1. 모듈 불러오기

```
import tensorflow as tf
from tensorflow.keras import layers,models,optimizers,initializers
import numpy as np
```

- layers : 다양한 신경망 레이어 정의(Dense, Conv2D, LSTM 등)
- models : 신경망 모델을 생성하고 관리(Sequential, Model)
- optimizers : 신경망의 학습과정을 최적화하는 알고리즘을 정의(SGD, Adam, RMSporp 등)
- initializers : 신경망 레이어의 가중치를 초기화하는 방법을 정의(HeNormal, GlorotUniform 등)

2. 하이퍼 파라미터 설정

```
D_i,D_k,D_o = 10,40,5
```

- D_i : 입력되는 벡터의 차원
- D_k : 히든 레이어가 출력하는 벡터의 차원
- D_o : 네트워크의 최종 출력 벡터의 차원

3. 모델 정의

```
inputs = tf.keras.Input(shape=(D_i,))
h1=layers.Dense(D_k,activation='relu',kernel_initializer=initializers.HeNormal())(inputs)
h2=layers.Dense(D_k,activation='relu',kernel_initializer=initializers.HeNormal())(h1)
outputs=layers.Dense(D_o,kernel_initializer=initializers.HeNormal())(h2)
model=tf.keras.models.Model(inputs=inputs,outputs=outputs)
```

- tf.keras.Input( ) 함수를 사용해서 입력데이터의 모양을 정의

    - shape = (A,) : 1차원의 벡터일 경우
    - shape = (A,B) : 다차원의 벡터일 경우

- layers.Dense( ) 함수를 사용해서 완전연결 레이어를 생성(히든레이어)

    - D_k : 출력 벡터의 차원을 결정.
    - activation = 'relu' : 활성화 함수로 ReLU 함수를 사용.
    - kernel_initializer = initializers.HeNormal( ) : 가중치 행렬을 초기화하는 방법을 지정.

    - tf.keras.models.Model( ) 함수 : 입력과 출력을 정의하여 신경망 모델을 생성.

        - inputs : 모델의 입력 데이터가 이 옵션을 통해서 모델에 전달됨.
        - outputs : 모델이 생성한 출력값이 이 옵션을 통해서 반환.

3. 가중치 초기화 함수

```
def weights_init(layer_in):
    if isinstance(layer_in,layers.Dense):
        layer_in.kernel.assign(initializers.HeNormal()(layer_in.kernel.shape))
        layer_in.bias.assign(initializers.Zeros()(layer_in.bias.shape))

model.layers[-1].kernel_initializer = initializers.HeNormal()
model.layers[-1].bias_initializer = initializers.Zeros()
```

- isinstance(A, B) : A객체가 B 클래스의 인스턴스인지를 확인하는 함수.
  즉, 주어진 레이어가 Dense 레이어인지 확인하는 것.
  여기서 Dense 레이어는 입력과 출력간 선형 변환만을 수행한다.

- initializers.HeNormal() : 가중치를 무작위로 초기화 하는 방법.
  Overfitting 이나 Underfitting 같은 문제들을 방지 할 수 있다.

- layers[-1] : 마지막 레이어

- model.layers[ ].kernel_initializer = initializers.HeNormal( ) :
  마지막 레이어의 가중치를 HeNormal 방법을 사용해서 초기화

- model.layers[ ].bias_initializer = initializers.Zeros( ) :
  마지막 레이어의 편향을 Zeros 방법을 사용해서 초기화. Zeros 방법은 모든 편향값을 0으로 초기화.

4. 손실 함수 및 옵티마이저 설정

```
criterion = tf.keras.losses.MeanSquaredError()
optimizer = tf.keras.optimizers.SGD(learning_rate=0.1,momentum=0.9)
```

- tf.keras.losses.MeanSquaredError() : 평균제곱오차 손실함수를 설정하는 함수.

