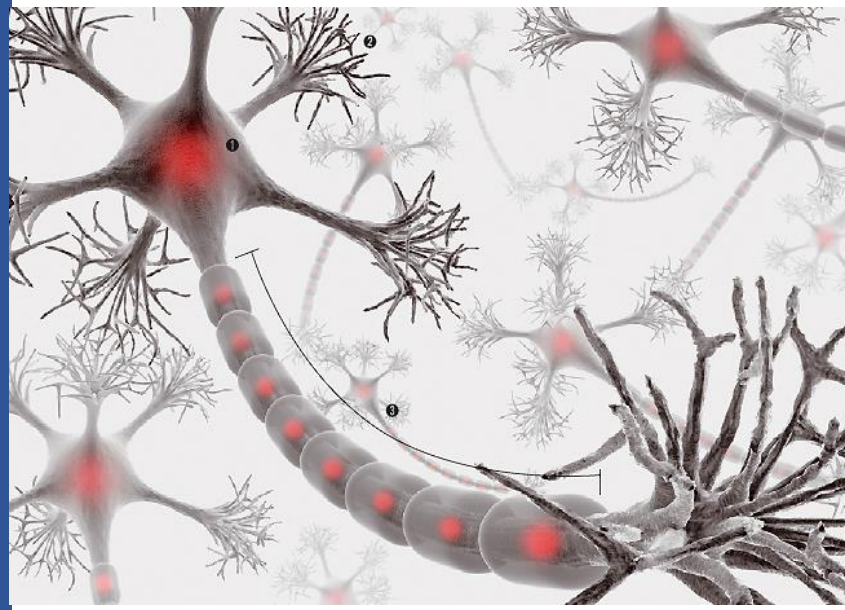
(DNN) Sine Regression

학습 목표

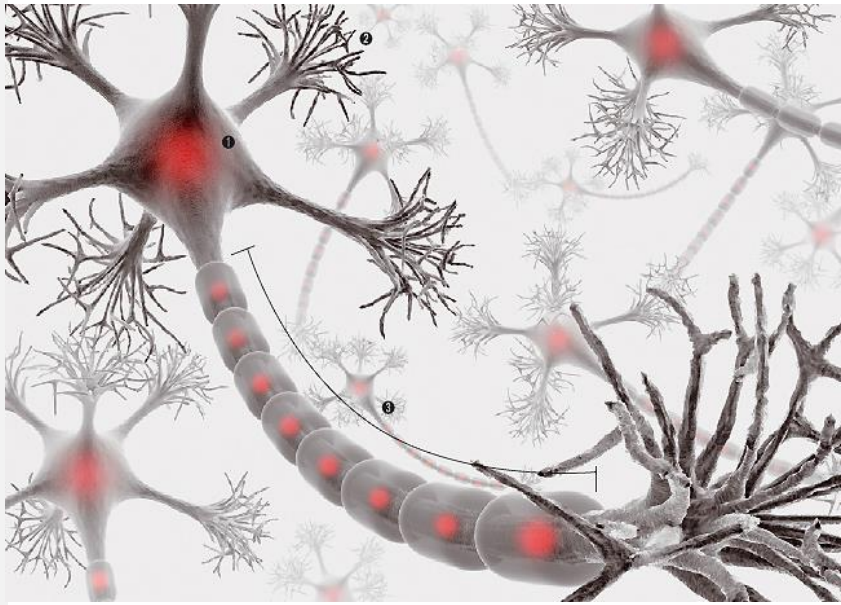
- Sine 함수에서 샘플링된 데이터를 Regression하는 신경망 모델을 만들어 본다.

