저수준 텐서플로 API

사용자 정의 층: keras.layers.Layer 상속

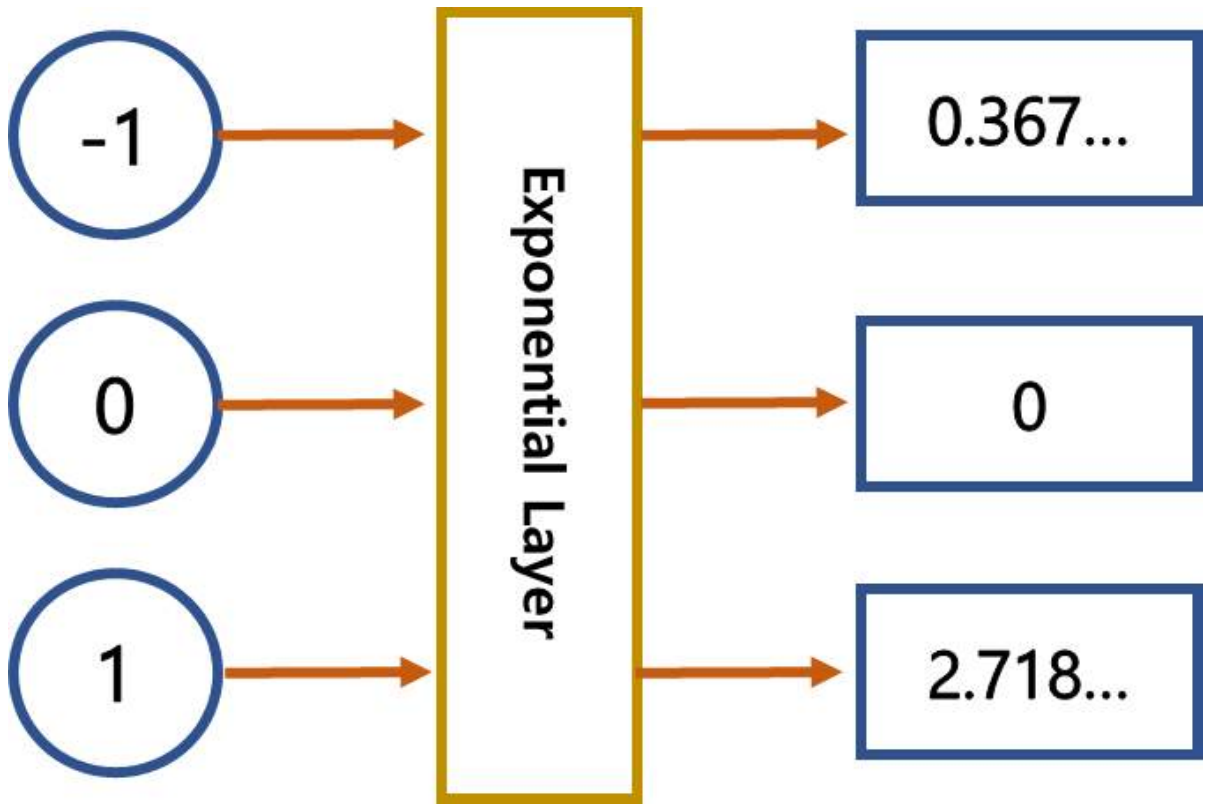
시퀀셜 API, 함수형 API, 서브클래싱 API에 기존의 층과 동일하게 사용 가능

