Custom model and Preprocessing with Tensorflow

Hands-On Machine Learning Part2

& Deep Learning from Scratch 3

TAVE Research DL001

Heeji Won

Contents

Contents

01. Custom model and training

- Loss function

```
class HuberLoss(keras.losses.Loss):
    def __init__(self, threshold=1.0, **kwargs):
        self.threshold = threshold
        super().__init__(**kwargs)

    def call(self, y_true, y_pred):
        error = y_true - y_pred
        is_small_error = tf.abs(error) < self.threshold
        squared_loss = tf.square(error) / 2
        linear_loss = self.threshold * tf.abs(error) - self.threshold**2 / 2
        return tf.where(is_small_error, squared_loss, linear_loss)

    def get_config(self):   # threshold 파라미터 추가
        base_config = super().get_config()
        return {**base_config, 'threshold':self.threshold}
```

```
model.compile(loss=HuberLoss(2.), optimizer='nadam')
model = keras.models.load_model('my_model_with_a_custom_loss.h5',
                                custom_objects={'HuberLoss': HuberLoss})
```

- Activation function, initializer, regularizer, constraints

```
def my_softplus(z):   # keras.activations.softplus()나 tf.nn.softplus()와 동일
    return tf.math.log(tf.exp(z) + 1.0)

def my_glorot_initializer(shape, dtype=tf.float32):  # keras.initializers.glorot
    stddev = tf.sqrt(2. / (shape[0] + shape[1]))
    return tf.random.normal(shape, stddev=stddev, dtype=dtype)

def my_l1_regularizer(weights):   # keras.regularizer.l1(0.01)과 동일
    return tf.reduce_sum(tf.abs(0.01 * weights))

def my_positive_weights(weights):  # tf.nn.relu(weights)와 동일
    return tf.where(weights < 0, tf.zeros_like(weights), weights)
```

```
layer = keras.layers.Dense(30, activaiton=my_softplus,
                           kernel_initializer=my_glorot_initializer,
                           kernel_regularizer=my_l1_regularizer,
                           kernel_constraint=my_positive_weights)
```

01. Custom model and training

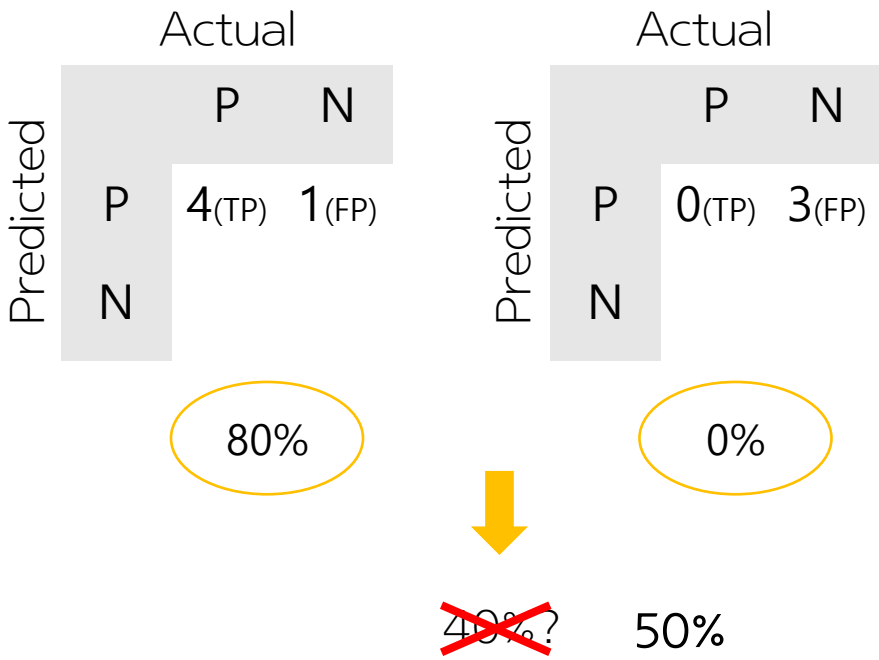
- **Metrics vs Loss**
  - Metrics is not different from loss essentially
    - **Loss is for training** (differentiable)
    - **Metrics is for evaluating** (easy)
  - In practice, making metrics function is the same as making loss function

