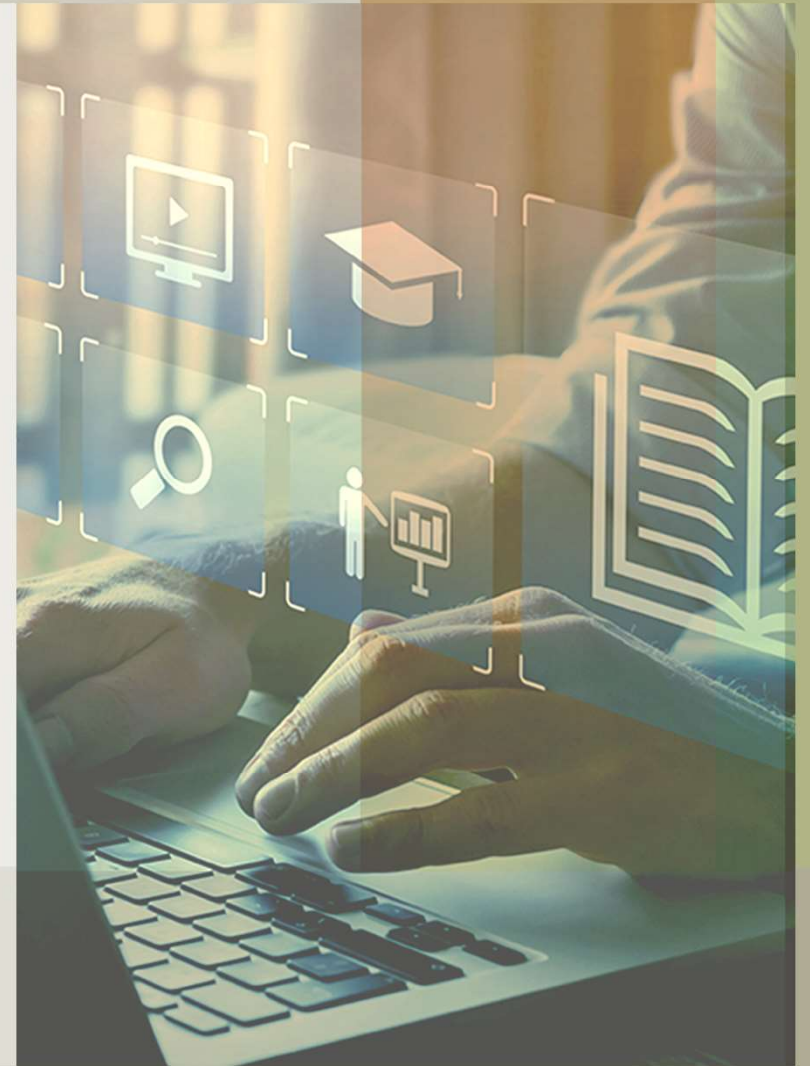


03

딥러닝

딥러닝 프레임워크

방송대 컴퓨터과학과 이병래 교수



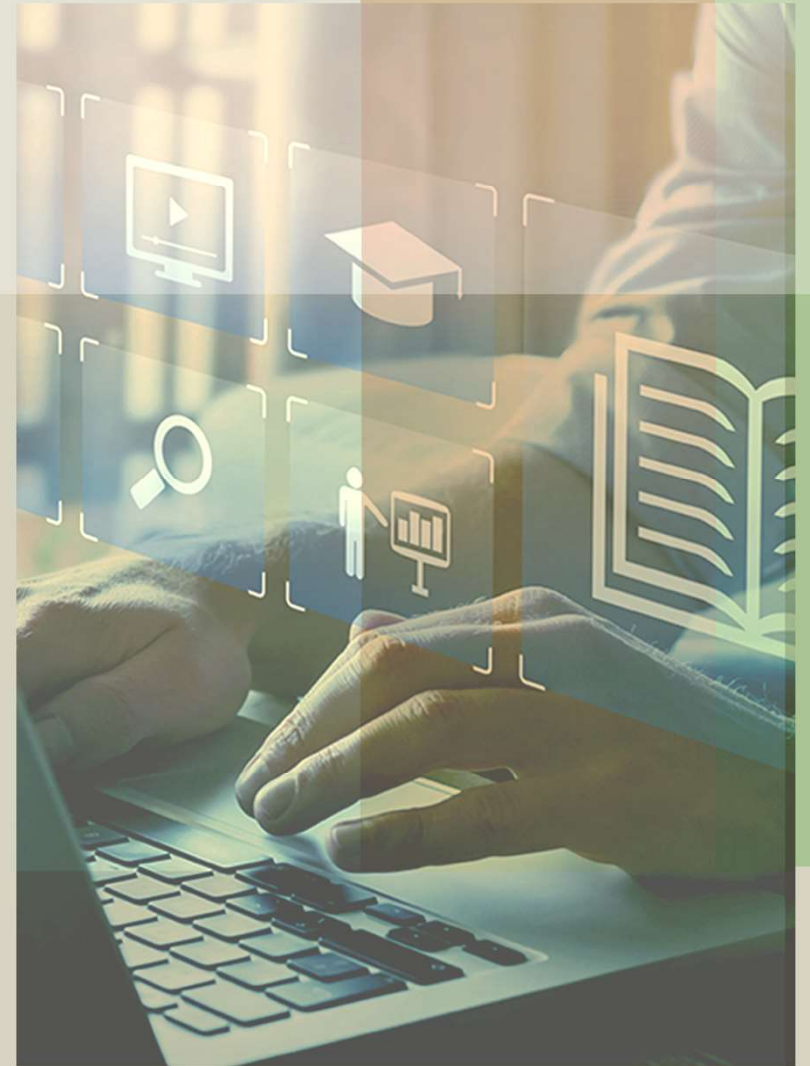
학습목차

- 1 딥러닝 프레임워크와 텐서플로
- 2 텐서
- 3 자동 미분
- 4 Keras를 이용한 모델의 구현



01

딥러닝 프레임워크와 텐서플로



1. 딥러닝 프레임워크

● 딥러닝 프레임워크란?

