# 주가예측실습(RNN, GRU, LSTM)

임세진

https://youtu.be/LQVzX4e4shc

HSU 한성대학교 HANSUNG UNIVERSITY

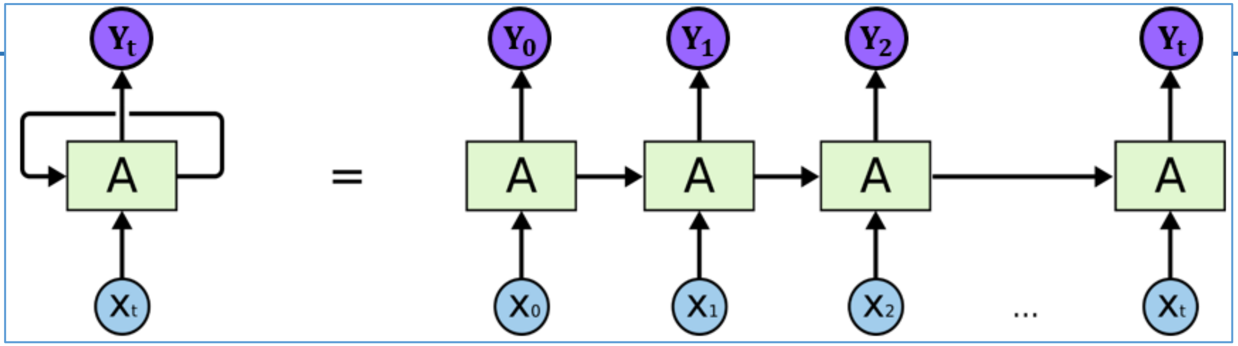
CryptoCraft LAB
https://crypto.modoo.at

# Contents



CryptoCraft LAB

# 01. GRU



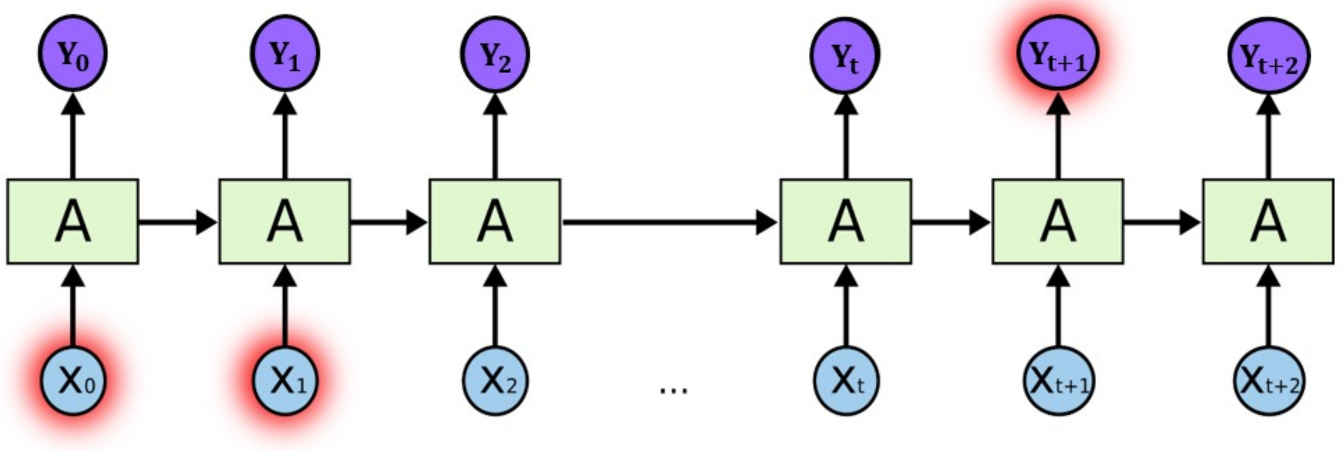

- Vanilla RNN (Recurrent Neural Network)

✓가장 단순한 형태의 RNN

✓이전 셀이 계산되어야만 현재 셀이 계산되므로 동작 속도 느림

✓비교적 짧은 시퀀스에서 효과적

✓장기 의존성 문제 (Long-Term Dependencies)