> **Streaming metric** (or stateful metric)

- updates gradually with each batch

cf) Precision

$$Precision = \frac{TP}{TP + FP}$$



✓ Streaming metric is necessary

01. Custom model and training

- Custom layers

  - Simple Dense layer

```
class MyDense(keras.layers.Layer):
    def __init__(self, units, activation=None, **kwargs):
        super().__init__(**kwargs)
        self.units = units
        self.activation = keras.activations.get(activation)

    def build(self, batch_input_shape):
        self.kernel = self.add_weight(
            name='kernel', shape=[batch_input_shape[-1], self.units],
            initializer='glorot_normal')
        self.bias = self.add_weight(
            name='bias', shape=[self.units], initializer='zero')
        super().build(batch_input_shape)  # 마지막에 호출해야

    def call(self, X):
        return self.activation(X @ self.kernel + self.bias)

    def compute_output_shape(self, batch_input_shape):
        return tf.TensorShape(batch_input_shape.as_list()[:-1] + [self.units])

    def get_config(self):
        base_config = super().get_config()
        return {**base_config, 'units': self.units,
                'activation': keras.activation.serialize(self.activation)}
```

  - Multiple inputs

```
# 두 개의 입력과 세 개의 출력을 만드는 층

class MyMultiLayer(keras.layers.Layer):
    def call(self, X):
        X1, X2 = X
        return [X1 + X2, X1 * X2, X1 / X2]

    def compute_output_shape(self, batch_input_shape):
        b1, b2 = batch_input_shape
        return [b1, b1, b1]
```

튜플

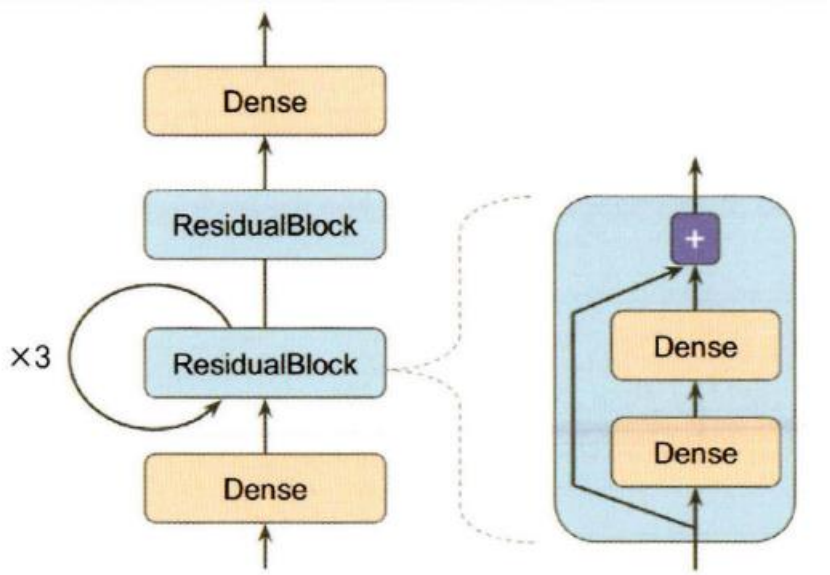
  - Behaving differently in train set and test set

```
# 훈련 시에만 가우스 잡음을 추가하는 층

class MyGaussianNoise(keras.layers.Layer):
    def __init__ (self, stddev, **kwargs):
        super().__init__(**kwargs)
        self.stddev = stddev

    def call (self, X, training=None):
        if training :
            noise = tf.random.normal(tf.shape(X), stddev=self.stddev)
            return X + noise
        else :
            return X

    def compute_output_shape(self, batch_input_shape):
        return batch_input_shape
```

