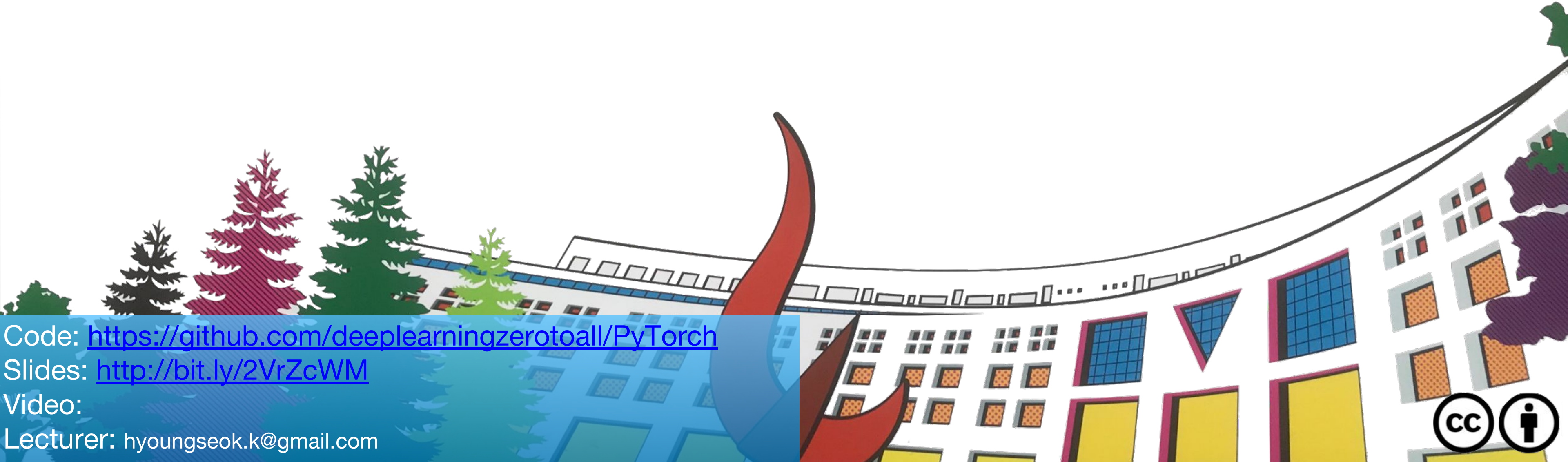
ML/DL for Everyone Season2

with PYTORCH

RNN - Time Series



Code: https://github.com/deeplearningzerotoall/PyTorch

Slides: http://bit.ly/2VrZcWM

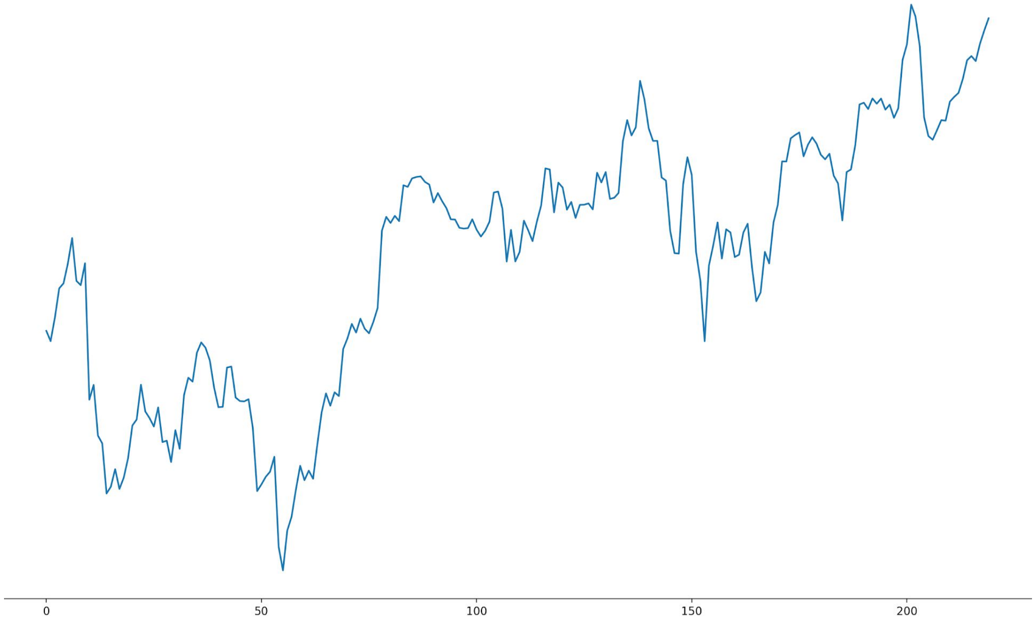
Video:

Lecturer: hyoungseok.k@gmail.com

RNN - Time Series

- Time Series Data
- Apply RNN
    - Many-to-One
    - Data Reading
    - Neural Net Setting
    - Training & Evaluation
- Exercise