# 01. GRU

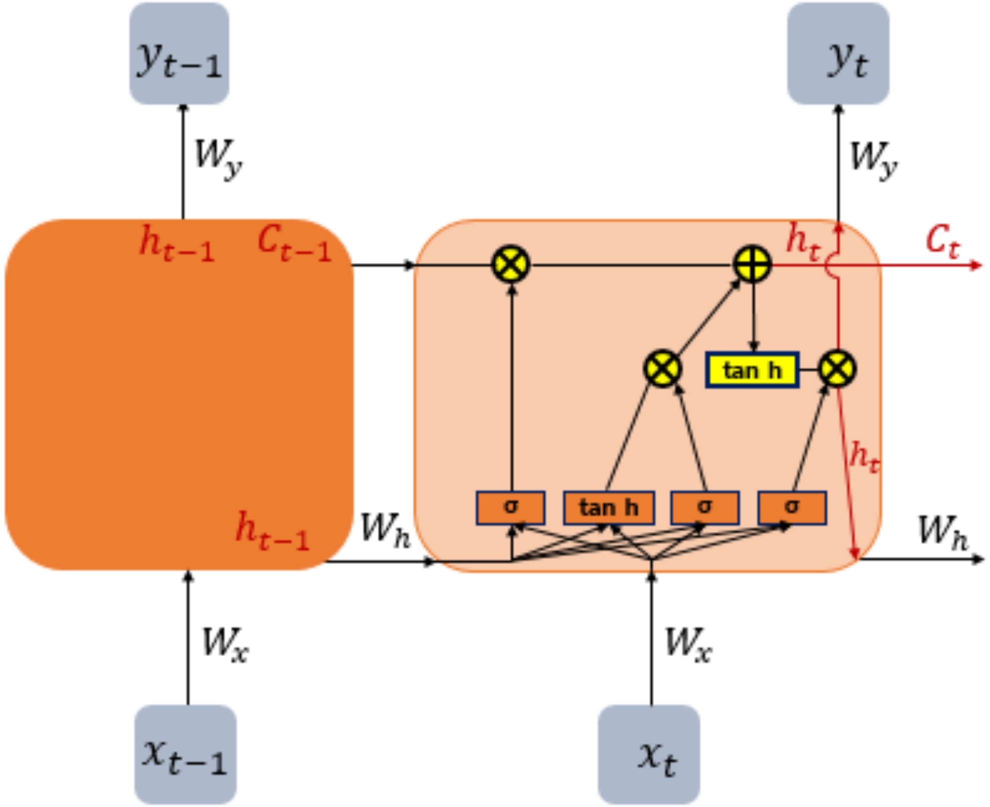
- LSTM (Long Short-Term Memory)

✓ Vanilla RNN의 단점 보완 ➔ 긴 시퀀스의 입력에 대해 높은 성능

✓ Cell state라는 값을 추가하여 Hidden Layer의 메모리 셀에 input, output, forget GATE를 추가

➔ 불필요한 기억을 지우고 기억해야할 것들을 정함

✓ hidden state를 계산하는 과정이 복잡해짐



# 01. GRU

- GRU (Gated Recurrent Unit)