주요 내용

- 1. 문제 정의
- 2. 데이터 준비
- 3. 모델 정의 및 훈련, 검증



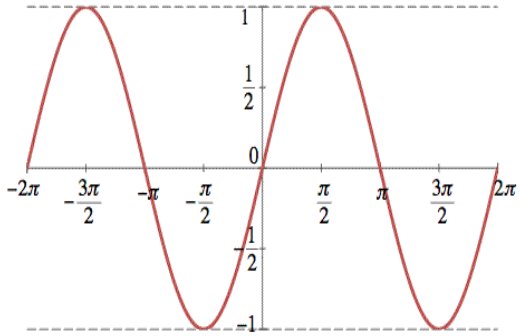
1 문제 정의



Regression 문제

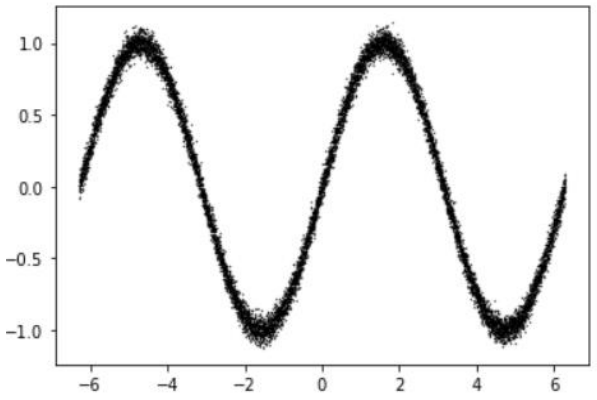
Sine 함수에서 샘플링한 데이터를 Regression해보자!