- 딥러닝을 위해 설계된 라이브러리, 기본적인 데이터 집합, 미리 구성된 네트워크 및 기타 유용한 도구를 제공하는 소프트웨어
- 텐서플로(TensorFlow), 파이토치(PyTorch), 카페(Caffe), MXNet 등

● 딥러닝 프레임워크가 제공하는 주요 기능

- 자동 미분
- 기본적인 신경망 구성 요소
- 손실함수 및 최적화 도구
- 계산 자원 활용
- 모델의 저장 및 로드



2. 텐서플로

● 텐서플로(TensorFlow)란?

- 2015년에 구글 브레인 팀(Google Brain Team)에서 구글 내부의 연구 및 제품 개발을 위해 만든 심층 신경망을 위한 오픈소스 딥러닝 프레임워크
- 2019년 9월에 텐서플로 2.0을 발표함
- 라이선스 : 아파치 라이선스 2.0
- 파이썬 언어를 위한 API 제공
 - C++ , Java 등의 언어를 위한 API도 제공함
- 텐서플로 2.0부터 Keras라는 고수준 신경망 API를 통합

 <https://www.tensorflow.org>



3. 텐서플로의 실행 모드

● 지연 실행(lazy execution) 모드

정적 계산 그래프 형태로
모델을 구성



세션 생성 후 데이터를
전달하여 계산을 실행

텐서플로 1.x 버전의 실행 모드



3. 텐서플로의 실행 모드

● 지연 실행(lazy execution) 모드

3-1 [1] 지연 실행 모드의 계산 그래프 구성

```
1 import tensorflow.compat.v1 as tf
2 tf.disable_v2_behavior()
3
4 # 계산 그래프 정의
5 a = tf.placeholder(tf.float32)
6 b = tf.placeholder(tf.float32)
7 c = a + b
8 print('c =', c)
```

➡ 출력 : c = Tensor("add:0", dtype=float32)



3. 텐서플로의 실행 모드

● 지연 실행(lazy execution) 모드

[2] 계산 그래프의 실행

```

1  # 세션을 생성하여 그래프를 실행함
2  with tf.Session() as sess:
3      # 'c'의 연산을 하는 그래프를 실행함
4      result = sess.run(c, {a:2., b:3.})
5  print('result =', result)
    
```

➡ 출력 : result = 5.0



3. 텐서플로의 실행 모드

● 즉시 실행(eager execution) 모드

- 일반 파이썬 코드처럼 계산 작업이 호출 즉시 실행됨

텐서플로 2.0 버전부터 도입된 실행 모드



장점

- 자연 실행 모드에 비해 직관적인 코드를 작성할 수 있음
- 모델의 테스트가 용이함

VS.



단점

- 명령 단위로 텐서플로 연산을 실행하여 결과를 파이썬으로 가져오는 과정을 반복하므로 프로그램 실행이 느림

➡ 그래프 실행(graph execution) 모드



3. 텐서플로의 실행 모드

● 즉시 실행(eager execution) 모드

3-2 [1] 즉시 실행 모드의 프로그램

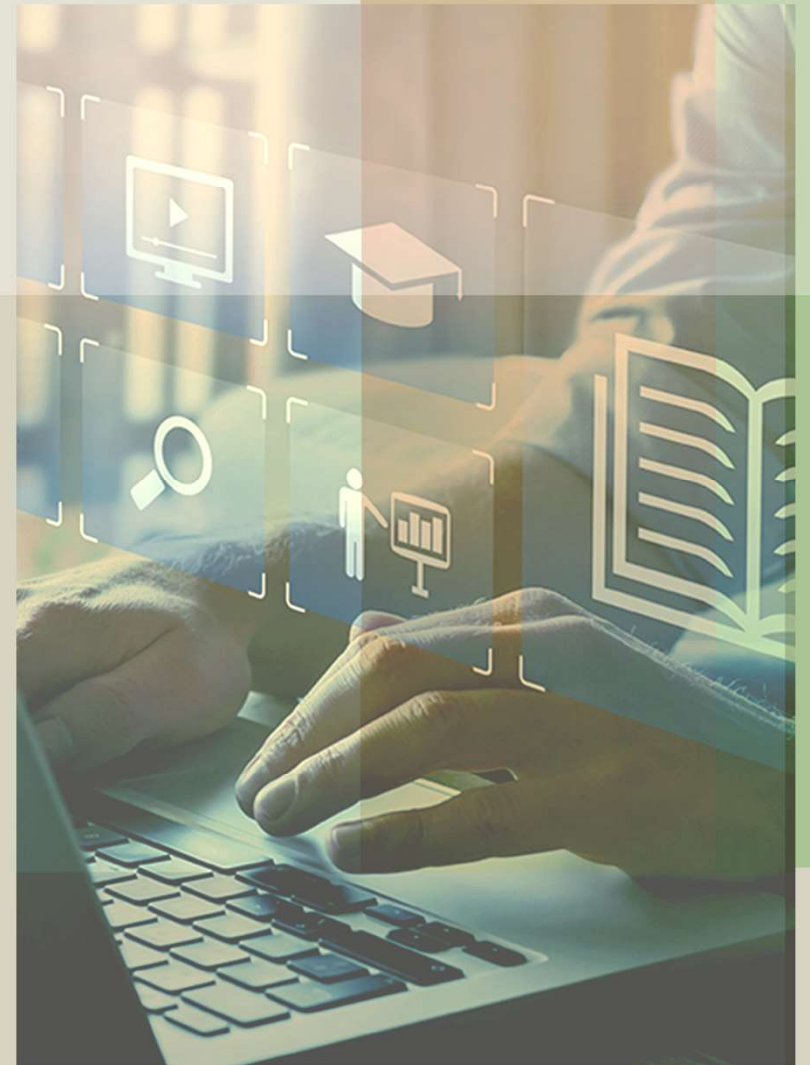
```
1 import tensorflow as tf
2
3 a = tf.constant(2.)
4 b = tf.constant(3.)
5 c = a + b
6 print('c =', c)
```