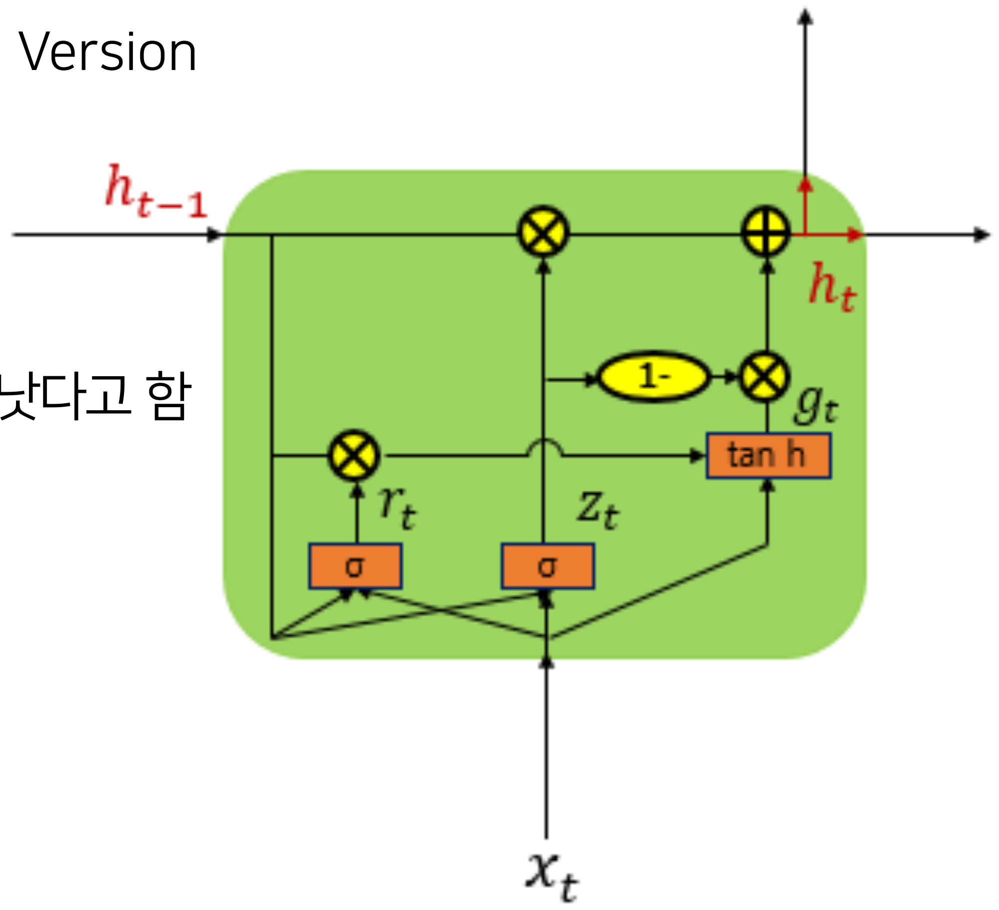
✓LSTM의 성능을 유지하면서 복잡했던 LSTM의 구조를 간단화 시킨 Version

✓hidden state를 업데이트하는 계산을 줄임

➔Update, Reset GATE만 존재

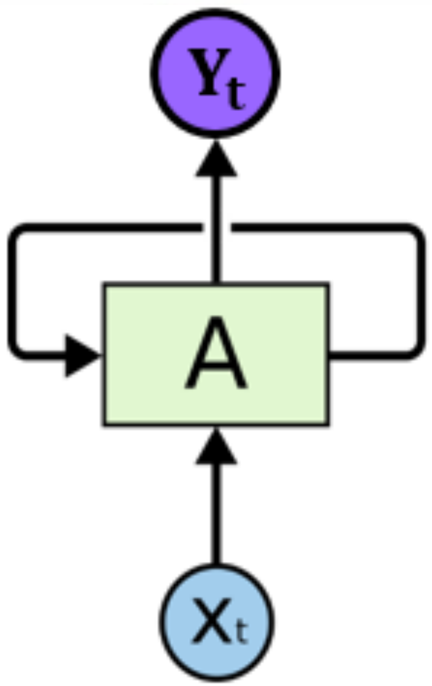
✓데이터 양이 적을 때는 GRU가, 데이터 양이 많을 때는 LSTM이 더 낫다고 함

✓복잡도 : RNN < GRU < LSTM



# 02. RNN in PyTorch

outputs
shape = ( -, -, - )

