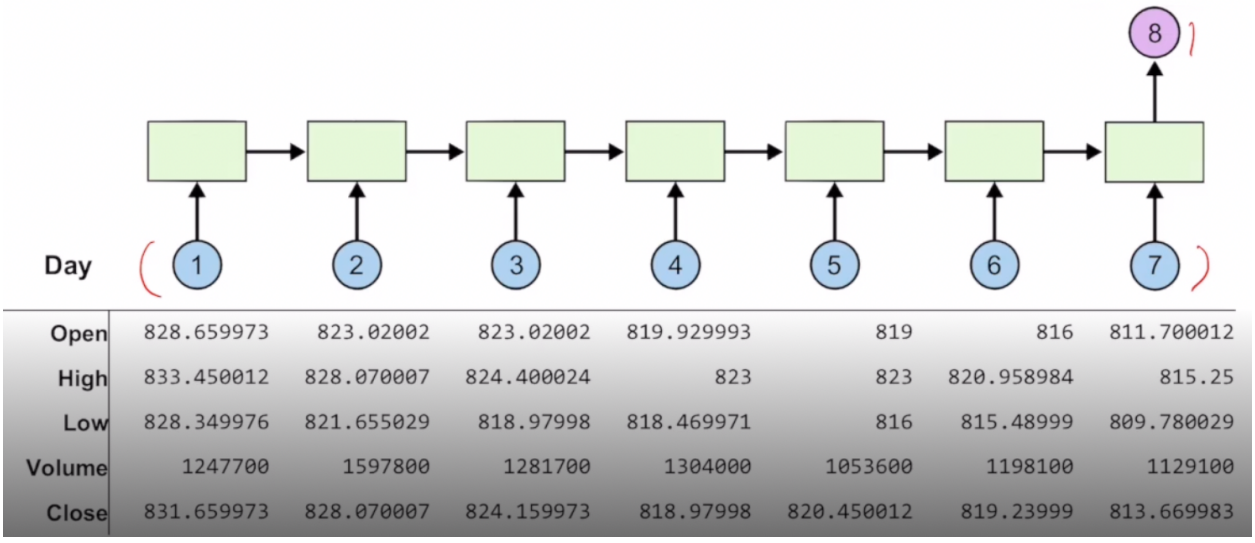
RNN Timeseries

상태	RNN
담당자	

Many-to-One 응용 사례



Open	828.659973	823.02002	823.02002	819.929993	819	816	811.700012
High	833.450012	828.070007	824.400024	823	823	820.958984	815.25
Low	828.349976	821.655029	818.97998	818.469971	816	815.48999	809.780029
Volume	1247700	1597800	1281700	1304000	1053600	1198100	1129100
Close	831.659973	828.070007	824.159973	818.97998	820.450012	819.23999	813.669983

주식 종가 예측이 대표적인 many-to-one RNN이 사용되는 사례이다. 7개의 input을 바탕으로 8일차의 주가를 예측하는 것이 과제이다.

이때 8일차의 output은 종가이기 때문에 1 dimension일 것이다. 이를 위해서는 이전 단계의 모든 hidden state 역시 1 dimension이 되어야 한다.

Hidden state의 dimension 보장

Input data로는 5 dimension의 텐서가 들어오게 되는데, 최종적으로 1 dimension output을 출력하기 위해 hidden state들은 매번 input data들을 압축해야 하는 부담을 가지게 된다.

> 💡 데이터를 유통하기 위한 hidden state의 dimension을 충분히 보장해 주자!

Hidden state의 dimension을 충분히 보장해 주고, 마지막 output 직전에 FC layer를 연결해 구성하도록 하자.

PyTorch

하이퍼파라미터

```
# hyper parameters
seq_length = 7 # 7일
data_dim = 5 # 5차원
hidden_dim = 10 # 임의 정의
output_dim = 1 # 1차원 종가 (fc layer가 맞춰야 하는)
learning_rate = 0.01
iterations = 500
```

데이터 준비

```
xy = np.loadtxt("data-02-stock_daily.csv", delimiter=",")
xy = xy[::-1] # 데이터를 시간 역순으로 ordering

train_size = int(len(xy) * 0.7) # 데이터의 70%를 train set으로 이용
train_set = xy[0:train_size] # train set
test_set = xy[train_size - seq_length:] # test set
```

이렇게 작성한 train, test set에 scaling을 적용한다. 예를 들어 주가, 거래량은 800, 1000000 등 그 scale이 너무 다른데, 이를 모두 0과 1 사이의 값으로 바꾸어 주는 것이다.

▼ `minmax_scaler`

```
# scaling function for input data
def minmax_scaler(data):
    numerator = data - np.min(data, 0)
    denominator = np.max(data, 0) - np.min(data, 0)
    return numerator / (denominator + 1e-7)
```

```
train_set = minmax_scaler(train_set)
test_set = minmax_scaler(test_set)
```

▼ `build_datasest`

```
# make dataset to input
def build_dataset(time_series, seq_length):
    dataX = []
    dataY = []
    for i in range(0, len(time_series) - seq_length):
        _x = time_series[i:i + seq_length, :]
        _y = time_series[i + seq_length, [-1]]  # Next close price
        print(_x, "->", _y)
        dataX.append(_x)
        dataY.append(_y)
    return np.array(dataX), np.array(dataY)
```

7일간의 입력, 8일자의 종가로 나누어 파이토치가 읽을 수 있는 형태로 변형해 준다.

RNN

```
class Net(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, layers):
        super(Net, self).__init__()
        self.rnn = torch.nn.LSTM(input_dim, hidden_dim, num_layers=layers, batch_first=True)
        self.fc = torch.nn.Linear(hidden_dim, output_dim, bias=True)

    def forward(self, x):
        x, _status = self.rnn(x)
        x = self.fc(x[:, -1])
        return x


net = Net(data_dim, hidden_dim, output_dim, 1) # input dim, hidden dim, output dim, num of layer
```

Criterion, loss 정의

```
criterion = torch.nn.MSELoss()
optimizer = optim.Adam(net.parameters(), lr = learning_rate)
```

학습

```
# start training
for i in range(iterations):

    optimizer.zero_grad()
    outputs = net(trainX_tensor)
    loss = criterion(outputs, trainY_tensor)
    loss.backward()
    optimizer.step()
    print(i, loss.item())
```

```
plt.plot(testY)
plt.plot(net(testX_tensor).data.numpy())
plt.legend(['original', 'prediction'])
plt.show()
```