Time Series Data

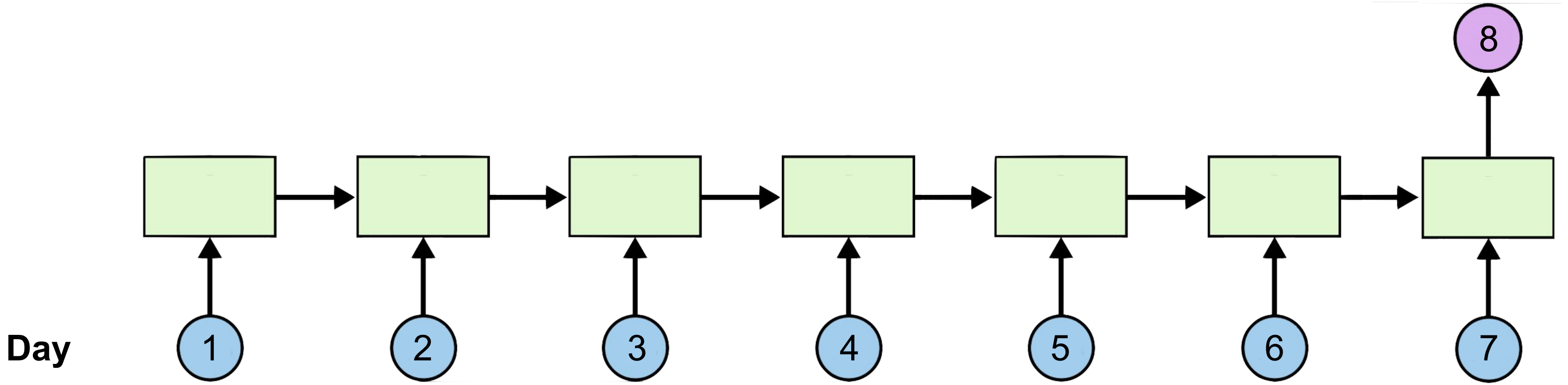


Example : GOOG

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**

data-02-stock_daily.csv

Apply RNN : Many-to-One



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

Apply RNN : Data Reading

```python
import torch
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt

# Random seed to make results deterministic and reproducible
torch.manual_seed(0)
```

```python
seq_length = 7
data_dim = 5
hidden_dim = 10
output_dim = 1
learning_rate = 0.01
iterations = 500

xy = np.loadtxt("data-02-stock_daily.csv", delimiter=",")
xy = xy[::-1]  # reverse order

train_size = int(len(xy) * 0.7)
train_set = xy[0:train_size]
test_set = xy[train_size - seq_length:]

```

Apply RNN : Data Reading

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

trainX, trainY = build_dataset(train_set, seq_length)
testX, testY = build_dataset(test_set, seq_length)

trainX_tensor = torch.FloatTensor(trainX)
trainY_tensor = torch.FloatTensor(trainY)

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

Apply RNN : Data Reading

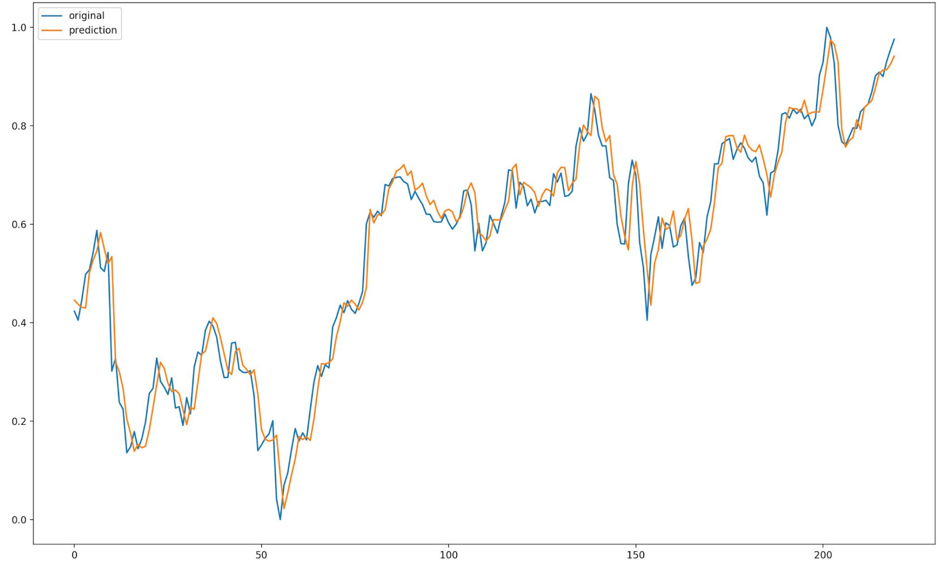
```python
def minmax_scaler(data):
    numerator = data - np.min(data, 0)
    denominator = np.max(data, 0) - np.min(data, 0)
    return numerator / (denominator + 1e-7)


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]]
        print(_x, "->", _y)
        dataX.append(_x)
        dataY.append(_y)
    return np.array(dataX), np.array(dataY)


```

Apply RNN : Neural Net Setting

```python
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)

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

```

Apply RNN : Training & Evaluation

```
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()
```



Exercise

- Implement stock prediction right now?
- Use more features to improve robustness

What’s Next?

- Train Seq2Seq model in PyTorch