➡ **출력** : c = tf.Tensor(5.0, shape=(), dtype=float32)



02

텐서



1. 텐서의 개념

○ 텐서(tensor)

- 동일 자료형의 데이터를 저장하는 다차원 배열
- numpy의 배열과 유사한 형태로 데이터를 저장하는 객체

rank-0 텐서
(스칼라)

5

shape: []

rank-1 텐서
(벡터)

5	-2	3
---	----	---

shape: [3]

rank-2 텐서
(행렬)

1	2	3
4	5	6

shape: [2, 3]

rank-3 텐서
(다차원 배열)

			10	20	21
			13	14	15
	7	8	9		
1	2	3			
4	5	6	12	18	24

shape: [4, 2, 3]



1. 텐서의 개념

● 텐서플로에서 텐서의 활용

- 입력 데이터 정의
- 모델 파라미터 정의 : 가중치, 바이어스 등
- 계산 수행
 - 행렬 곱셈, 합성곱 및 활성화함수와 같은 수학적 연산에 활용
 - 텐서에 대한 다양한 연산 제공
- 중간 결과 저장
 - 모델 계산 중에 중간 결과를 저장하는 데 사용
- 손실함수 정의



2. 텐서플로의 텐서 사용

● 상수 텐서

- 값을 수정할 수 없는 텐서
- `tf.constant` 함수를 사용하여 만듦

● 변수 텐서

- 프로그램이 동작하는 동안 여러 가지 연산을 통해 값이 변화할 수 있는 상태를 표현하기 위한 텐서
- `tf.Variable` 클래스의 인스턴스를 생성
- 변수 텐서에 다른 값을 저장할 때는 `assign` 메소드를 사용
 - `assign`, `assign_add`, `assign_sub`



2. 텐서플로의 텐서 사용

3-3 [1] 필요한 패키지 불러오기

```
1 import tensorflow as tf
2 import numpy as np
```

3-3 [2] 상수 텐서

```
1 a = tf.constant(10.)
2 b = tf.constant([1, 2, 3, 4])
3 c = tf.constant([[[[1, 2], [3, 4], [5, 6]],
4                  [[7, 8], [9, 10], [11, 12]],
5                  [[13, 14], [15, 16], [17, 18]],
6                  [[19, 20], [21, 22], [23, 24]]], dtype=tf.float32)
7 print('a: dtype =', a.dtype, '\n', a)
8 print('b: shape =', b.shape, '\n', b)
9 print('c: device =', c.device)
```

2. 텐서플로의 텐서 사용

3-3 [3] 변수 텐서

```
1 x = tf.Variable(10.)
2 y = tf.Variable([[1., 2., 3.], [4., 5., 6.]])
3 z = np.array([[1., 3.], [2., 4.], [3., 5.]], dtype=np.float32)
4 print('x: dtype =', x.dtype, '\n', x)
5 print('y: shape =', y.shape, '\n', y)
6 print('y: device =', y.device)
```

3-3 [4] 변수 텐서에 값을 대입하기

```
1 x.assign_add(20.)
2 print('x = ', x.numpy())
```



2. 텐서플로의 텐서 사용

3-3 [5] 텐서에 대한 산술 연산 및 수학 함수

```
1 print('a * b =', (a * tf.cast(b, tf.float32)).numpy())
2 print('tf.math.exp(y) =', tf.exp(y))
3 print('tf.math.reduce_sum(c, axis=2) =', tf.reduce_sum(c, axis=2))
```

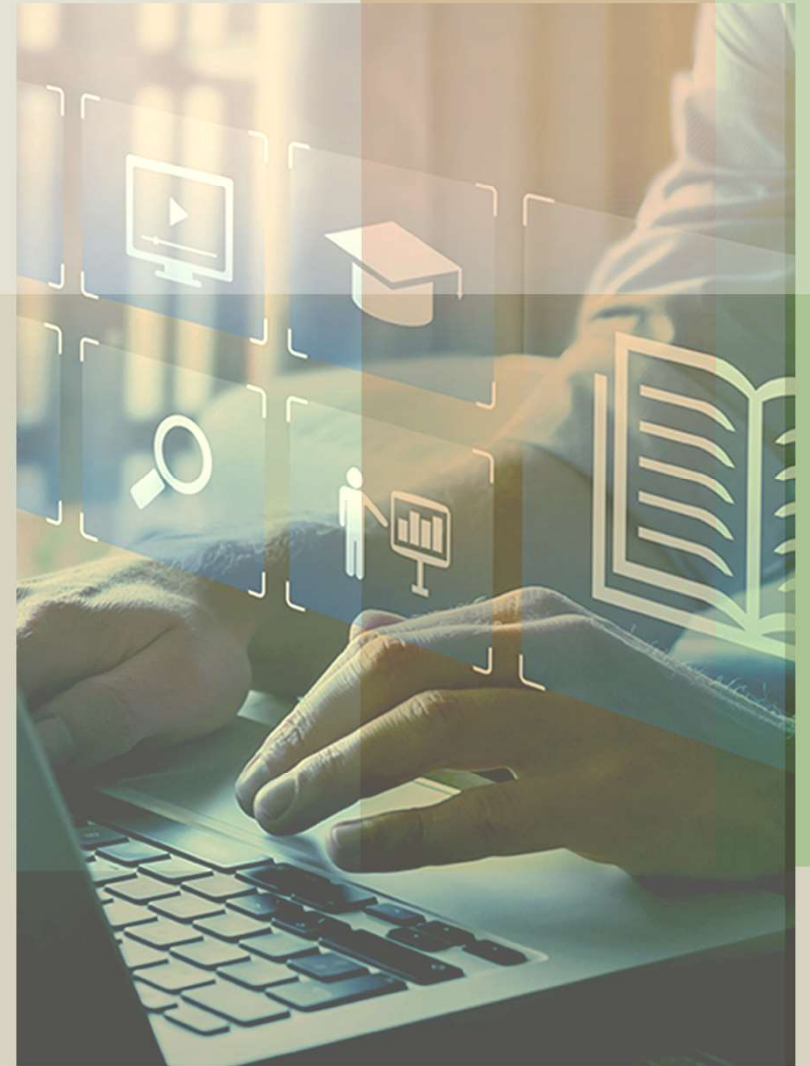
3-3 [6] 선형대수 연산 및 묵시적 형 변환

```
1 print('tf.linalg.matmul(y, z) =')
2 print(tf.matmul(y, z))
3 print('np.matmul(y, z) =')
4 print(np.matmul(y, z))
```