01. Custom model and training

- Custom model

– Model with ResidualBlock layer




```python
# ResidualBlock 층 만들기 (출력에 입력을 더하는 층)

class ResidualBlock(keras.layers.Layer):
    def __init__(self, n_layers, n_neurons, **kwargs):
        super().__init__(**kwarge)
        self.hidden = [keras.layers.Dense(n_neurons, activation='elu',
                                          kernel_initializer='he_normal')
                       for _ in range(n_layers)]

    def call(self, inputs):
        Z = inputs
        for layer in self.hidden:
            Z = layer(Z)
# 모델 정의

class ResidualRegressor(keras.Model):
    def __init_(self, output_dim, **kwargs):
        super().__init__(**kwargs)
        self.hidden1 = keras.layers.Dense(30, activation='elu',
                                          kernel_initializer='he_normal')
        self.block1 = ResidualBlock(2, 30)
        self.black2 = ResidualBlock(2, 30)
        self.out = keras.layers.Dense(output_dim)

    def call(self, inputs):
        Z = self.hidden1(inputs)
        for _ in range(1 + 3):
            Z = self.block1(Z)
        Z = self.block2(Z)
        return self.out(Z)
```

01. Custom model and training

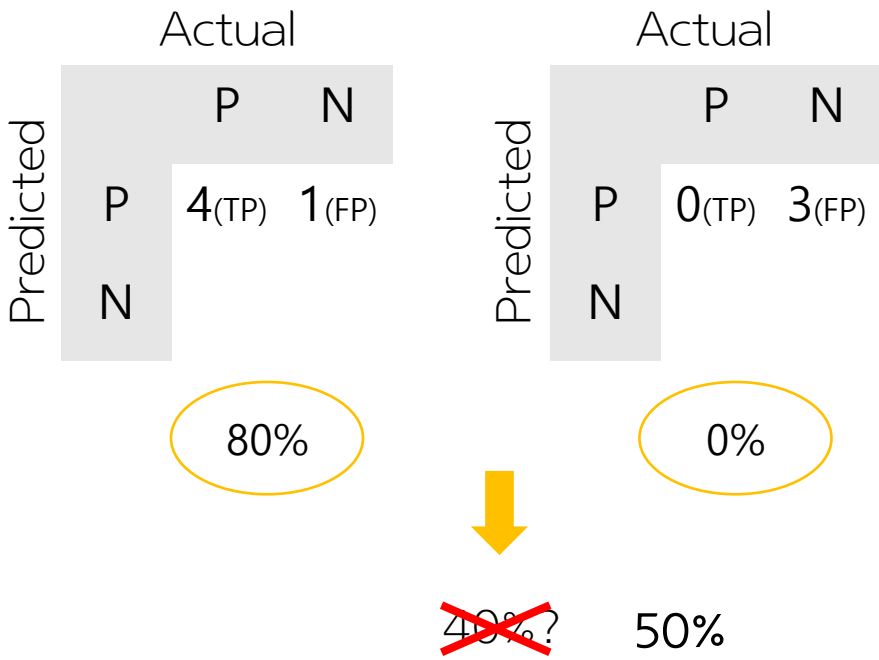
- **Metrics vs Loss**
  - Metrics is not different from loss essentially
    - **Loss is for training** (differentiable)
    - **Metrics is for evaluating** (easy)
  - In practice, making metrics function is the same as making loss function

> **Streaming metric** (or stateful metric)