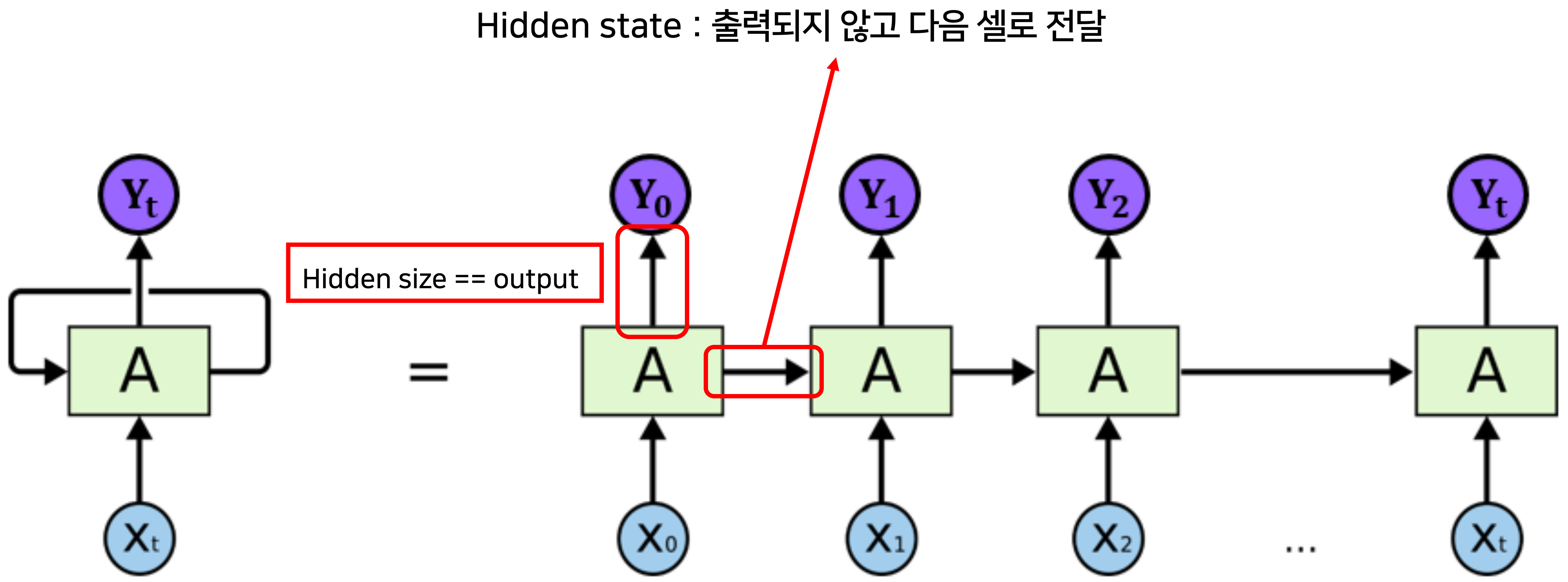



```
rnn = torch.nn.RNN(input_size, num_layers, hidden_size)
```

```
output, _status = rnn(input_data)
```

Input_data
shape = ( batch_size, sequence_length, input_size )