03

자동 미분



1. 텐서플로의 자동 미분

● tf.GradientTape API를 이용한 자동 미분

- tf.GradientTape 문맥 생성
- 정방향 진행(forward pass)의 연산을 '테이프'에 기록
 - 테이프의 watch 메소드로 추적할 텐서를 지정
 - 📝 변수 텐서(trainable 속성의 디폴트 값이 True임)는 기본적으로 추적 대상이므로 별도로 추적 대상으로 지정할 필요 없음
- '테이프'를 '되감기' 하며 미분 계산
 - tf.GradientTape의 gradient 메소드 사용




2. 자동 미분 계산 예

- $y = (x_1 + 2x_2)^2$ 의 편미분 $\partial y / \partial x_1, \partial y / \partial x_2$ 계산($x_1 = 3, x_2 = 1$)

3-4 [1] 자동 미분

```

1 import tensorflow as tf
2
3 x1 = tf.Variable(3.)
4 x2 = tf.Variable(1., trainable=False)
5 with tf.GradientTape() as t:
6     t.watch(x2)
7     y = (x1 + 2 * x2) ** 2
8 dy_dx = t.gradient(y, [x1, x2])
9 print(f'dy/dx1 = {dy_dx[0]}')
10 print(f'dy/dx2 = {dy_dx[1]}')
```

 $dy/dx1 = 10.0$
 $dy/dx2 = 20.0$



3. 자동 미분을 이용한 선형 회귀의 학습

● 선형 회귀 문제

$$\hat{y} = wx + b$$

- x 는 독립변수, y 는 종속변수
- 학습표본 집합을 바탕으로 w 와 b 를 학습하여 y 를 예측한 \hat{y} 를 구함

📝 손실함수 : 오차 제곱

$$E(\hat{y}, y) = (wx + b - y)^2$$

📝 경사 하강법을 이용한 w 와 b 의 학습

$$w(t+1) = w(t) - \eta \frac{\partial E(\hat{y}, y)}{\partial w}$$

$$b(t+1) = b(t) - \eta \frac{\partial E(\hat{y}, y)}{\partial b}$$



3. 자동 미분을 이용한 선형 회귀의 학습

3-5 [1] 필요한 패키지 불러오기

```
1 import tensorflow as tf
2 import numpy as np
```

3-5 [2] 학습표본 집합 및 가중치와 바이어스 등 정의

```
1 x = tf.constant([1., 3., 5., 7.])
2 y = tf.constant([2., 3., 4., 5.])
3 w = tf.Variable(1.)
4 b = tf.Variable(0.5)
5 learning_rate = 0.01
6 epochs = 1000
```



3. 자동 미분을 이용한 선형 회귀의 학습

3-5 [2] 학습표본 집합 및 가중치와 바이어스 등 정의

```
1 x = tf.constant([1., 3., 5., 7.])
2 y = tf.constant([2., 3., 4., 5.])
3 w = tf.Variable(1.)
4 b = tf.Variable(0.5)
5 learning_rate = 0.01
6 epochs = 1000
```

3-5 [3] 학습 단계의 처리 함수 정의

```
1 def train_step(x, y):
2     with tf.GradientTape() as t:
3         y_hat = w * x + b
4         loss = (y_hat - y) ** 2
5         grads = t.gradient(loss, [w, b])
6         w.assign_sub(learning_rate * grads[0])
7         b.assign_sub(learning_rate * grads[1])
```

$$w(t+1) = w(t) - \eta \frac{\partial E(\hat{y}, y)}{\partial w}$$

$$b(t+1) = b(t) - \eta \frac{\partial E(\hat{y}, y)}{\partial b}$$



3. 자동 미분을 이용한 선형 회귀의 학습

3-5 [4] 학습표본 집합에 대한 반복 학습

```
1 for i in range(epochs):
2     for k in range(len(y)):
3         train_step(x[k], y[k])
```

3-5 [5] 학습된 파라미터 출력

```
1 print('w: {:.8.5f}    b: {:.8.5f}'.format(w.numpy(), b.numpy()))
```

➡ w: 0.50000 b: 1.50000



3. 자동 미분을 이용한 선형 회귀의 학습

3-5 [6] 학습된 파라미터를 이용한 모델 실행

```
1 f = 'x:{:8.5f} --> y:{:8.5f}'  
2 for k in range(len(y)):  
3     y_hat = w * x[k] + b  
4     print(f.format(x[k].numpy(), y_hat.numpy()))
```

x: 1.00000 --> y: 2.00000



x: 3.00000 --> y: 3.00000

x: 5.00000 --> y: 4.00000

x: 7.00000 --> y: 5.00000



4. 그래프 실행 모드의 활용

● 그래프 실행(graph execution) 모드

- 텐서플로 그래프(tf.Graph)를 실행하는 방식

 tf.Graph : 데이터 흐름 형태로 표현되는 계산 구조

● 텐서플로 그래프 실행의 장점

- 사용 가능한 하드웨어에서 효율적으로 실행되도록 최적화하여 컴파일되므로 빠른 계산을 할 수 있음
- 병렬처리 활용
- 파이썬 인터프리터가 없는 장치에서도 사용할 수 있음