Sine Function

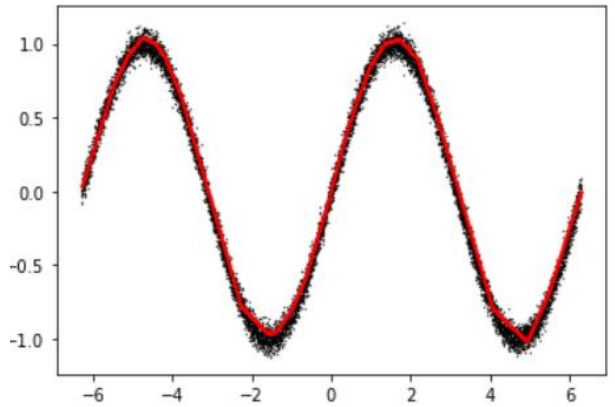


$$f(\theta) = \sin(\theta)$$

Data Sampling

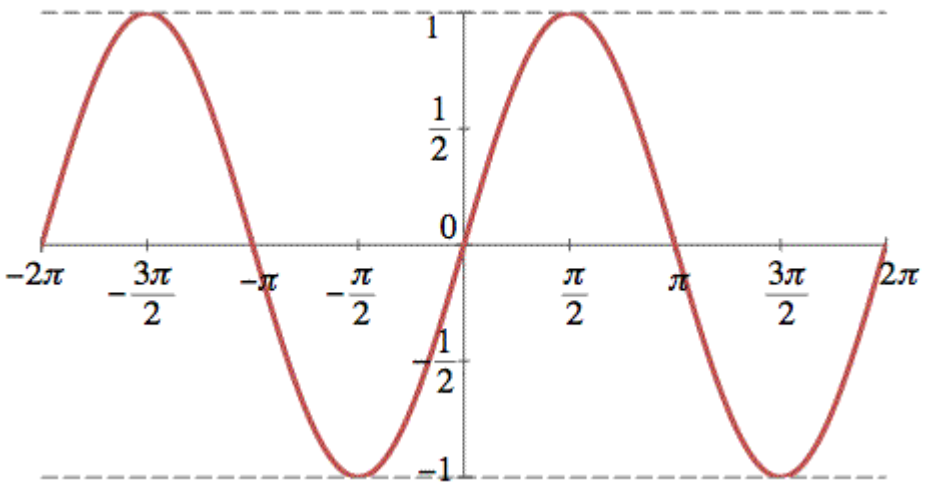


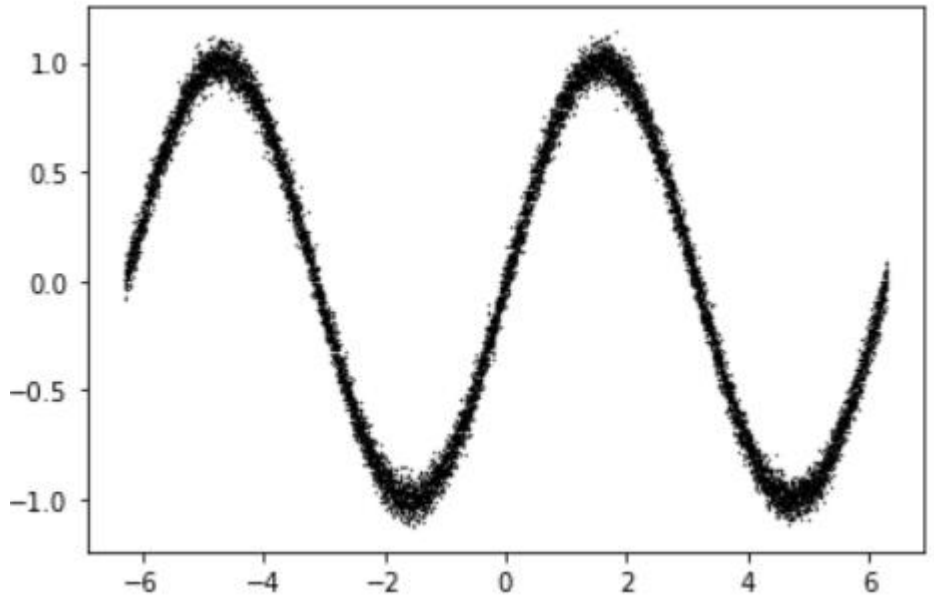
Regression



Hint : Dataset Sampling

$$f(\theta) = \sin(\theta)$$





Input 생성

θ : $[-2\pi, 2\pi]$에서 10000개의 θ를 등간격 샘플링

Label 생성

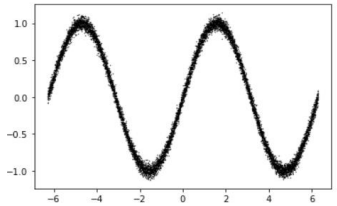
$$\sin(\theta) + 0.05 * \mathcal{N}(0,1)$$