# 02. RNN in PyTorch



# 02. RNN in PyTorch

<코스피 주가예측>

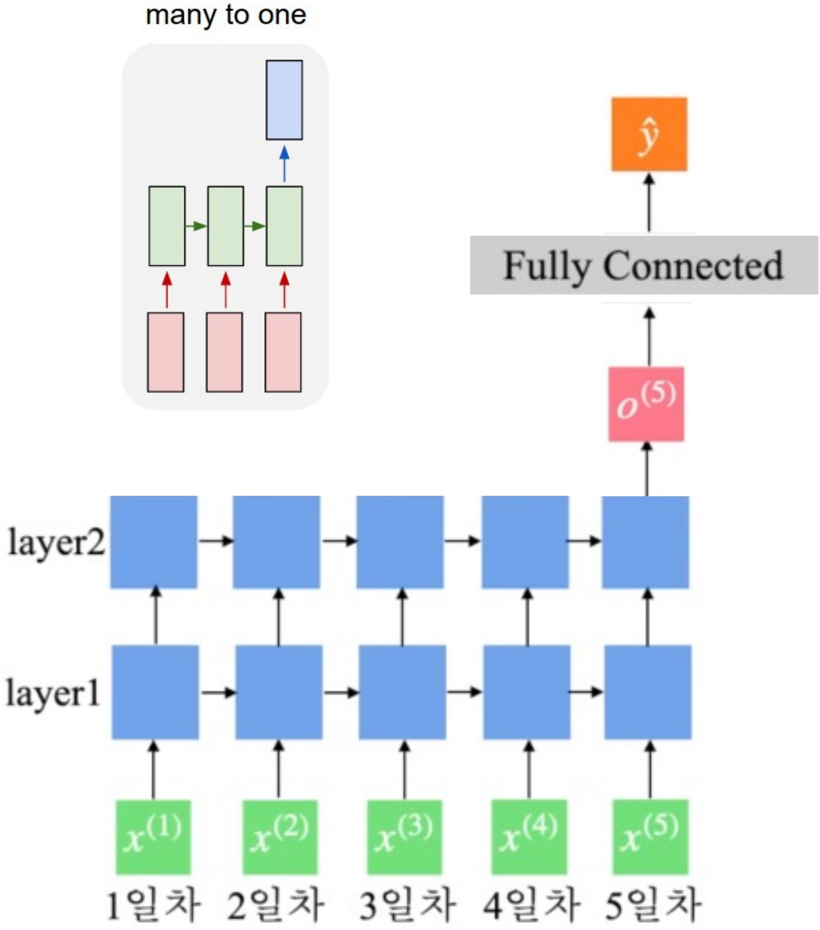
- Fully Connected ? **Hidden size == output ?**

```
# RNN에서 필요한 정보
input_size = 4 # x_seq.size(2)
num_layers = 2
hidden_size = 8
```

```
class VanillaRNN(nn.Module):

    def __init__(self, input_size, hidden_size, sequence_length, num_layers, device):
        super(VanillaRNN, self).__init__()
        self.device = device
        self.hidden_size = hidden_size # 각 일마다 나오는 값의 크기
        self.num_layers = num_layers
        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Sequential(nn.Linear(hidden_size, 1), nn.Sigmoid())
        # 여기서 1은 "한개의 값". 마지막 예측하는 하루를 의미

    def forward(self, x):
        out, _ = self.rnn(x)
        out = out[:,-1] # many to one
        out = self.fc(out)
        return out
```

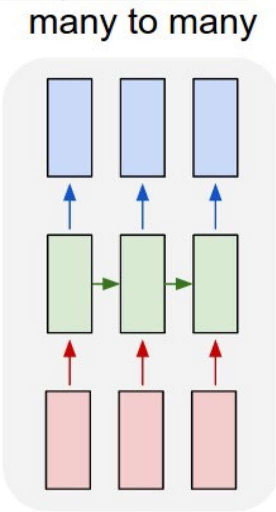
output.size()

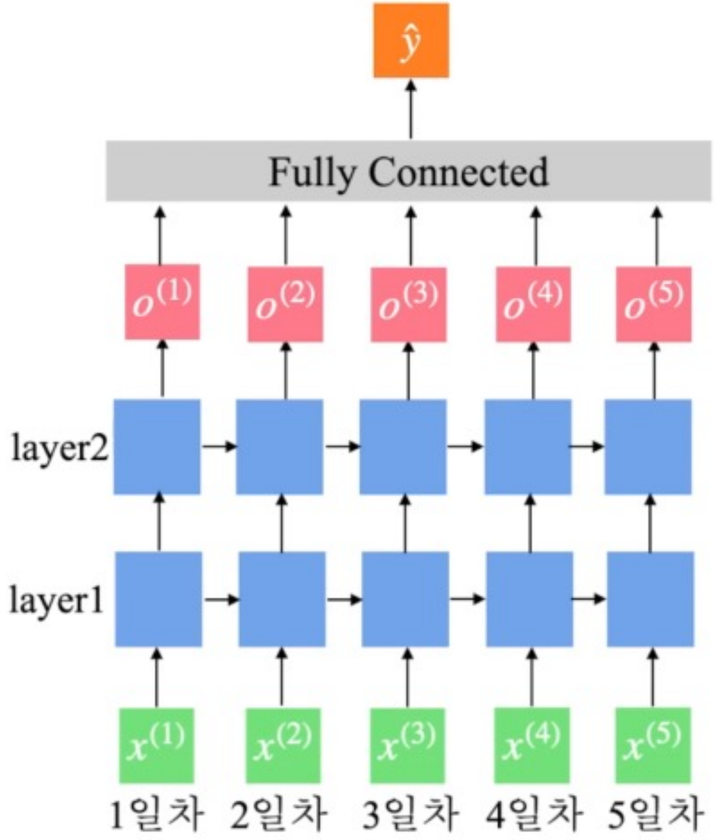
➔ [batch_size, sequence_length, hidden_size]