4. 그래프 실행 모드의 활용

● 그래프 실행 모드의 프로그램 작성

- `tf.function`: 즉시 실행 모드의 프로그램을 작성하듯 텐서플로 기반의 코드를 작성하면 이를 자동적으로 그래프로 변환

`tf.function`을 직접 호출

- 일반적인 파이썬 함수를 인수로 전달하여 `tf.function`을 호출
- 파이썬이 호출할 수 있는 함수로 변환하여 돌려주면 이 함수를 원래 함수를 사용하듯 사용

`tf.function`을 수식어로 사용

- 그래프 실행 모드로 사용하려는 함수를 '@`tf.function`'으로 수식함



변환된 함수는 동일한 형태의 입력에 대해서는 첫 호출 시 그래프로 변환 후 실행되고, 그 이후에는 이미 변환된 그래프를 실행함



4. 그래프 실행 모드의 활용

◉ tf.function을 직접 호출하는 방법

3-5a [3] 학습 단계의 처리 함수 정의

```
1 def train_step(x, y):
2     with tf.GradientTape() as t:
3         y_hat = w * x + b
4         loss = (y_hat - y) ** 2
5         grads = t.gradient(loss, [w, b])
6         w.assign_sub(learning_rate * grads[0])
7         b.assign_sub(learning_rate * grads[1])
```

3-5a [4] 학습표본 집합에 대한 반복 학습

```
1 train_step_graph = tf.function(train_step)
2 for i in range(epochs):
3     for k in range(len(y)):
4         train_step_graph(x[k], y[k])
```



4. 그래프 실행 모드의 활용

● tf.function을 수식으로 사용하는 방법

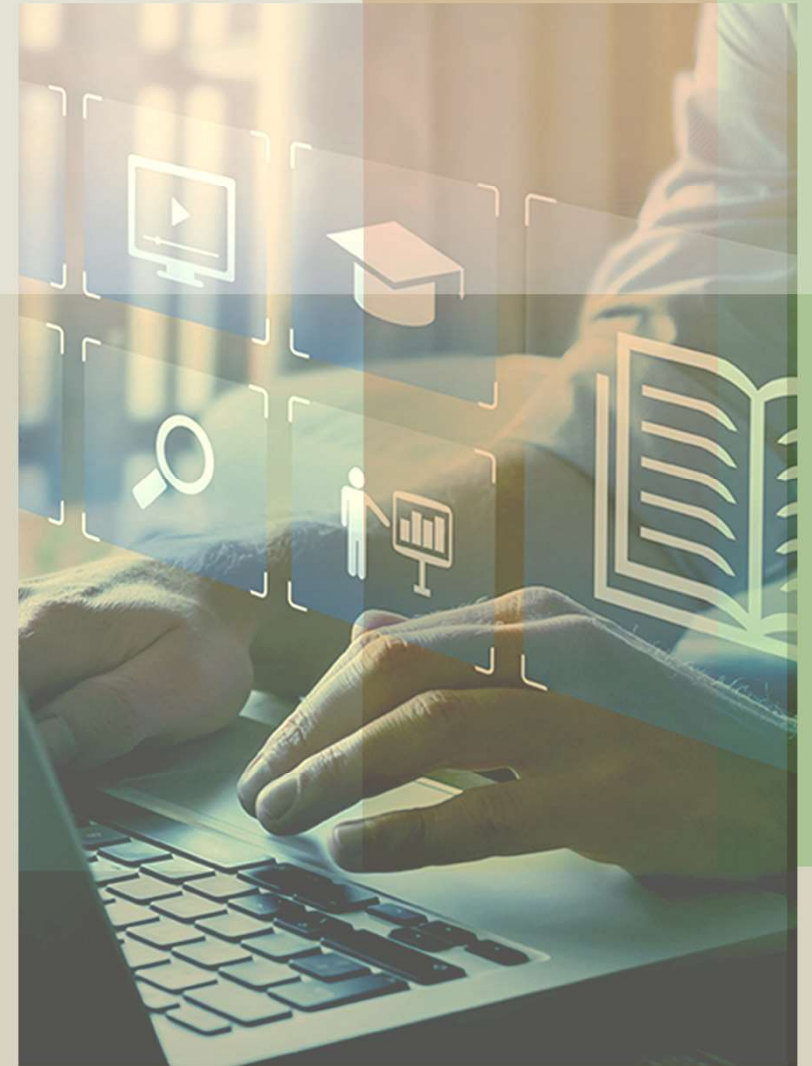
3-5b [3] 학습 단계의 처리 함수 정의

```
1 @tf.function
2 def train_step(x, y):
3     with tf.GradientTape() as t:
4         y_hat = w * x + b
5         loss = (y_hat - y) ** 2
6         grads = t.gradient(loss, [w, b])
7         w.assign_sub(learning_rate * grads[0])
8         b.assign_sub(learning_rate * grads[1])
```



04

Keras를 이용한 모델의 구현



1. Keras의 개요

● Keras란?

- 신경망 구현을 위한 고수준 오픈소스 라이브러리
- 2.4 버전부터는 텐서플로만 지원
- 텐서플로의 `tf.keras` 모듈에 제공됨
 - 여러 가지 신경망 층, 활성화함수, 손실함수, 최적화기 등을 제공