표준정규분포 Noise 추가

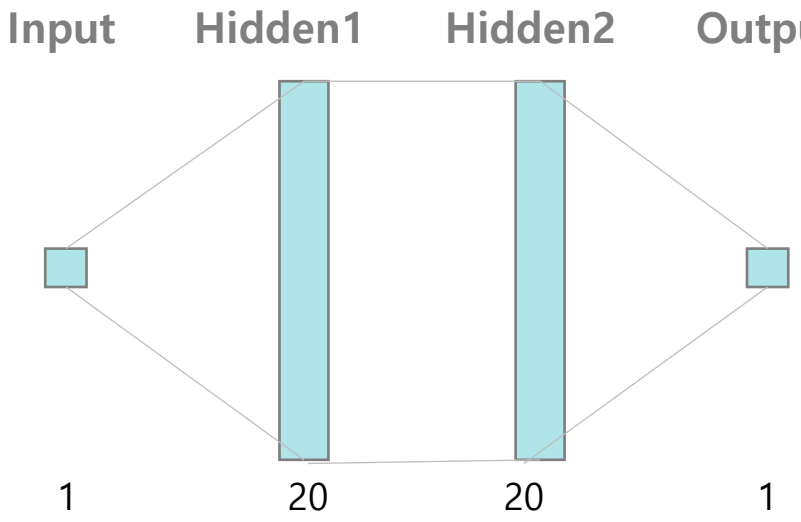
Hint : Network 구성

$x = (\theta)$

- 1차원 입력 벡터



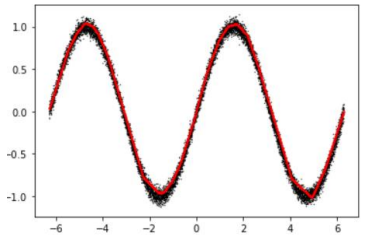
Fully Connected Network



- 2-hidden layers
- 20 neurons
- ReLu

$f(\theta) = \sin(\theta)$

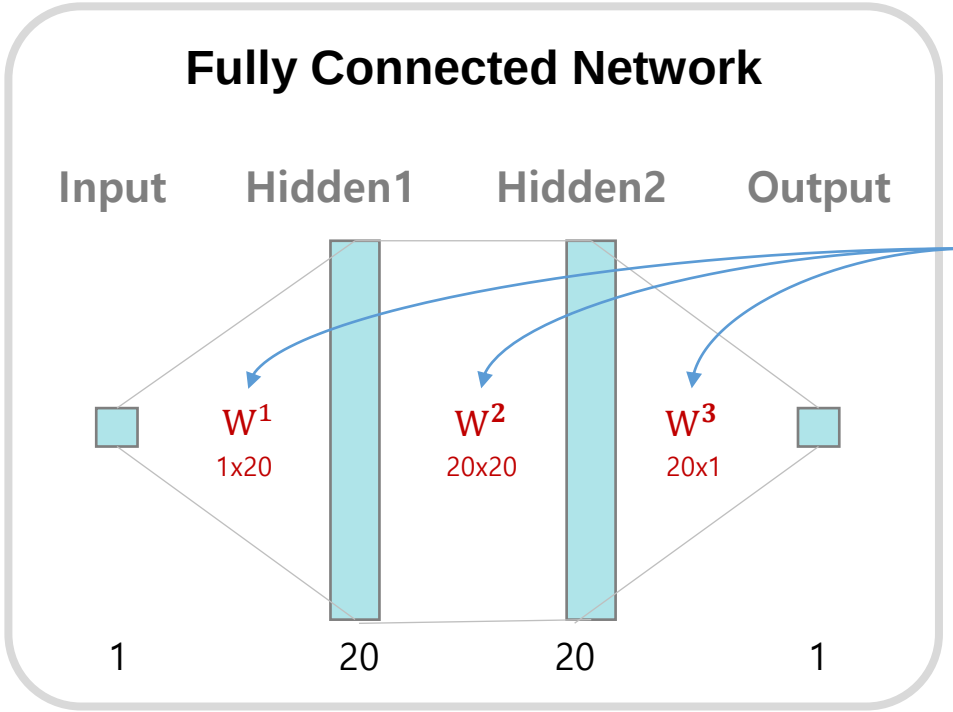
- Real-valued Scalar



Hint : Network 구성

$x = (\theta)$

- 1차원 입력 벡터

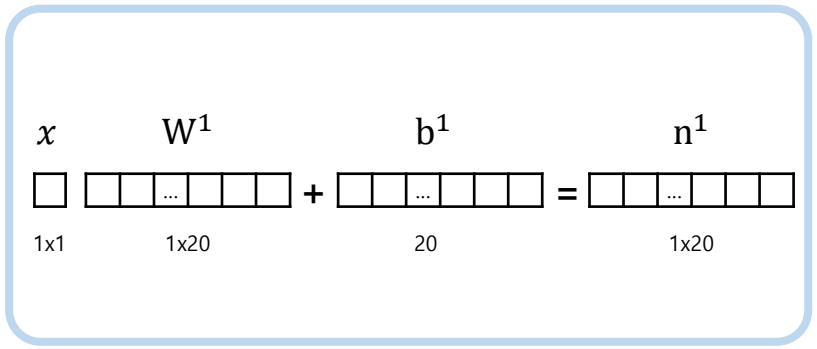


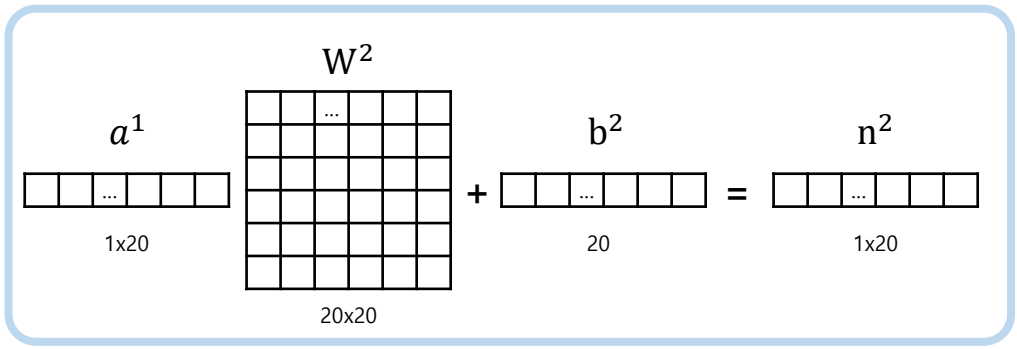
가중치 행렬의 크기는
(이전 Layer 뉴런 개수 x 현재 Layer 뉴런 개수)