# 02. RNN in PyTorch

```
class VanillaRNN(nn.Module):

    def __init__(self, input_size, hidden_size, sequence_length, num_layers, device):
        super(VanillaRNN, self).__init__()
        self.device = device
        self.hidden_size = hidden_size # 각 일마다 나오는 값의 크기
        self.num_layers = num_layers
        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Sequential(nn.Linear(hidden_size*sequence_length, 1), nn.Sigmoid())
        # 총 40개의 노드 (각 일마다 8개씩 5일)
        # 여기서 1은 "한개의 값". 마지막 예측하는 하루를 의미

    def forward(self, x):
        out, _ = self.rnn(x)
        out = out.reshape(out.shape[0], -1) # many to many 전략
        out = self.fc(out)
        return out
```



# 03. GRU in PyTorch

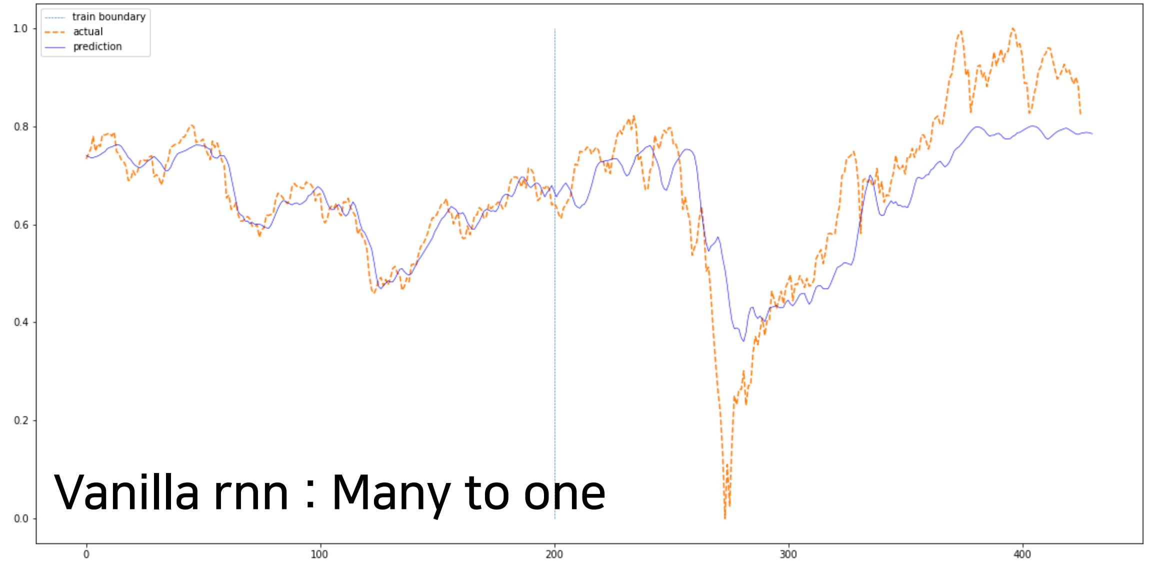
```
class GRU(nn.Module):

    def __init__(self, input_size, hidden_size, sequence_length, num_layers, device):
        super(GRU, self).__init__()
        self.device = device
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.gru = nn.GRU(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size*sequence_length, 1)

    def forward(self, x):
        out, _ = self.gru(x)
        out = out.reshape(out.shape[0], -1)
        out = self.fc(out)
        return out
```