- updates gradually with each batch

cf) Precision

$$Precision = \frac{TP}{TP + FP}$$



✓ Streaming metric is necessary

01. Custom model and training

- Reconstruction loss

: MSE between Reconstruction and input

```
# 사용자 정의 재구성 손실을 가지는 모델

class ReconstructionRegressor(keras.Model):
    def __init__(self, output_dim, **kwargs):
        super().__init__(**kwargs)
        self.hidden = [keras.layers.Dense(30, activation='selu',
                                          kernel_initializer='lecun_normal')
                       for _ in range(5)]
        self.out = keras.layers.Dense(output_dim)

    def build(self, batch_input_shape):  # 완전 연결 층 추가 (입력을 재구성)
        n_inputs = batch_input_shape[-1]
        self.reconstruct = keras.layers.Dense(n_inputs)
        super().build(batch_input_shape)

    def call(self, inputs):
        Z = inputs
        for layer in self.hidden:
            Z = layer(Z)
        reconstruction = self.reconstruct(Z)
        recon_loss = tf.reduce_mean(tf.square(reconstruction - inputs))
        self.add_loss(0.05 * recon_loss)
        return self.out(Z)
```

- Automatic differentiation

```
# GradientTape 한번 이상 호출 시

with tf.GradientTape(persistent=True) as tape:
    z = f(w1, w2)

dz_dw1 = tape.gradient(z, w1)
dz_dw2 = tape.gradient(z, w2)
print(dz_dw1)
print(dz_dw2)

del tape
```

```
tf.Tensor(36.0, shape=(), dtype=float32)
tf.Tensor(10.0, shape=(), dtype=float32)
```

01. Custom model and training

- Reconstruction loss

: MSE between Reconstruction and input

```
# 사용자 정의 재구성 손실을 가지는 모델

class ReconstructionRegressor(keras.Model):
    def __init__(self, output_dim, **kwargs):
        super().__init__(**kwargs)
        self.hidden = [keras.layers.Dense(30, activation='selu',
                                          kernel_initializer='lecun_normal')
                       for _ in range(5)]
        self.out = keras.layers.Dense(output_dim)

    def build(self, batch_input_shape):  # 완전 연결 층 추가 (입력을 재구성)
        n_inputs = batch_input_shape[-1]
        self.reconstruct = keras.layers.Dense(n_inputs)
        super().build(batch_input_shape)

    def call(self, inputs):
        Z = inputs
        for layer in self.hidden:
            Z = layer(Z)
        reconstruction = self.reconstruct(Z)
        recon_loss = tf.reduce_mean(tf.square(reconstruction - inputs))
        self.add_loss(0.05 * recon_loss)
        return self.out(Z)
```

- Automatic differentiation

```
# GradientTape 한번 이상 호출 시

with tf.GradientTape(persistent=True) as tape:
    z = f(w1, w2)

dz_dw1 = tape.gradient(z, w1)
dz_dw2 = tape.gradient(z, w2)
print(dz_dw1)
print(dz_dw2)

del tape
```

```
tf.Tensor(36.0, shape=(), dtype=float32)
tf.Tensor(10.0, shape=(), dtype=float32)
```

Contents

02. Data loading and Preprocessing

- Data loading and Preprocessing

```
# 재사용가능한코드를 만들기 위해 지금까지의 코드를 하나의 헬퍼 함수로 만들기
# 적재, 전처리, 셔플링, 반복, 배치를 적용한 데이터 반환

def csv_reader_dataset(filepaths, repeat=1, n_readers=5,
                       n_read_threads=None, shuffle_buffer_size=10000,
                       n_parse_threads=5, batch_size=32):

    dataset = tf.data.Dataset.list_files(filepaths).repeat(repeat)
    dataset = dataset.interleave(
        lambda filepath: tf.data.TextLineDataset(filepath).skip(1),
        cycle_length=n_readers, num_parallel_calls=n_read_threads)
    dataset = dataset.shuffle(shuffle_buffer_size)
    dataset = dataset.map(preprocess, num_parallel_calls=n_parse_threads)
    return dataset.batch(batch_size).prefetch(1)  # 아래에서 살펴봄
```

```
def preprocess(line):
    defs = [0.] * n_inputs + [tf.constant([], dtype=tf.float32)]
       # defs : [0, 0, 0, ..., 0, tf.Tensor]  -- X부분은 0으로 y는 텐서로
    fields = tf.io.decode_csv(line, record_defaults=defs)
       # record_defaults는 기본값 배열(누락값의 기본값을 0으로, y는 기본값 X)

    x = tf.stack(fields[:-1])  # stack은 1D 텐서 배열을 반환
    y = tf.stack(fields[-1:])
    return (x - X_mean) / X_std, y
```