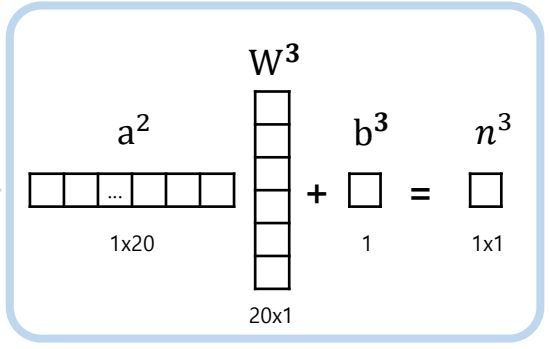
$f(\theta) = \sin(\theta)$

- Real-valued Scalar

입력 샘플이 1개 때

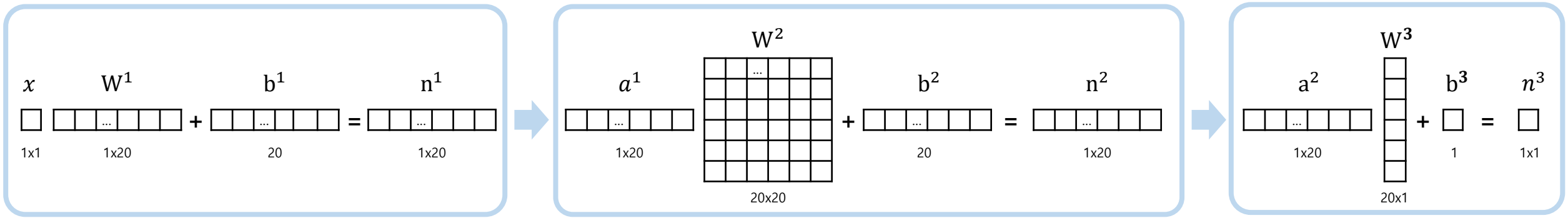






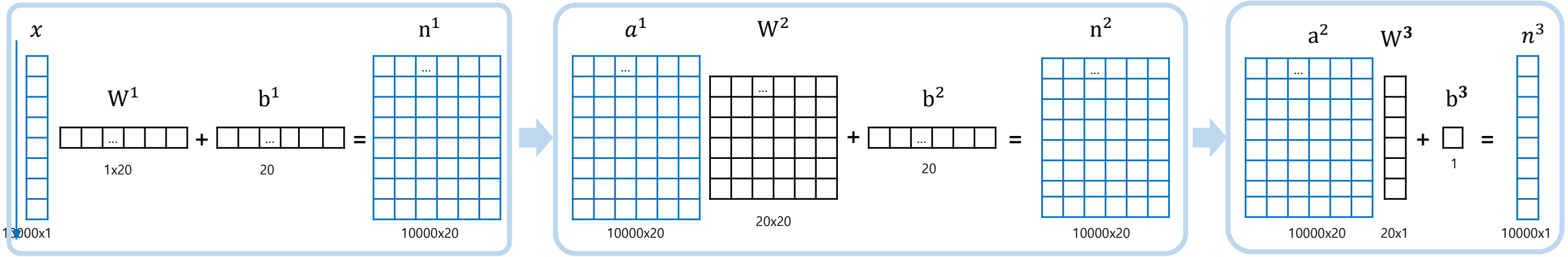
Hint : Network 구성

입력 샘플이 1개 때



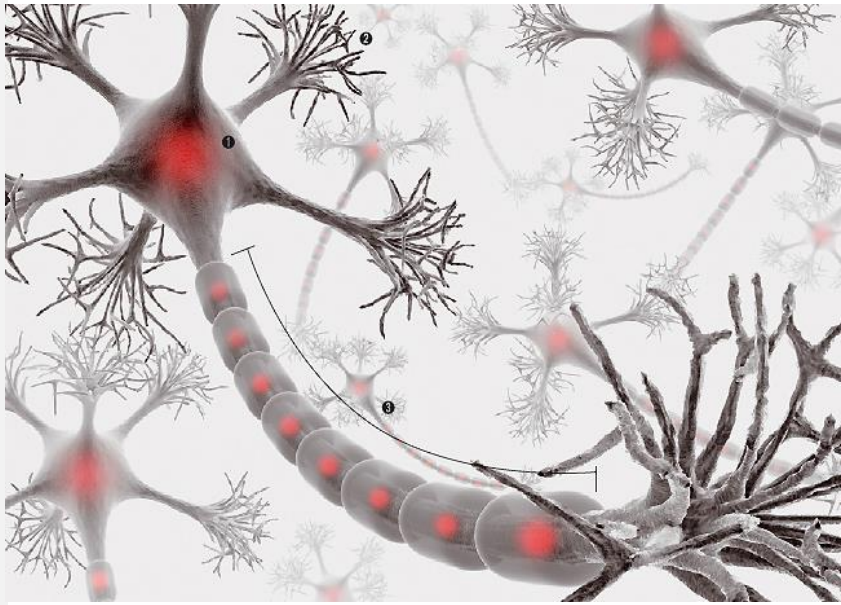
입력 배치가 10000일 때

Batch size



- 1차원이 배치 크기 : 뉴런의 입력과 출력 행렬의 1차원이 배치 크기에 따라 조정됨

2 데이터 준비



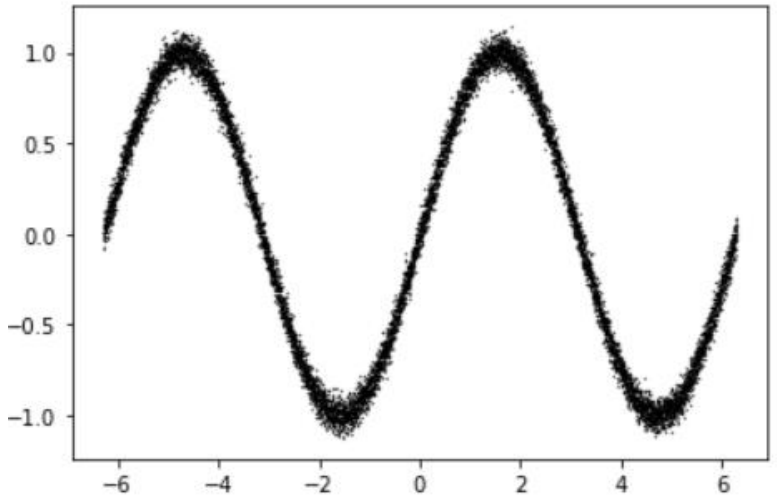
패키지 임포트

```
# tensorflow와 tf.keras를 임포트합니다
import tensorflow as tf