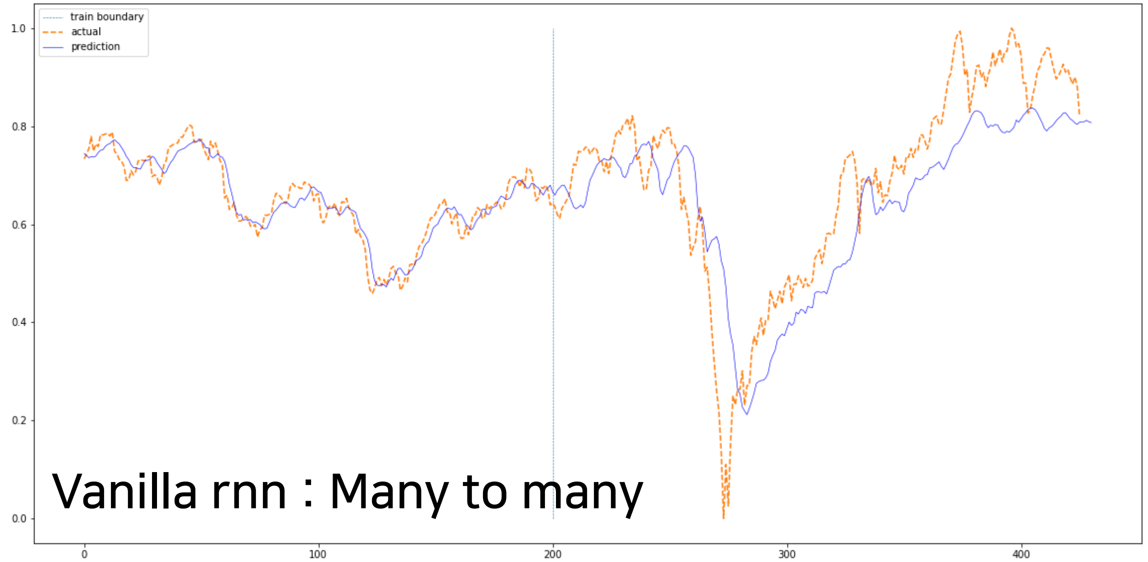
# 04. LSTM in PyTorch

```
class LSTM(nn.Module):

    def __init__(self, input_size, hidden_size, sequence_length, num_layers, device):
        super(LSTM, self).__init__()
        self.device = device
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size*sequence_length, 1)

    def forward(self, x):
        out, _ = self.lstm(x)
        out = out.reshape(out.shape[0], -1) # <- state 추가
        out = self.fc(out)
        return out
```