- list_files() 함수 : 파일 경로를 섞은 데이터 셋 반환
- interleave() 함수 : 한 번에 n_readers개의 파일 한 줄씩 번갈아 읽기
- num_parallel_calls 옵션 : 병렬화 개수
- prefetch(1) : 마지막에 prefetch(1)를 호출하면 데이터셋은 항상 한 배치가 미리 준비되도록!
  즉, 한 배치를 훈련하는 동안 다음 배치를 준비함

02. Data loading and Preprocessing

- Preprocessing categorical feature
  - One-hot vector

```
vocab = [ "<1H OCEAN" , "INLAND" , "NEAR OCEAN" , "NEAR BAY", "ISLAND" ]
indices = tf.range(len(vocab), dtype=tf.int64)

# tf.lookup.KeyValueTensorInitializer : 범주 리스트와 해당 인덱스를를 전달하여
                                    # 룩업 테이블을 위해 초기화 객체를 만듦
table_init = tf.lookup.KeyValueTensorInitializer(vocab, indices)


# 어휘 사전에 없는 범주를 찾으면 룩업 테이블이 계산한 이 범주의 해시값을 이용해
# oov(out-of-vocabulary) 버킷 중 하나에 할당 (있는 범주 다음부터! 여기서는 5, 6
num_oov_buckets = 2
table = tf.lookup.StaticVocabularyTable(table_init, num_oov_buckets)
```

```
# 룩업 테이블을 사용해 원-핫 벡터로 인코딩해보기

categories = tf.constant(['NEAR BAY', 'DESERT', 'INLAND', 'INLAND'])
cat_indices =  table.lookup(categories)
cat_indices
```

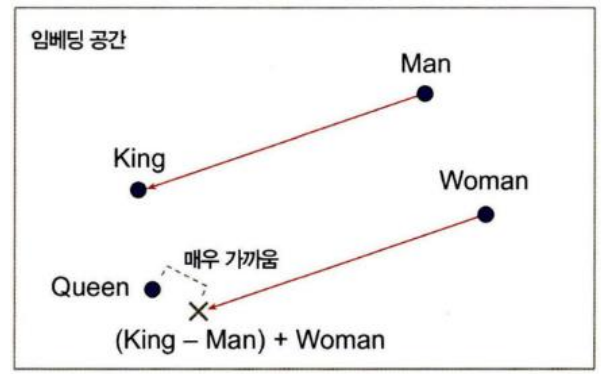
```
<tf.Tensor: shape=(4,), dtype=int64, numpy=array([3, 5, 1, 1], dtype=int64)>
```

```
cat_one_hot = tf.one_hot(cat_indices, depth=len(vocab) + num_oov_buckets)
cat_one_hot
```

```
<tf.Tensor: shape=(4, 7), dtype=float32, numpy=
array([[0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.]], dtype=float32)>
```

- Embedding

  : Trainable Dense Vector




```
tf.nn.embedding_lookup(embedding_matrix, cat_indices)  # [3, 5, 1, 1]해당 벡터
```

```
<tf.Tensor: shape=(4, 2), dtype=float32, numpy=
array([[0.7686013 , 0.8976238 ],
       [0.50123084, 0.02763808],
       [0.09627533, 0.7377459 ],
       [0.09627533, 0.7377459 ]], dtype=float32)>
```

```
# 모두를 연결해보기 -> 임베딩을 학습하는 케라스 모델 만들 수 있음

regular_inputs = keras.layers.Input(shape=[8])
cat_embed = keras.layers.Embedding(input_dim=6, output_dim=2)(cat_indices)
encoded_inputs = keras.layers.concatenate([regular_inputs, cat_embed])
outputs = keras.layers.Dense(1)(encoded_inputs)  # Embedding 층과 동일한 역할
model = keras.models.Model(inputs=[regular_inputs, categories],
                           outputs=[outputs])
```