# 헬퍼(helper) 라이브러리를 임포트합니다
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

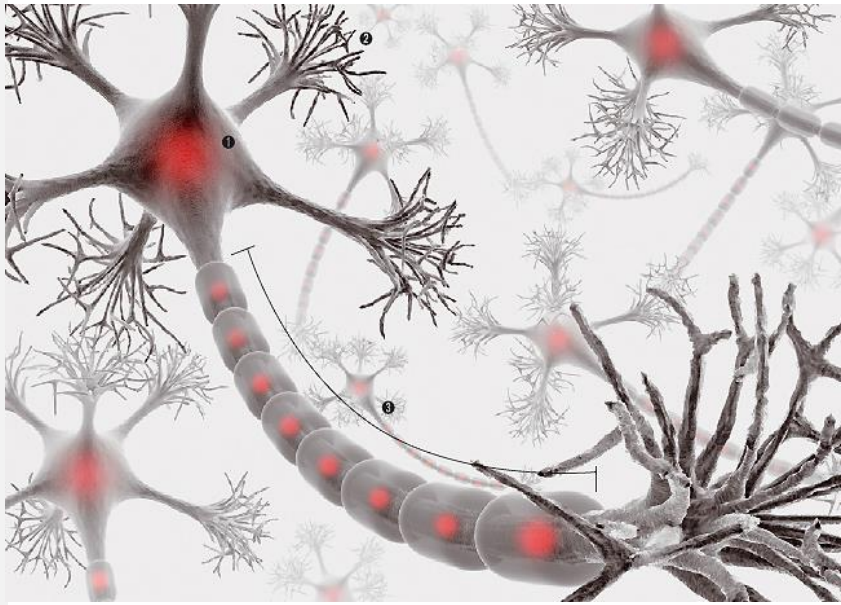
```
2.0.0-dev20190524
```

데이터셋 생성

```
# generate the data
inputs = np.linspace(-2*np.pi, 2*np.pi, 10000)[:, None]                    # inputs   (10000, 1)
outputs = np.sin(inputs) + 0.05 * np.random.normal(size=[len(inputs),1])  # outputs (10000, 1)

plt.scatter(inputs[:, 0], outputs[:, 0], s=0.1, color='k', marker='o')
```



3 모델 정의 및 훈련, 검증



모델 정의 (문제)