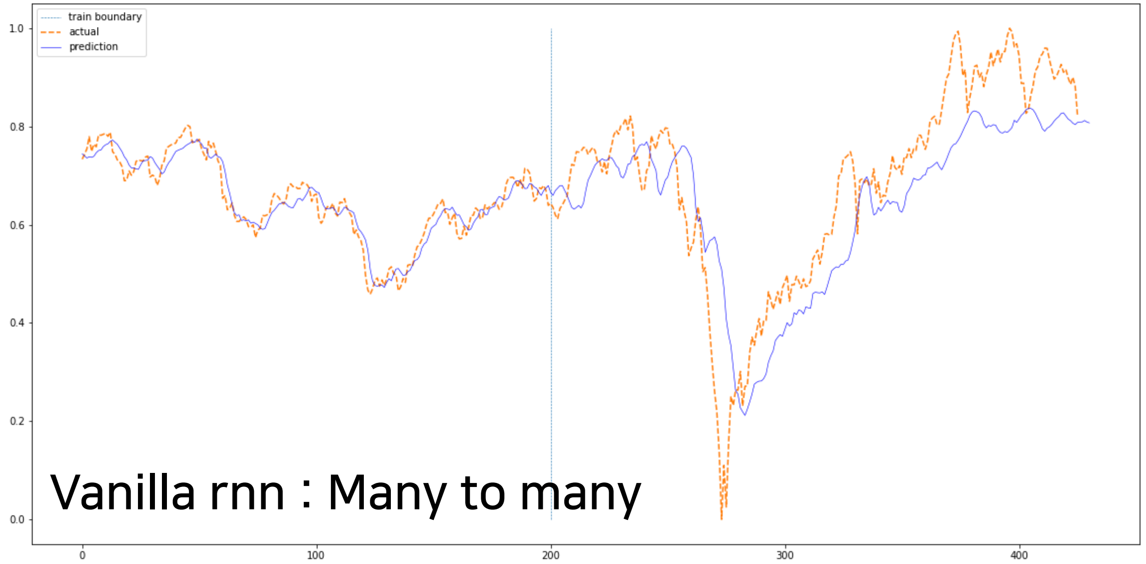
# 04. LSTM in PyTorch

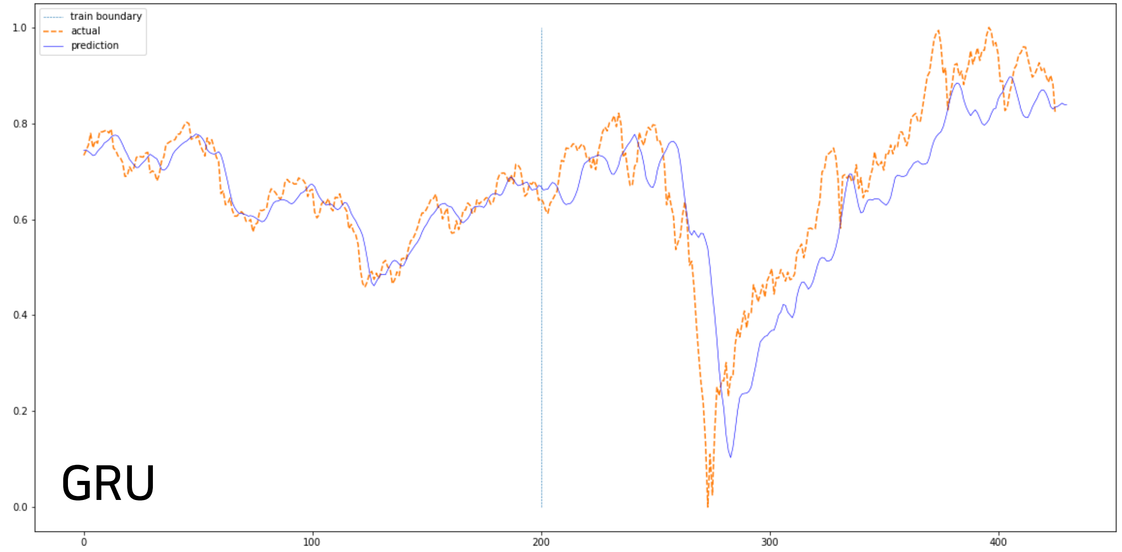
<성능비교>

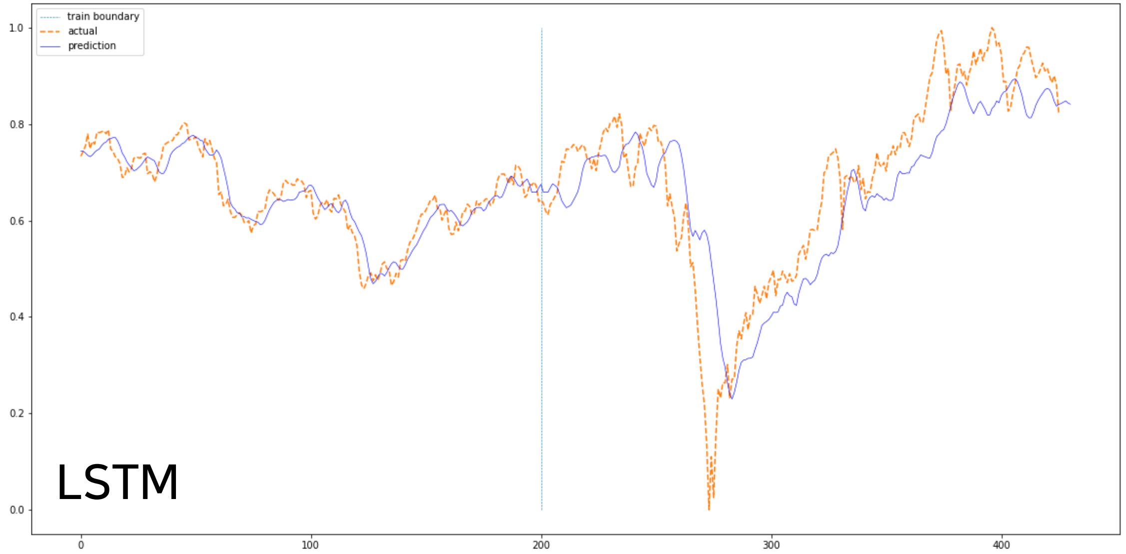


# 04. LSTM in PyTorch



감사합니다