가중치가 없는 간단한 층 → keras.layers.Lambda 로 감싸기

```
exponential_layer = keras.layers.Lambda(lambda x: tf.exp(x))
```

```
exponential_layer([-1., 0., 1.])
```

```
<tf.Tensor: shape=(3,), dtype=float32, numpy=array([0.36787948, 1.        , 2.7182817 ], dtype=float32)>
```



keras.layers.Dense 구현

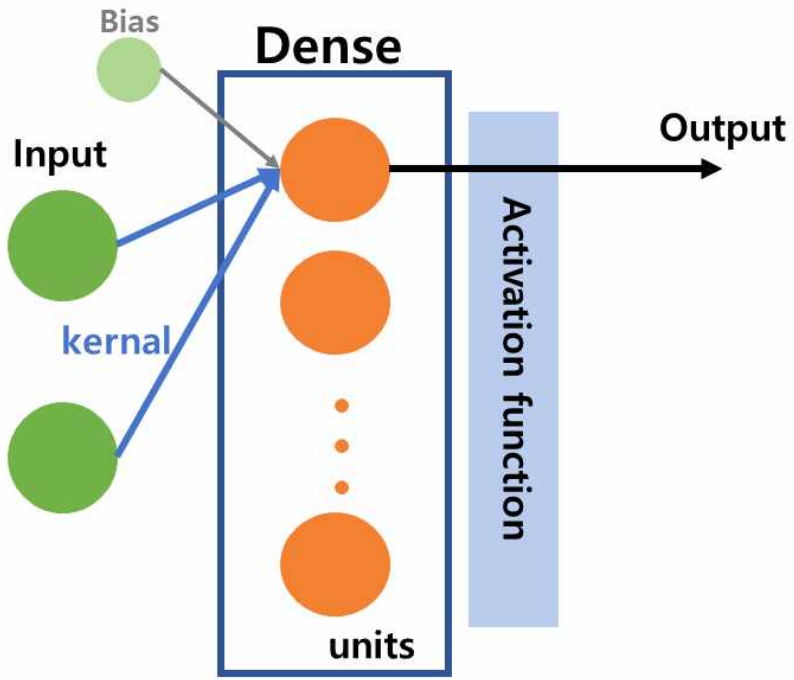
Unit 개수 지정, 활성화 함수 지정

연결 가중치(Kernal): input 개수 × unit 개수 (개)
초기값: 글로럿 초기화