```
class Model(tf.Module):
    def __init__(self):
        # create variables
        initializer = tf.initializers.GlorotUniform()
        W0 = tf.Variable(initializer(shape=[1, 20]), dtype=tf.float32, name='W0')
        W1 = # your code
        W2 = # your code

        b0 = # your code        # bias는 0으로 초기화
        b1 = # your code
        b2 = # your code

        self.weights = [W0, W1, W2]
        self.biases = [b0, b1, b2]
        self.activations = [tf.nn.relu, tf.nn.relu, None]

    def __call__(self, input):
        x = input
        for W, b, activation in zip(self.weights, self.biases, self.activations):
            # affine transformation (Hint : tf. matmul를 이용해서 작성)
            x = # your code
            # activation
            if activation is not None:
                x = # your code
        return x
```

Q. 모델 생성 코드를 작성하시오.
init() 함수 : 가중치와 편향의 변수 선언
call() 함수 : 각 계층 별로 가중 합산과 활성 함수 실행 코드를 작성하시오.

모델 훈련 (문제)

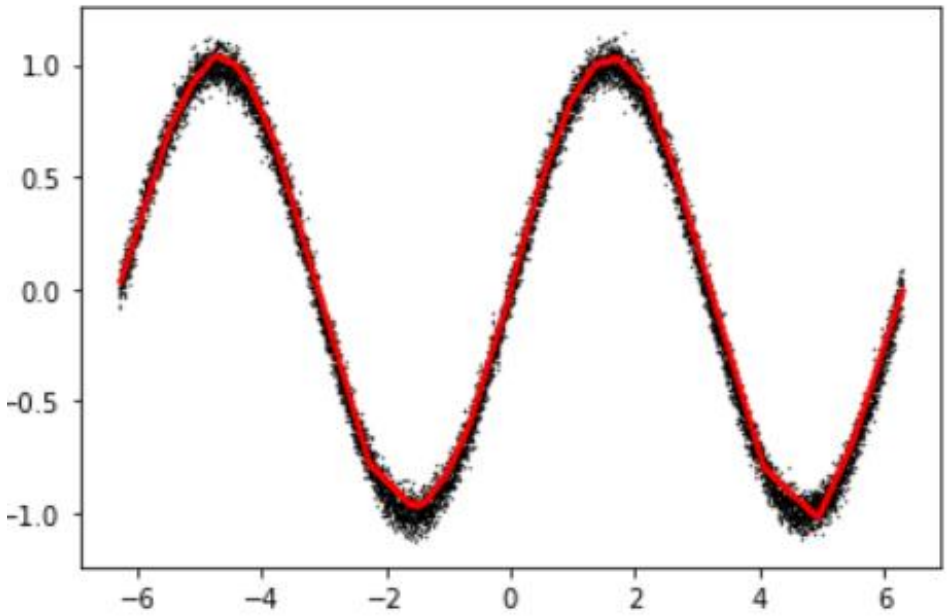
Q. GradientTape 방식으로 훈련 코드를 작성하시오.

```
model = Model()
optimizer = tf.optimizers.Adam() # create optimizer

# run training
batch_size = 32
for training_step in range(10000):
    # get a random subset of the training data
    indices = np.random.randint(low=0, high=len(inputs), size=batch_size)
    input_batch = tf.Variable(inputs[indices], dtype=tf.float32, name='input')
    output_batch = tf.Variable(outputs[indices], dtype=tf.float32, name='output')

    with tf.GradientTape() as tape:
        output_pred = model(input_batch)
        # mean squared loss (Hint : tf.reduce_mean와 tf.square를 이용해서 작성)
        mse = # your code
        # gradient 계산 (Hint : tape.gradient 사용, model.trainable_variables 사용)
        grads  = # your code
        # parameter update (Hint : optimizer.apply_gradients 사용,  model.trainable_variables 사용)
        optimizer. # your code

    if training_step % 1000 == 0:
         print('{0:04d} mse: {1:.3f}'.format(training_step, mse))
```

```
0000 mse: 2.342
1000 mse: 0.052
2000 mse: 0.020
3000 mse: 0.022
4000 mse: 0.007
5000 mse: 0.002
6000 mse: 0.004
7000 mse: 0.002
8000 mse: 0.001
9000 mse: 0.001
```

테스트

```
test_input = tf.Variable(inputs, dtype=tf.float32, name='input')
test_output = tf.Variable(outputs, dtype=tf.float32, name='output')
test_output_pred = model(test_input)

plt.scatter(inputs[:, 0], test_output[:, 0], c='k', marker='o', s=0.1)
plt.scatter(inputs[:, 0], test_output_pred[:, 0], c='r', marker='o', s=0.1)
```



Thank you!