지막 레이어의 결과값을 다중 분류를 위한 확률값으로 해석할 수 있도록 하기 위함입니다.
10개의 값을 출력하는 이유는 입력 이미지가 0~9까지의 어떤 숫자를 의미하는지에 대한 각각의 확률을 얻고자 함입니다.
이렇게 정의된 모델을 학습하기에 앞서 model.compile로 컴파일합니다.
모델의 학습 중에 역전파를 통한 가중치 최적화를 위한 기울기 방향에 대한 경사하강을 위한 방법으로 Adam을 사용했으며 손실 함수로 다중 분류의 Cross Entropy

Error인 'sparse_categorical_crossentropy'를 지정하였습니다.
그리고 모델 평가를 위한 평가 지표로 'accuracy'를 지정하였습니다.
summray로 모델을 요약하고, 이제 fit을 통해 모델을 학습할 수 있게 되는데, 학습에 사용되는 데이터셋과 학습 반복수로 5 Epoch을 지정했습니다.
Epoch은 전체 데이터셋에 대해서 한번 학습할 때의 단위입니다.
학습이 완료되면 평가를 위한 데이터셋을 지정하고, 평가가 끝나면 평가 데이터셋에 대한 손실값과 정확도가 결과로 표시되게 됩니다.
다음은 퍼셉트론과 활성화 함수에 대한 이야기가 나오는데 여기보다는 게이트에 대해 설명하도록 하겠습니다.
논리 게이트는 AND와 OR, XOR이 있는데 AND 게이트는 둘다 1인 경우에만 1인 것을 말하고, OR 게이트는 둘 중 하나라도 1이면 1, XOR 게이트는 둘 중 하나가 1인 경우에만 1인 베타적인 논리 연산을 말합니다.
하지만 여기서 문제가 발생하는데, 퍼셉트론으로 AND와 OR은 구현이 가능하지만 XOR은 퍼셉트론으로 구현이 불가능합니다.
그래서 이를 해결하기 위해 여러 개의 퍼셉트론을 층으로 쌓아 만든 다중 퍼셉트론을 사용하게 되는데, 이렇게 하면 단층 퍼셉트론으로 표현하지 못하는 것을 구현할 수 있게 됩니다.

```
import tensorflow as tf
import numpy as np
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[0], [1], [1], [0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=2, activation='sigmoid', input_shape=(2,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.3), loss='mse')
model.summary()
```

그래서 코드를 보면 Dense층 두 개를 쓴 것을 볼 수 있습니다.
다음은 회귀와 분류입니다.
회귀는 연속적인 값을 예측하는 것이고, 분류는 불연속적인 값을 예측하는 것을 의미합니다.
회귀를 분석하는 방법으로는 단순 선형 회귀 분석과 다중 선형 회귀 분석이 있는데, 단순 선형 회귀 분석이란 가장 적합한 일차 함수식을 추정하는 것을 말합니다.
상관 분석에서는 두 변수 사이에 차별이 없지만, 회귀 분석에서는 두 변수 사이에 독립과 종속의 관계가 성립되게 됩니다.
따라서 가장 적합하게 추정된 일차 함수식은 독립변수와 종속변수로 표현되며,
y=a + b * x 형태로 나타나게 됩니다.
단순 선형 회귀 분석의 과정은 먼저 회귀 직선의 회귀 계수를 추정하여 모집단에서 유의한지 그렇지 않은지 판단하고, 유의하다면 얼만큼 실제 값들을 설명해줄 수 있

는지 적합도를 봅니다.
이에 반해 다중 선형 회귀 분석은 여러 개의 독립 변수들을 가지고 종속변수를 예측하기 위한 것입니다.
단순 선형 회귀 분석의 식이 y = a + b * x의 형태로 나타내어 지는 반면, 다중 선형 회귀 분석의 식은 y = a+ b * 2 형태로 나타내집니다.
다중 선형 회귀 분석은 어떤 설명 변수가 종속 변수에 영향을 미치는지를 파악할 수 있게 해주고, 이 설명 변수를 사용해 종속 변수를 가능한 정확히 예측할 수 있게 해주기 때문에 매우 유용한 분석 방법입니다.
그 과정은 먼저 회귀 모형에 포함될 설명변수들을 선택하고, 이 회귀 모형에서 편회귀 계수 추정치를 구한 후 모집단에서 유의한지 그렇지 않은지 판단하고, 유의하다면 얼만큼 실제 값들을 설명해줄 수 있는지 적합도를 봅니다.
다음 손실 함수와 최적화 과정은 시간 관계상 넘어가고, part 6에서 일찍 멈춤 기능에 대해 말하면 딥러닝을 비롯한 머신 러닝 모델의 한 가지 중요한 딜레마는 너무 많은 Epoch은 오버피팅을 일으키고, 너무 적은 Epoch은 언더피팅을 일으킨다는 것입니다.
Epoch을 정하는데 많이 사용되는 Early stopping은 무조건 Epoch을 많이 돌린 후, 특정 시점에서 멈추는 것을 말합니다.
그 특정 시점을 어떻게 정하느냐가 Early stopping의 핵심이라고 할 수 있는데, 일반적으로 hold-out validation set에서의 성능이 더 이상 증가하지 않을 때 학습을 중지시키게 됩니다.
시간이 부족할 듯 하여 마지막 part7은 넘어가도록 하겠습니다.
이 포트폴리오를 준비하며 수업 자료 뿐만 아니라 블로그 등을 통해 다양한 자료들을 조사하며 좀 더 심화된 공부를 할 수 있었습니다.
자료를 보기만 하는 것이 아닌 새롭게 알게 된 것들을 적고, 이해하는 과정이 단순 포트폴리오를 넘어 후에 관련된 내용을 다루게 될 때에 많은 도움이 될 수 있을 것이라고 생각됩니다.
이상 기호정이었습니다. 감사합니다.