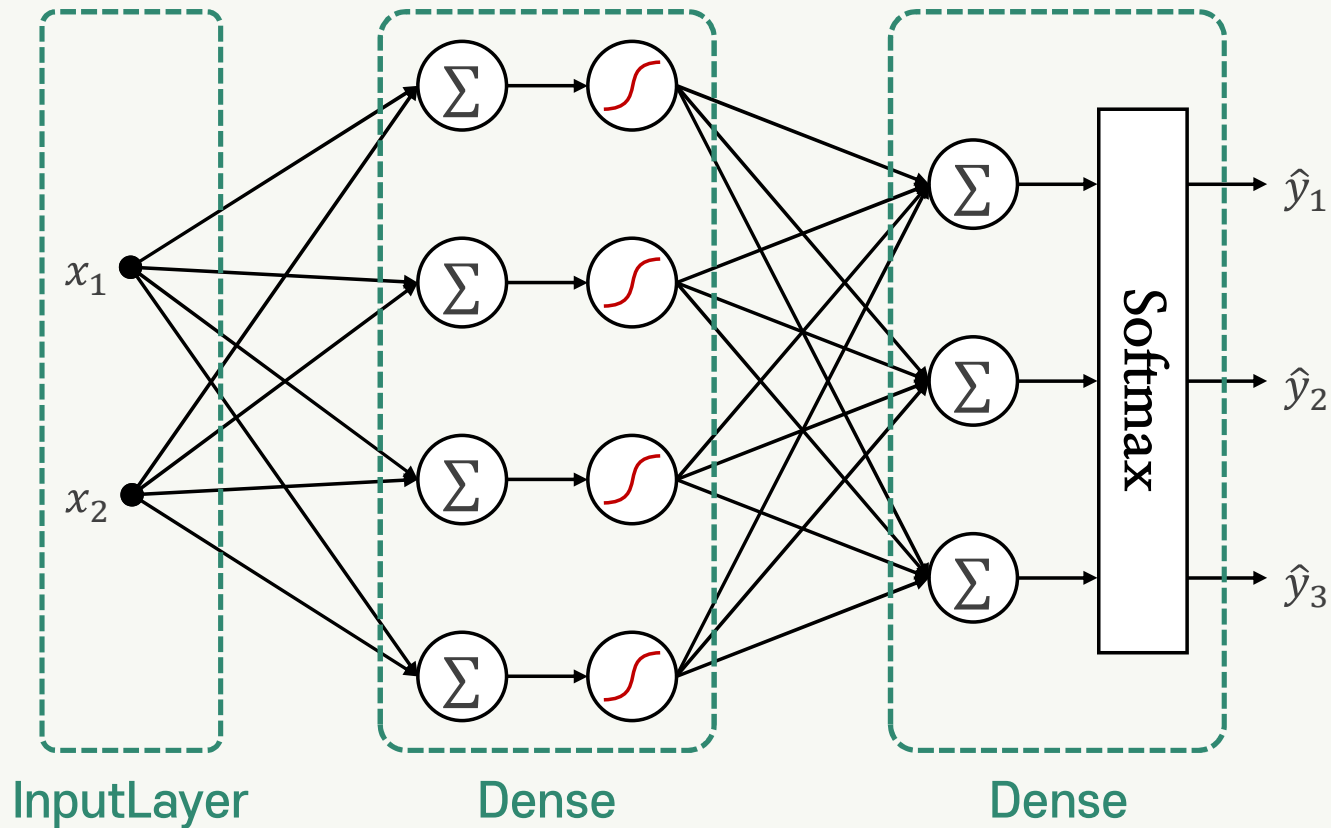
● Keras를 이용하여 신경망 모델을 구성하는 방법

- `tf.keras.Sequential` 클래스를 이용한 순차적으로 연결된 모델 구성
- 함수형(functional) API를 이용한 모델 구성
- 서브클래싱(subclassing) API를 이용한 모델 구성



2. 실습 : 텐서플로를 이용한 다층 퍼셉트론

● 구현할 다층 퍼셉트론



2. 실습 : 텐서플로를 이용한 다층 퍼셉트론

● 사용할 Keras 모듈

```
from tensorflow import keras  
from tensorflow.keras import layers, optimizers, losses
```

- keras : 텐서플로에 제공되는 Keras 모듈
- layers : 모델 구성에 필요한 여러 가지 층 클래스를 제공하는 모듈
- optimizers : 다양한 최적화기 클래스를 제공하는 모듈
- losses : 다양한 손실 함수 클래스를 제공하는 모듈



2. 실습 : 텐서플로를 이용한 다층 퍼셉트론

● tf.keras.Sequential 클래스를 이용한 모델 구성

■ tf.keras.Sequential 클래스

```
tf.keras.Sequential(layers=None, name=None)
```

• 순차적으로 층이 연결된 모델을 구성하는 클래스

- layers : 모델에 연결할 층의 리스트
- name : 생성된 모델의 이름

예 bp_model_tf = tf.keras.Sequential()



2. 실습 : 텐서플로를 이용한 다층 퍼셉트론

● tf.keras.Sequential 클래스를 이용한 모델 구성

■ tf.keras.layers.InputLayer 클래스를 이용한 입력층 구성

```
tf.keras.layers.InputLayer(input_shape=None, name=None)
```

• 네트워크의 진입점으로 사용할 층을 구성하기 위한 클래스

- input_shape : 입력의 형태 지정
- name : 생성된 입력층의 이름

예 in_layer = tf.keras.layers.InputLayer(input_shape=(2,))



2. 실습: 텐서플로를 이용한 다층 퍼셉트론

● tf.keras.Sequential 클래스를 이용한 모델 구성

■ tf.keras.layers.Dense 클래스를 이용한 완전연결층 구성

```
tf.keras.layers.Dense(units, activation=None, use_bias=True,  
                        kernel_initializer='glorot_uniform',  
                        bias_initializer='zeros')
```

