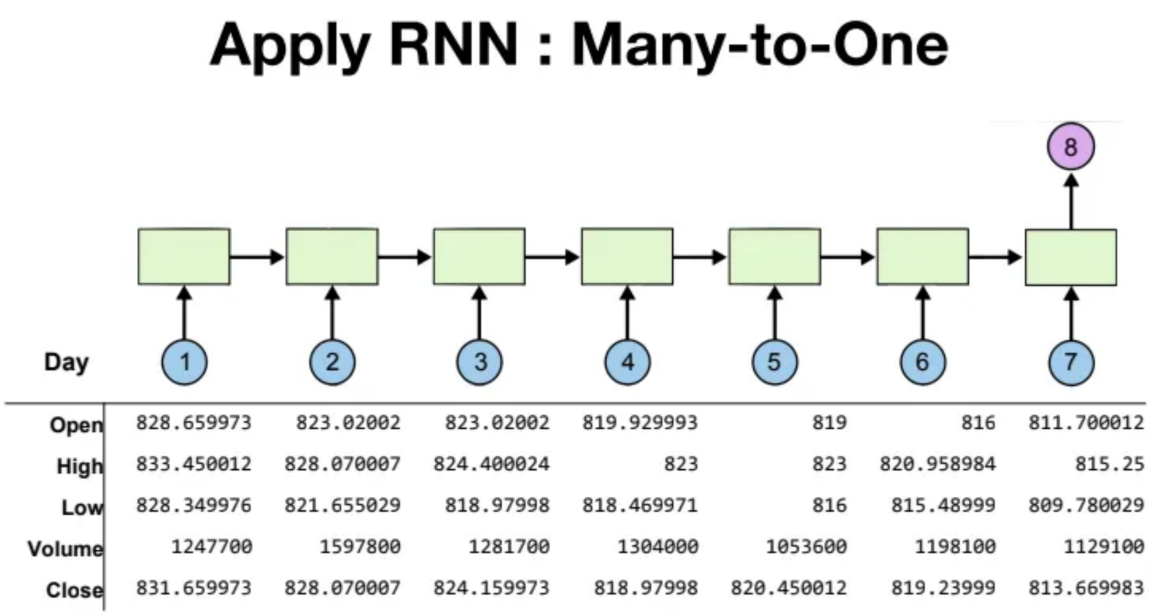
exam 개인학습

Lec 28

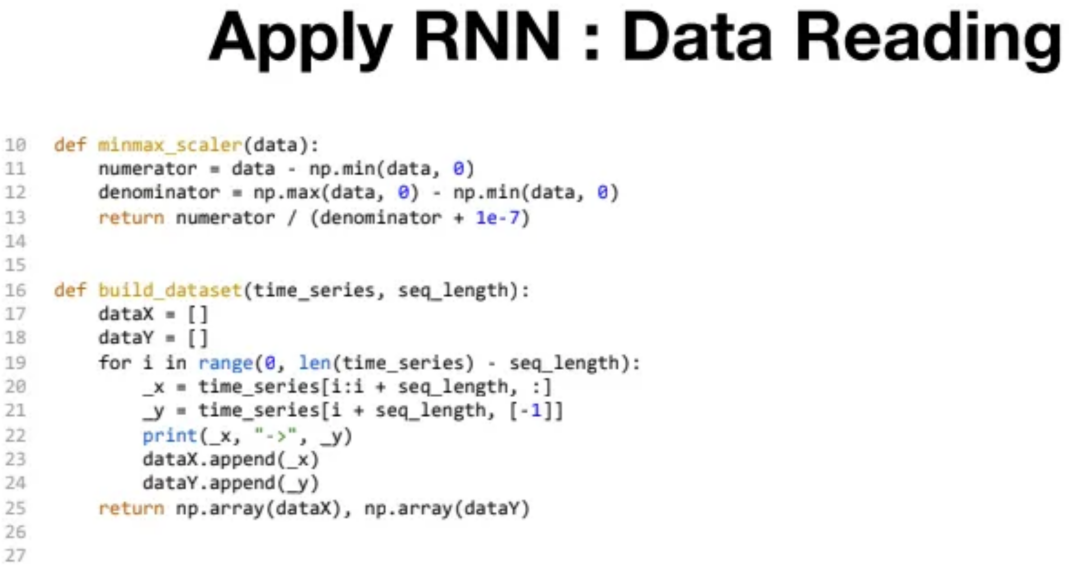
Type something...

01) RNN timeseries

01_Time Series Data



- `open` , `high` , `low` , `volume`, `close` 5개의 변수를 가진 시계열 데이터를 가지고 8일차 주식의 가격을 맞추는 RNN 모델을 만들어 볼 것이다.



- 데이터를 보면 다른 데이터에 비해 `volume` 의 데이터가 매우 크기 때문에 `minmax_scaler` 를 통해서 scale를 조정해준다.

- data를 `seq_length` 만큼 잘라서 저장해주는 함수를 만들어보자.

