Implementation of Seq2Seq on Time Series Data

Team 1
나하원 배인원 유진이 이다영 이여진이
준석

May 2022

Contents

1. Introduction

Introduction

현업에서 시계열 데이터 예측에 대한 관심과 필요성이 증대됨에 따라 시계열 데이터를 중심으로 예측 모델을 구현

시계열 데이터 예측 모델을 적용할 수 있는 예:

- 제품의 수요예측을 통한 재고 최적화
- 주식 및 가상화폐 가격 예측을 통한 수익극대화 등

시간의 순서에 따라 움직이는 시계열 데이터에 적합한 효율적인 딥러닝 모델을 찾기 위한 비교 진행

2. DNN Dataset

DNN Dataset

Data
- Bitcoin
- S&P500 지수

Data Time Interval
- 1 hour

Data Type
- Close Price

DNN Dataset - Bitcoin

DNN 요구사항: 2차원 데이터

24시간 거래 특성 반영, 한국시간 기준 0시~24시 사이의 데이터를 1일 데이터로 X데이터 구성

예측값: 한국시간 기준 익일 오전 1시 Close Price를 사용

24시간 기준 가격 예측

DNN Dataset - S&P500

미국 거래소 운영시간인 9:30 ~ 16:00 사이의 데이터를 1일 데이터로 X 데이터 구성

예측값: 일별 Close Price 사용

Time Stamp + 1 시점에 이전 데이터의 Y값이 들어가지만, Time Stamp가 다르므로 모델 학습에 문제 없을 것으로 예상

8시간 기준 가격 예측

DNN Dataset - Code

```
x_test = test[:-24]
y_test = test[24:]
print(x_test.shape,y_test.shape)
```

```
(48, 1) (48, 1)
```

```
def windowDataset(data,label,window_size=24):
    feature_list = []
    label_list = []
    for i in range(len(data) - window_size):
        feature_list.append(np.array(data[i:i+window_size]))
        label_list.append(np.array(label.iloc[i]))
    return np.array(feature_list), np.array(label_list)
```

3. DNN

DNN - 비교

간단한 모델과 고도화된 모델 학습 및 비교 결과, 간단한 모델의 성능이 우위

고도화된 모델의 경우 하이퍼파라미터의 최적화가 관건

```
model = keras.models.Sequential([
    Dense(512,activation='swish',kernel_initializer='random_normal'),
    Dense(256,activation='swish'),
    Dense(128,activation='swish'),
    Dense(64,activation='swish'),
    Dense(1)
])
```

DNN - Loss function

MAPE
Mean Absolute Percentage Error

$$\mathrm{MAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|$$

A: Actual Value F: For ecast Value

RMSE
Root Mean Square Error

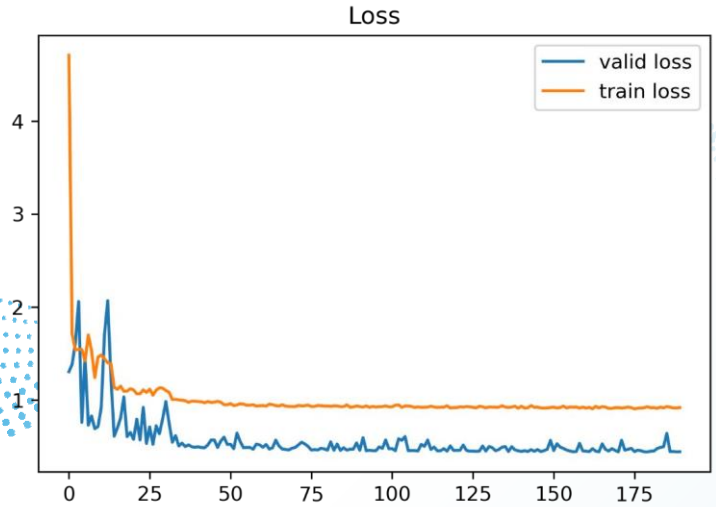
$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \|y(i) - \hat{y}(i)\|^2}{N}},$$