● 은닉층 및 출력층을 위한 완전연결층을 구성하기 위한 클래스

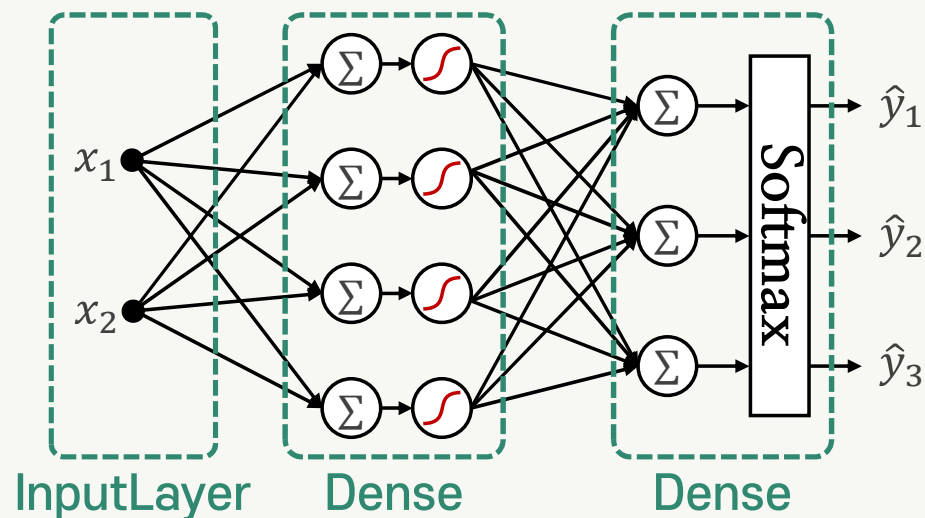
- units: 출력의 수(뉴런의 수)
- activation: 사용할 활성화함수
- use_bias: 바이어스 사용 여부
- kernel_initializer, bias_initializer: 가중치 및 바이어스의 초기화

예 h_layer = tf.keras.layers.Dense(4, activation='sigmoid')



2. 실습: 텐서플로를 이용한 다층 퍼셉트론

- tf.keras.Sequential 클래스를 이용한 모델 구성

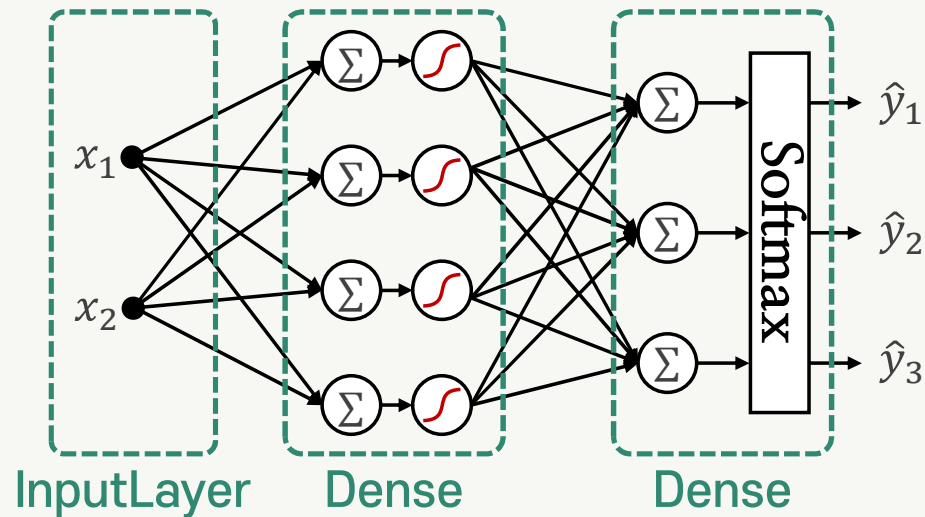


➡

```
bp_model_tf = keras.Sequential()
bp_model_tf.add(layers.InputLayer(input_shape=(2,)))
bp_model_tf.add(layers.Dense(4, activation='sigmoid'))
bp_model_tf.add(layers.Dense(nClasses, activation='softmax'))
```

2. 실습: 텐서플로를 이용한 다층 퍼셉트론

- tf.keras.Sequential 클래스를 이용한 모델 구성



→

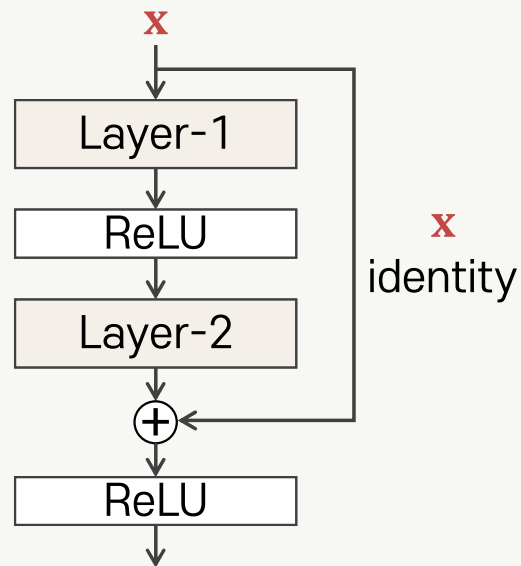
```
bp_model_tf = keras.Sequential([  
    layers.Dense(4, input_shape=(2,), activation='sigmoid'),  
    layers.Dense(nClasses, activation='softmax')  
])
```