# http://finance.yahoo.com/quote/GOOG/history?it=1				
# Open	High	Low	Volume	Close
828.659973	833.450012	828.349976	1247700	831.659973
823.02002	828.070007	821.655029	1597800	828.070007
819.929993	824.400024	818.97998	1281700	824.159973
819.359985	823	818.469971	1304000	818.97998
819	823	816	1053600	820.450012
816	820.958984	815.48999	1198100	819.23999
811.700012	815.25	809.780029	1129100	813.669983
809.51001	810.659973	804.539978	989700	809.559998
807	811.840027	803.190002	1155300	808.380005
803.98999	810.5	801.780029	1235200	806.969971
799.700012	801.669983	795.25	1174200	801.340027
802.98999	806	800.369995	1460400	801.48999
793.799988	802.700012	792	1525800	798.530029
799.679993	801.190002	791.190002	2023300	795.695007
796.859985	801.25	790.52002	2143500	796.789978
814.659973	815.840027	799.799988	3228900	802.320007
834.710022	841.950012	820.440002	2951800	823.309998
837.809998	838	827.01001	2734400	832.150024
829.619995	835.77002	825.059998	1494500	835.669983
822.299988	825.900024	817.820984	1461000	823.869995
807.25	820.869995	803.73999	1901600	819.309998

- 데이터가 2차원 데이터임을 주의하자.
- `y` 에는 다음날 close price가 저장되어 있다.

- indexing을 할 때 `iloc` 를 붙여서 하자.

```
[47]  # split train-test set
      train_size = int(len(xy) * 0.7)
      train_set = xy[0:train_size]
      test_set = xy[train_size - seq_length:]

      # scaling data
      train_set = minmax_scaler(train_set)
      test_set = minmax_scaler(test_set)

      # make train-test dataset to input
      trainX, trainY = build_dataset(train_set, seq_length)
      testX, testY = build_dataset(test_set, seq_length)

      # convert to tensor
      trainX_tensor = torch.FloatTensor(trainX)
      trainY_tensor = torch.FloatTensor(trainY)

      testX_tensor = torch.FloatTensor(testX)
      testY_tensor = torch.FloatTensor(testY)
```

- train 70% / test 30% 로 나누어 학습을 진행한다.

```
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
```

```
[49]  net = Net(data_dim, hidden_dim, output_dim, 1)
```

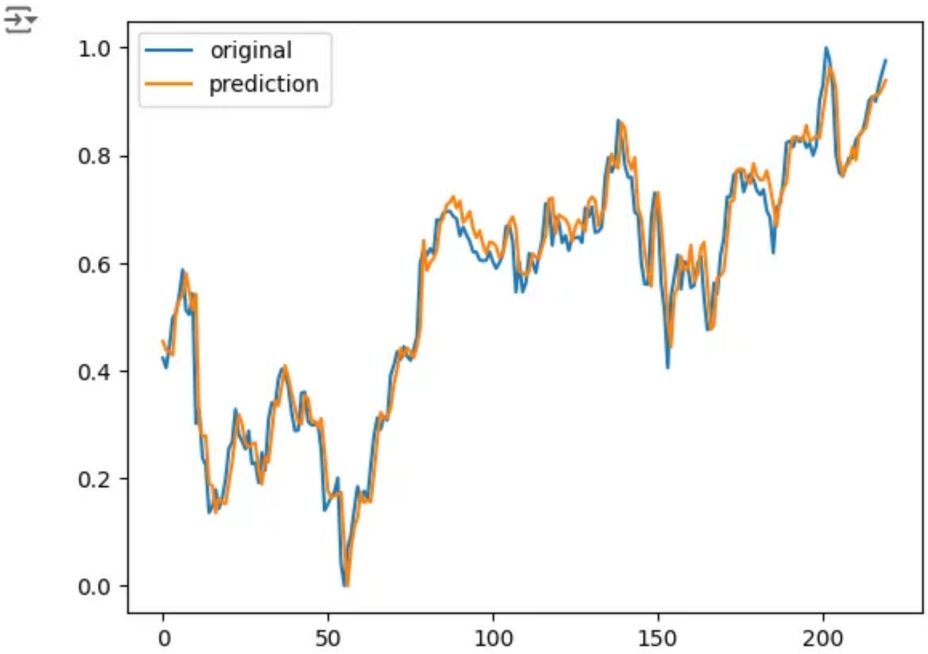
```
[50] # loss & optimizer setting
     criterion = torch.nn.MSELoss()
     optimizer = optim.Adam(net.parameters(), lr=learning_rate)
```

```
[51] # 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())
```

- 모델은 LSTM을 사용했는데 추후에 LSTM에 대해 자세히 포스팅 하겠다.
- 나머지 학습방법은 동일하므로 생략하겠다.

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



- test와 prediction을 붙여서 그려봤더니 생각보다 잘 예측하는 것을 확인할 수 있다.