편향 가중치: unit 개수 (개), 초기값 = 0

연산: [X 행렬곱 Kernal + Bias] → 활성화 함수

전체 설정 저장: get_config 매서드



사용자 정의 모델: Keras.Model 상속

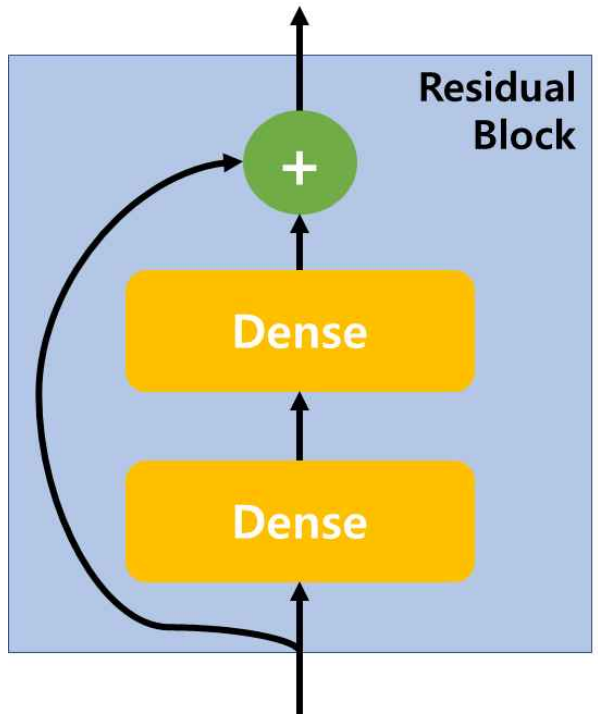
잔차 블록

블록 내부에 Dense층이 n_layer개

각각의 Dense층에는 노드가 n_neurons개

Dense층의 활성화 함수는 ELU, 초기화 방법은 He 초기화

Input이 Dense층들을 통과한 값과 Input값 합치기



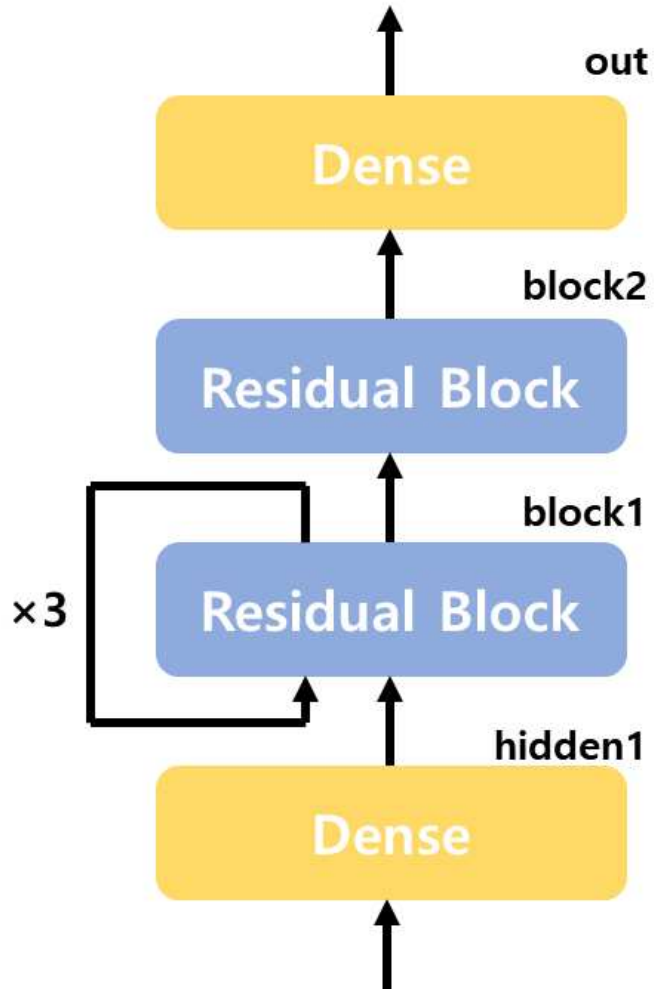
잔차 블록 모델

hidden1: 뉴런 30개, 활성화 함수 ELU, He 초기화

block1: 내부 Dense층 2개, 각 Dense당 뉴런 30개
1번 통과 + 3번 반복 통과 = 4번 통과