Contents

03. Deep learning from Scratch

- Review

```python
class Function(object):
    def __call__(self, *inputs):  # * 는 가변 길이 인수
        xs = [x.data for x in inputs]
        ys = self.forward(*xs)
        if not isinstance(ys, tuple):
            ys = (ys, )
        outputs = [Variable(as_array(y)) for y in ys]

        self.generation = max([x.generation for x in inputs]) # 추가된 부분

        for output in outputs:
            output.set_creator(self)

        self.inputs = inputs
        self.outputs = outputs
        return outputs if len(outputs) > 1 else outputs[0]

    def forward(self, xs):
        raise NotImplementedError()

    def backward(self, gys):
        raise NotImplementedError()
```

```python
class Variable:
    def __init__(self, data):
        if data is not None:
            if not isinstance(data, np.ndarray):
                raise TypeError('{}은(는) 지원하지 않습니다.'.format(type(data)))

        self.data = data
        self.grad = None
        self.creator = None
        self.generation = 0   # 세대 수를 기록하는 변수
```

```python
    def set_creator(self, func):
        self.creator = func
        self.generation = func.generation + 1  # 세대를 기록한다 (부모세대 + 1)

    def backward(self):
        if self.grad is None:
            self.grad = np.ones_like(self.data)

        funcs = []   # (추가된 부분) 순서를 정렬할 함수들의 리스트
        seen_set = set()  # (함수 여러번 불리는 일 방지) 함수의 unique값

        def add_func(f):  # 함수 리스트를 세대 순으로 정렬하는 역할
            if f not in seen_set:
                funcs.append(f)
                seen_set.add(f)
                funcs.sort(key=lambda x: x.generation)

        add_func(self.creator)

        while funcs:
            f = funcs.pop()
            gys = [output.grad for output in f.outputs]
            gxs = f.backward(*gys)  # 인수에 * 붙여 호출하면 리스트 언택됨
            if not isinstance(gxs, tuple):
                gxs = (gxs, )

            for x, gx in zip(f.inputs, gxs): # x에 grad, func 넣어주기
                if x.grad is None:
                    x.grad = gx
                else:
                    x.grad = x.grad + gx   # 이미 grad가 있는 경우에 더해주기

                if x.creator is not None:
                    add_func(x.creator)  # 수정 전 : funcs.append(x.creator)

    def cleargrad(self):
            self.grad = None
```

03. Deep learning from Scratch

- Modify..

```python
class Function:
    def __call__(self, *inputs):
        inputs = [as_variable(x) for x in inputs]  # as_variable 함수 추가해주기

        xs = [x.data for x in inputs]
        ys = self.forward(*xs)
        if not isinstance(ys, tuple):
            ys = (ys, )
        outputs = [Variable(as_array(y)) for y in ys]

        if Config.enable_backprop:  # if문 안으로 넣으줌
            self.generation = max([x.generation for x in inputs]) # 세대 만들필
            for output in outputs:  # 연결 해줄 필요 X in 역전파 비활성 모드
                output.set_creator(self)
            self.inputs = inputs
            self.outputs = [weakref.ref(output) for output in outputs]

        return outputs if len(outputs) > 1 else outputs[0]

    def forward(self, xs):
        raise NotImplementedError()

    def backward(self, gys):
        raise NotImplementedError()
```

```python
# using_config 함수 구현

@contextlib.contextmanager

def using_config(name, value):
    old_value = getattr(Config, name)  # Config.name 값
    setattr(Config, name, value)  # Config.name 값  value값으로 넣어주기
    try:
        yield
    finally:
        setattr(Config, name, old_value)
```

```python
# with문 간단히하기

def no_grad():
    return using_config('enable_backprop', False)

with no_grad():
    x = Variable(np.array(2.0))
    y = square(x)
```