- tf.keras.optimizers.SGD() : 확률적 경사 하강법 옵티마이저를 설정하는 함수.
  - learning_rate : 학습률
  - momentum : 비선형 모델에서 지역 최소값을 만나더라도 전역 최소값을 탐색하는데 도움을 주는 옵션.
    값이 클수록 이전 기울기 업데이트의 영향이 커지며, 모델의 학습이 안정화되고
    더 빠르게 수렴할 수 있다. 범위는 0 < momentum < 1.

5. 학습률 스케줄러 설정

```
scheduler =tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=0.1,
    decay_steps=10,
    decay_rate=0.5,
    staircase=True
)
```

- tf.keras.optimizers.schedules.ExponentialDecay(): 지수감소 학습률 스케줄러를 설정하는 함수.
    - initial_learning_rate : 초기 학습률을 설정하는 옵션.
    - decay_step : 에포크마다 학습률이 감소되는 빈도.
    - decay_rate : 학습률이 감소 되는 비율.
    - staircase : 학습률을 계단식으로 감소시키는 옵션. True면 계단식으로, False면 지수적으로 감소한다.
    - decay_step에 도달하기 전까지는 학습률을 유지하다, decay_steps를 넘어서면 decay_rate 만큼 학습률을 감소 시킨다.
    - 에포크(epoch) : 모델이 전체 훈련 데이터셋을 한 번 학습한 것.

6. 데이터 준비

```
x = np.random.randn(100,D_i)
y = np.random.randn(100,D_o)
dataset=tf.data.Dataset.from_tensor_slices((x,y)).batch(10).shuffle(buffer_size=100,seed=42)
```

- np.random.randn(a,b) : 크기가 a,b인 난수로 이루어진 입력 데이터 행렬을 생성.

- tf.data.Dataset.from_tensor_slices() : 입력 데이터와 출력 데이터를 하나의 데이터셋으로 묶는 함수이며, 데이터를 메모리에 로드하지 않고, 텐서 슬라이스를 생성하여 효율적으로 데이터를 처리한다.

- .batch(a) : 데이터셋을 미니배치로 나누는 함수. 입력과 출력 데이터를 a개씩 묶어서 학습하는 것을 의미.

- .shuffle(buffer_size = a, seed = b) :
a는 섞을 때 사용할 버퍼의 크기를 지정하는 것이고, seed는 재현성을 위한 값.

- buffer : 데이터를 저장하거나 처리하는데 사용되는 임시 메모리 공간.

7. 학습 루프

```
for epoch in range(100):
    epoch_loss =0.0
    for step,(x_batch,y_batch)in enumerate(dataset):
        with tf.GradientTape()as tape:
            pred = model(x_batch,training=True)
            loss = criterion(y_batch,pred)
        grads =tape.gradient(loss,model.trainable_variables)
        optimizer.apply_gradients(zip(grads,model.trainable_variables))

        epoch_loss +=loss.numpy()

    print(f'Epoch {epoch:5d}, loss {epoch_loss:.3f}')
```

- enumerate( ) : 각 요소의 인덱스와 해당 요소를 함께 반환하는 것을 의미.
    - dataset : tf.data.Dataset 객체로, 일반적으로 배치로 나누어진 데이터셋을 의미 한다.
    - batch : 데이터셋을 작은 단위로 나눈 것.

- tf.GradientTape() : 자동미분을 위해 사용되는 함수.

- training = True : 학습중에만 적용되는 레이어들이 올바르게 적용되도록 하는 옵션.

- tape.gradients( ) : 컨텍스트 내에서 손실 함수에 대한 모델의 학습 가능한 변수들의 기울기를 계산하는 함수.
    - model.trainable_variables : 모델 내에서 학습 가능한 모든 변수들을 포함하는 리스트.

- optimizer.apply_gradient( ) : 기울기와 변수의 쌍을 입력으로 받아 변수를 업데이트 하는 함수.
  zip( ) : 주어진 두 개 이상의 시퀀스(리스트, 튜플 등)를 병렬로 묶어주는 함수.

- loss.numpy( ) : 현재 배치의 손실을 배열로 반환하는 함수.