block2: 내부 Dense층 2개, 각 Dense당 뉴런 30개

out: 뉴런 개수 지정 output_dim개



재구성 손실

보조 출력에 연결된 손실, 주 손실과 합해 규제의 역할, 과대적합 방지

주 은닉층 5개

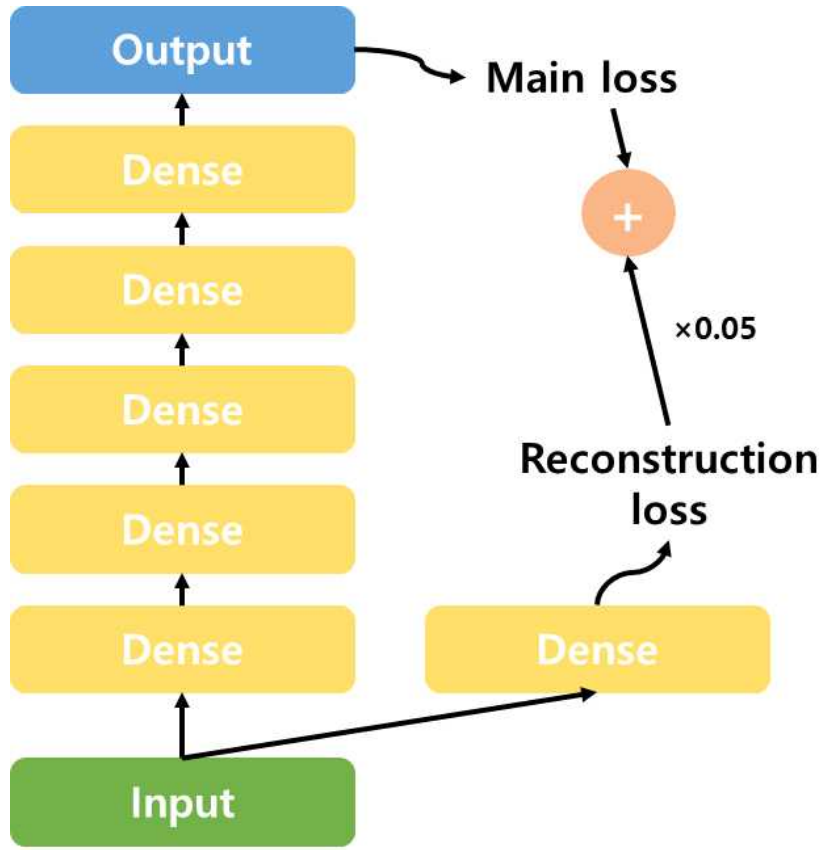
각 은닉층: 뉴런 30개, 활성화함수 SeLU, 르쿤 초기화

주 출력층: 뉴런 개수 지정 output_dim개

보조 출력층: 뉴런 개수 입력층의 뉴런 개수와 동일

재구성 손실: (보조 출력 − 입력)²

손실 = 주 손실 + 0.05 × 재구성 손실



자동 미분

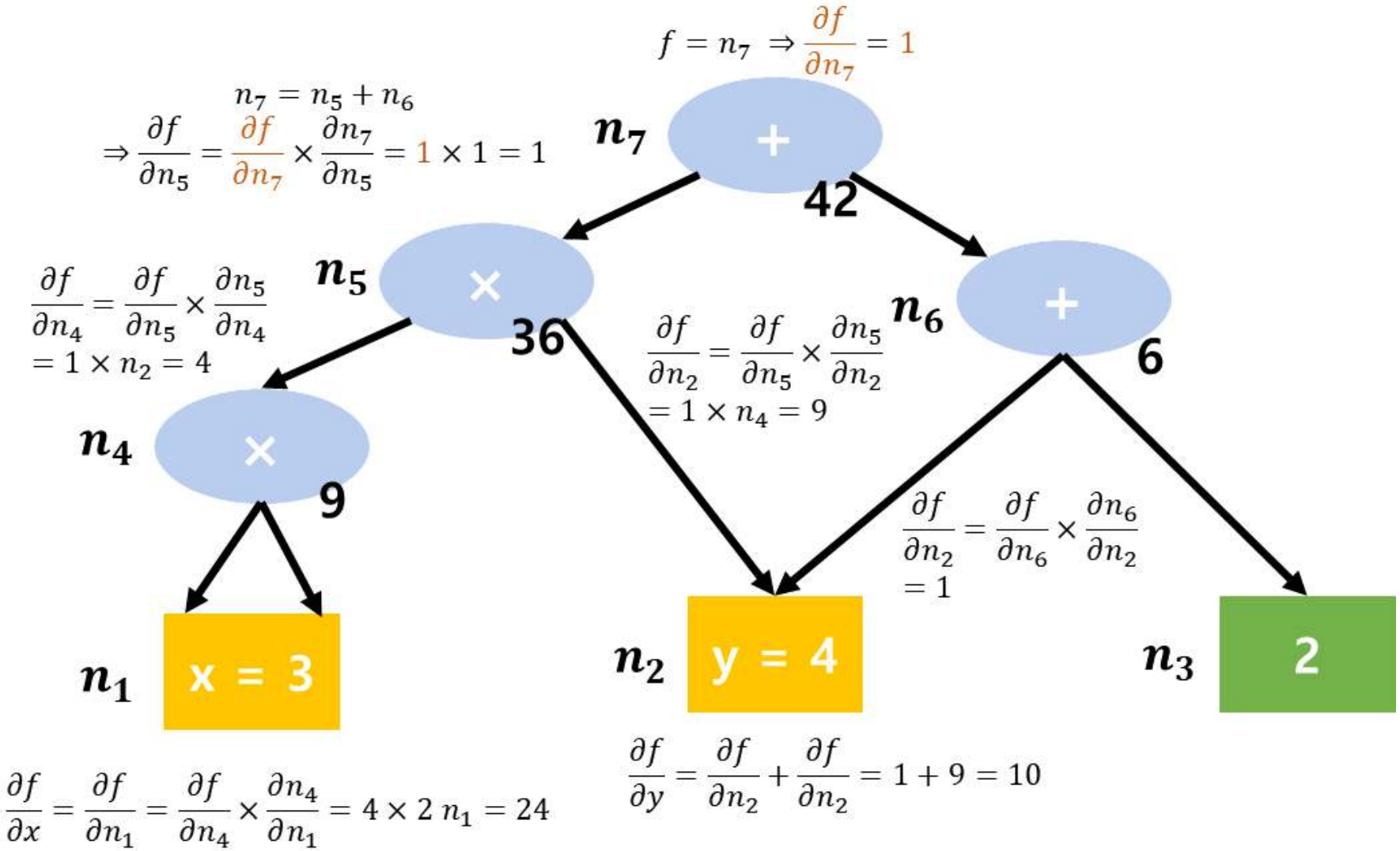
tf.GradientTape(): 후진 모드 자동 미분을 이용해 실행된 모든 연산을 테이프에 기록