03. Deep learning from Scratch

- Modify..

```python
class Variable:

    __array_prioirity__ = 200  # 큰 값으로 지정

    def __init__(self, data, name=None):  # name 추가
        if data is not None:
            if not isinstance(data, np.ndarray):
                raise TypeError('{} is not supported'.format(type(data)))

        self.data = data
        self.name = name
        self.grad = None
        self.creator = None
        self.generation = 0

    def set_creator(self, func):
        self.creator = func
        self.generation = func.generation + 1

    def backward(self, retain_grad=False):  # 중간값 그레디언트 유지할지
        if self.grad is None:
            self.grad = np.ones_like(self.data)

        funcs = []
        seen_set = set()

        def add_funcs(f):
            if f not in seen_set:
                funcs.append(f)
                seen_set.add(f)
                funcs.sort(key=lambda x: x.generation)

        add_funcs(self.creator)
```

```python
        while funcs:
            f = funcs.pop()
            gys = [output().grad for output in f.outputs]
                # output 약한 참조해서 데이터에 접근하려면 output() 해줘야
            gxs = f.backward(*gys)  # 인수에 * 붙이면 리스트 언팩됨
            if not isinstance(gxs, tuple):
                gxs = (gxs, )

            for x, gx in zip(f.inputs, gxs):
                if x.grad is None:
                    x.grad = gx
                else:
                    x.grad = x.grad + gx

                if x.creator is not None:
                    add_funcs(x.creator)

            if not retain_grad:  # f.inputs의 grad만 남기고 f.outputs은 제거
                for y in f.outputs:
                    y().grad = None  # y는 약한 참조이므로

    def cleargrad(self):
        self.grad = None

    @property  # shape 메서드를 인스턴스 변수처럼 사용할 수 있게
            # x.shape() 대신 x.shape으로
    def shape(shape):
        return self.data.shape

    @property
    def ndim(self):
        return self.data.ndim

    @property
    def size(self):
        return self.data.size
```

03. Deep learning from Scratch

- Modify..

```
    @property
    def dtype(self):
        return self.data.dtype

    def __len__(self):
        return len(self.data)

    def __repr__(self):   # print 하면 variable(...) 형태로 출력하기
        if self.data is None:
            return 'variable(None)'
        p = str(self.data).replace('₩n', '₩n' + ' ' * 9) # 다차원 array일 때 줄
        return 'variable(' + p + ')'
```

```
def add(x0, x1):
    x1 = as_array(x1)
    return Add()(x0, x1)

def mul(x0, x1):
    x1 = as_array(x1)
    return Mul()(x0, x1)
```

```
# 좌변이 float이나 int일 수 있으므로

Variable.__add__ = add
Variable.__mul__ = mul
Variable.__radd__ = add # (float) + Variable 이면 연산자의 오른쪽에 위치한
Variable.__rmul__ = mul # Variable에서 radd 메서드가 호출됨
```

```
class Neg(Function):
    def forward(self, x):
        return -x

    def backward(self, gy):
        return -gy

def neg(x):
    return Neg()(x)

Variable.__neg__ = neg

class Sub(Function):
    def forward(self, x0, x1):
        y = x0 - x1
        return y

    def backward(self, gy):
        return gy, -gy

def sub(x0, x1):
    x1 = as_array(x1)
    return Sub()(x0, x1)

Variable.__sub__ = sub

# __rsub__는 제대로 처리 못함 (순서를 바꿔서 계산하므로)
# => rsub 함수 만들기

def rsub(x0, x1):
    x1 = as_array(x1)
    return Sub()(x1, x0)  # 순서 바꾸기

Variable.__rsub__ = rsub
```

Thank you