2. 실습: 텐서플로를 이용한 다층 퍼셉트론

● 함수형 API를 이용한 모델 구성

- 다양한 형태의 층과 비 순차적 연결이 존재하는 모델의 구성에 활용

예 잔차 학습 블록

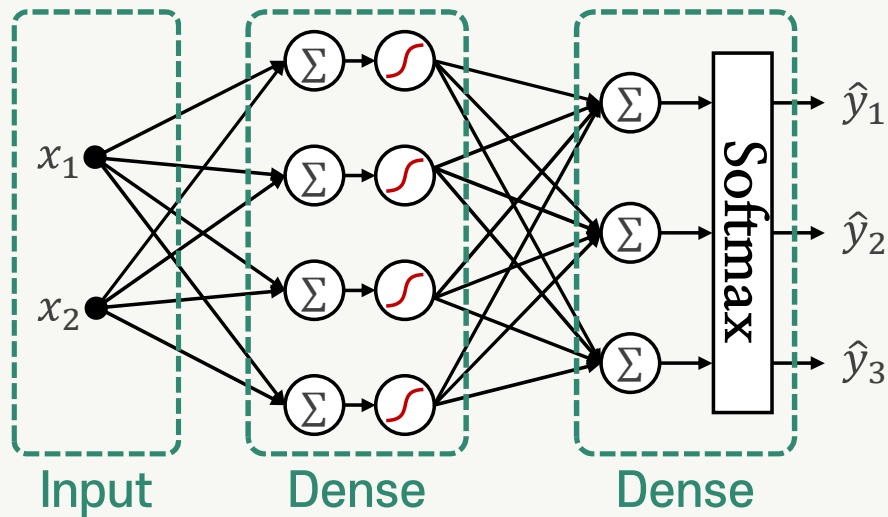


- 각 층의 출력에 해당되는 텐서를 이어지는 층의 인스턴스에 전달하는 형식으로 그래프를 구성



2. 실습: 텐서플로를 이용한 다층 퍼셉트론

● 함수형 API를 이용한 모델 구성



inputs = keras.Input(shape=(nDim,))
h = layers.Dense(4, activation='sigmoid')(inputs)
y = layers.Dense(nClasses, activation='softmax')(h)
bp_model_tf = keras.Model(inputs, y)

2. 실습: 텐서플로를 이용한 다층 퍼셉트론

● 서브클래싱 API를 이용한 모델 구성

- `tf.keras.Model` 클래스의 서브클래스로 목적에 맞게 설계된 모델 클래스를 선언하는 것

```
class BP_iris(keras.Model):  
    def __init__(self):  
        super(BP_iris, self).__init__()  
        self.h_layer = layers.Dense(4, activation='sigmoid')  
        self.o_layer = layers.Dense(nClasses, activation='softmax')  
  
    def call(self, x):  
        x = self.h_layer(x)  
        return self.o_layer(x)  
  
bp_model_tf = BP_iris()
```



2. 실습 : 텐서플로를 이용한 다층 퍼셉트론

● 모델의 컴파일

■ 모델의 훈련을 위한 설정

```
tf.keras.Model.compile(optimizer='rmsprop',  
                        loss=None, metrics=None)
```

● 주요 파라미터

- optimizer : 최적화기를 지정하는 스트링 또는 최적화기 인스턴스
- loss : 손실함수를 지정하는 스트링 또는 손실함수 인스턴스
- metrics : 훈련 및 테스트 과정에서 평가를 위한 척도의 리스트

예 `bp_model_tf.compile(optimizer=optimizers.SGD(0.1, momentum=0.9),
 loss=losses.SparseCategoricalCrossentropy(),
 metrics=['accuracy'])`



2. 실습 : 텐서플로를 이용한 다층 퍼셉트론

● 모델의 훈련

- 지정된 에폭(epoch)만큼 모델의 훈련을 반복함

```
tf.keras.Model.fit(x, y, batch_size, epochs, verbose,  
                   validation_split, validation_data, shuffle)
```

- 주요 파라미터

- x, y : 입력 데이터와 레이블
- batch_size : 미니배치 크기(디폴트는 32)
- epochs : 전체 x, y에 대한 훈련을 몇 회 반복할 것인가를 지정함
- verbose : 훈련 진행 정보 출력 방식 설정
- validation_split : x, y에 제공된 데이터 중 검증용 데이터의 비율
- validation_data : 검증용 데이터 및 레이블의 튜플
- shuffle : 데이터 순서를 섞은 후 훈련을 할 것인지 지정(디폴트는 True)



2. 실습 : 텐서플로를 이용한 다층 퍼셉트론

● 모델의 훈련

- 지정된 에폭(epoch)만큼 모델의 훈련을 반복함

```
tf.keras.Model.fit(x, y, batch_size, epochs, verbose,  
                   validation_split, validation_data, shuffle)
```

- 반환되는 결과

- History 객체 : 매 에폭에서의 손실 및 평가척도 값(훈련 및 검증)을 담고 있는 객체

예 `bp_model_tf.fit(X_tr, y_tr, batch_size=15, epochs=1000,
 verbose=2, validation_data=(X_val, y_val))`



2. 실습: 텐서플로를 이용한 다층 퍼셉트론

● 모델을 이용한 예측

- 입력 데이터에 대한 출력을 예측함

```
tf.keras.Model.predict(x, batch_size, verbose)
```

- 주요 파라미터

- x: 입력 데이터 집합이 저장된 numpy 배열
- batch_size: 미니배치 크기(디폴트는 32)
- verbose: 훈련 진행 정보 출력 방식 설정

- 반환되는 결과

- 예측한 출력이 저장된 numpy 배열

예

```
y_hat = bp_model_tf.predict(X_val, verbose=0)
y_hat_lbls = np.array([np.argmax(y_hat[k])
                        for k in range(len(X_val))])
```



정리하기

- 텐서플로와 같은 딥러닝 프레임워크를 사용하면 딥러닝 모델의 구성과 학습 등에 필요한 여러 가지 유용한 기능을 활용할 수 있다.
- 텐서는 동일 자료형의 데이터를 저장하는 다차원 배열이며, 모델의 파라미터, 입력, 모델의 중간 계산 결과를 저장하는 등의 용도로 활용된다.
- `tf.GradientTape` API를 이용하여 자동 미분을 할 수 있다.
- 그래프 실행 모드를 이용하면 계산 성능을 높일 수 있다.
- `tf.function`을 이용하여 즉시 실행 모드의 프로그램을 그래프로 변환할 수 있다.



정리하기

- Keras는 여러 가지 신경망 층, 활성화함수, 손실함수, 최적화기 등을 제공하는 고수준 오픈소스 라이브러리로서, 딥러닝 모델을 편리하게 구성하고 훈련하는데 유용하다.
- 순차적으로 구성되는 단순한 모델은 `tf.keras.Sequential` 클래스를 이용하여 구성할 수 있다.
- 복잡한 구조의 모델을 구성하려면 Keras의 함수형 API 또는 서브클래싱 API를 이용할 수 있다.



다음시간안내

04

딥러닝의 학습 기술(1)