기록된 연산 중 gradient() 매서드를 이용해 그래디언트 값 불러오기

한 번 gradient() 매서드를 호출하면 자동으로 테이프가 지워짐
 지워지지 않으려면: persistent = True

변수(tf.Variable)가 포함된 연산만을 기록

후진 모드 자동 미분


$f = n_7 \Rightarrow \frac{\partial f}{\partial n_7} = 1$
$n_7 = n_5 + n_6$
$\Rightarrow \frac{\partial f}{\partial n_5} = \frac{\partial f}{\partial n_7} \times \frac{\partial n_7}{\partial n_5} = 1 \times 1 = 1$
n_7
+
42
n_5
×
36
n_6
+
6
$\frac{\partial f}{\partial n_4} = \frac{\partial f}{\partial n_5} \times \frac{\partial n_5}{\partial n_4}$
$= 1 \times n_2 = 4$
$\frac{\partial f}{\partial n_2} = \frac{\partial f}{\partial n_5} \times \frac{\partial n_5}{\partial n_2}$
$= 1 \times n_4 = 9$
n_4
×
9
$\frac{\partial f}{\partial n_2} = \frac{\partial f}{\partial n_6} \times \frac{\partial n_6}{\partial n_2}$
$= 1$
n_1
x = 3
n_2
y = 4
n_3
2
$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial n_1} = \frac{\partial f}{\partial n_4} \times \frac{\partial n_4}{\partial n_1} = 4 \times 2\, n_1 = 24$
$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial n_2} + \frac{\partial f}{\partial n_2} = 1 + 9 = 10$

텐서플로 함수

파이썬 함수에 텐서(tf.constant) 대입 → 텐서 출력

```
cube(tf.constant(2.))
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=8.0>
```

tf.function(파이썬 함수) → 텐서플로 함수

```
# 텐서플로 함수로 바꾸기
tf_cube = tf.function(cube)
tf_cube
```

```
<tensorflow.python.eager.def_function.Function at 0x7fc4f57a4090>
```

텐서를 출력.

자료형 주의: int32, float32, float64...

데코레이터(@tf.function)로 텐서플로 함수 생성

```
# tf.function 데코레이터 사용
@tf.function
def tf_cube(x):
  return x ** 3
```

텐서플로 함수를 파이썬 함수로: .python_function()

```
# 원본 파이썬 함수 변환
tf_cube.python_function(2)
```

```
8
```

참고 자료

- 오헬리앙 제롱, *핸즈온 머신러닝 2판* (한빛미디어, 2020), 479-500