y(i) : Actual Value ŷ (i) : Forecast Value

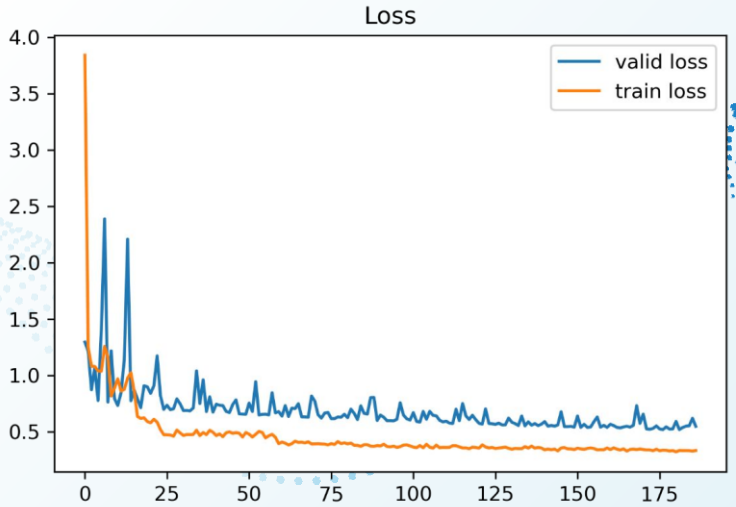
DNN - Loss

Validation Loss, Train Loss < 1.0%

BTC

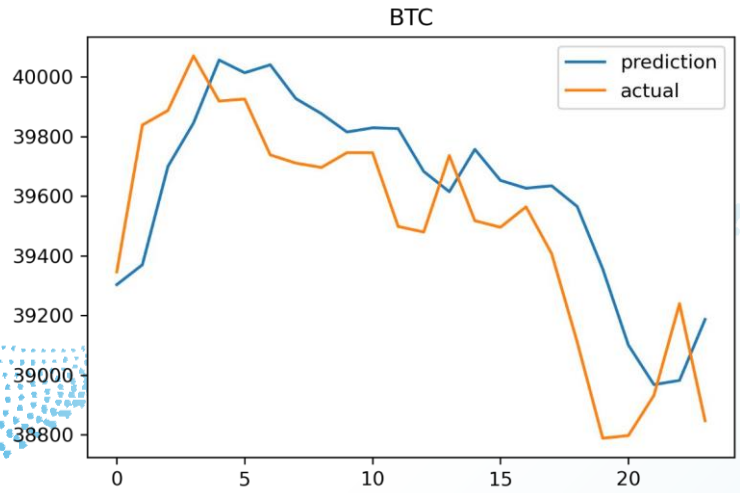


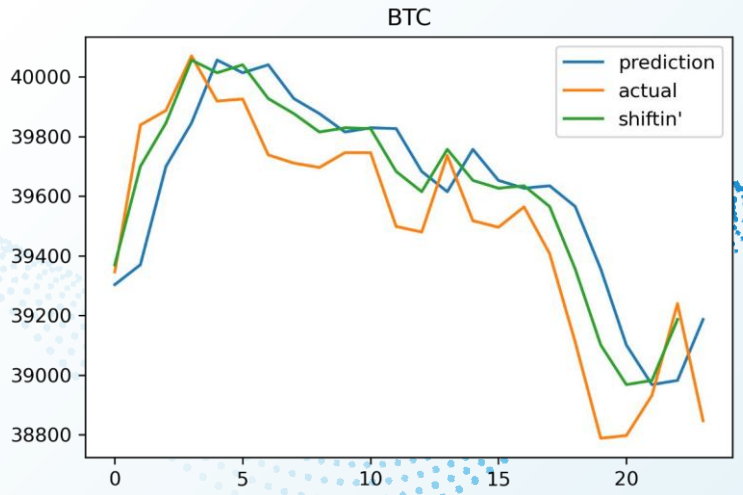
S&P500



DNN - BTC Result

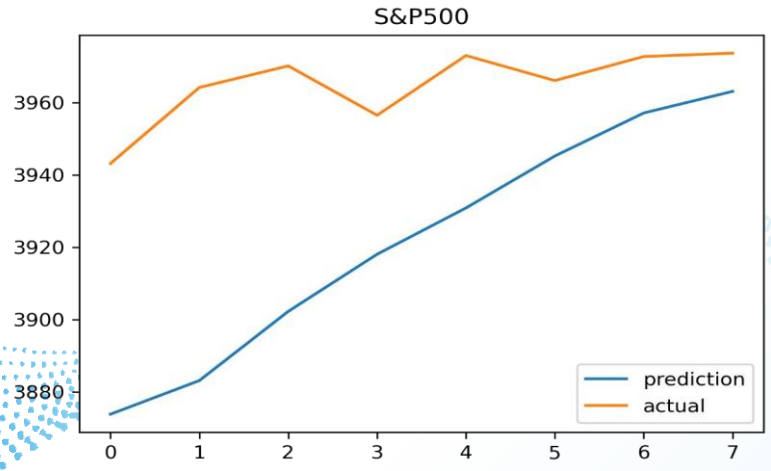
BTC 훈련 결과, 예측값이 실제값으로 Shifting
실제 Shifting한 결과, 추세선의 유사성 확인

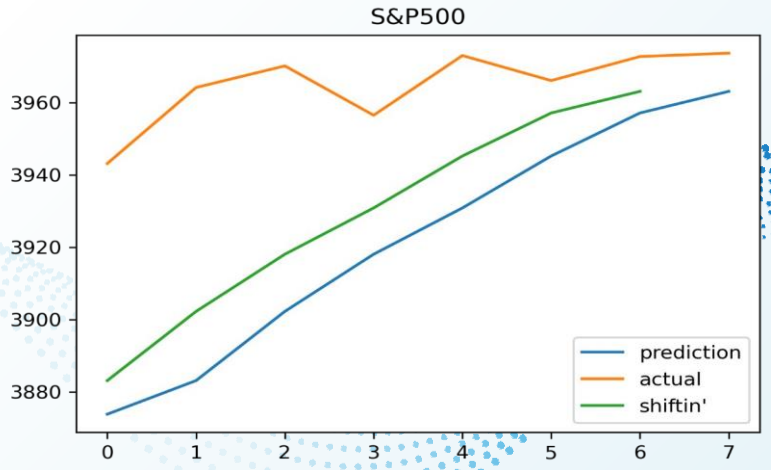




DNN - S&P500 Result

RMSE = 50.1
MAPE = 1.09%





4. LSTM Dataset

LSTM - Dataset

DL은 고차원의 데이터셋 구성이 가능하여 ML적용과는 다른 차원의 데이터셋을 적용

Test set의 경우, 총 8시간의 데이터가 가운데 차원을 구성, Feature의 개수는 예측값의 종류가 Close Price 한 개 임에 따라 1개의 Feature만 적용

```
X_test,y_test=make_dataset(X_test,y_test,8)
print(X_test.shape,y_test.shape)
```

```
(8, 8, 1) (8, 1)
```

```
def make_dataset(data,label,window_size=8):
    feature_list=[]
    label_list=[]
    for i in range(len(data)-window_size):
        feature_list.append(np.array(data[i:i+window_size]))
        label_list.append(np.array(label.iloc[i]))
    return np.array(feature_list), np.array(label_list)
```

5. LSTM

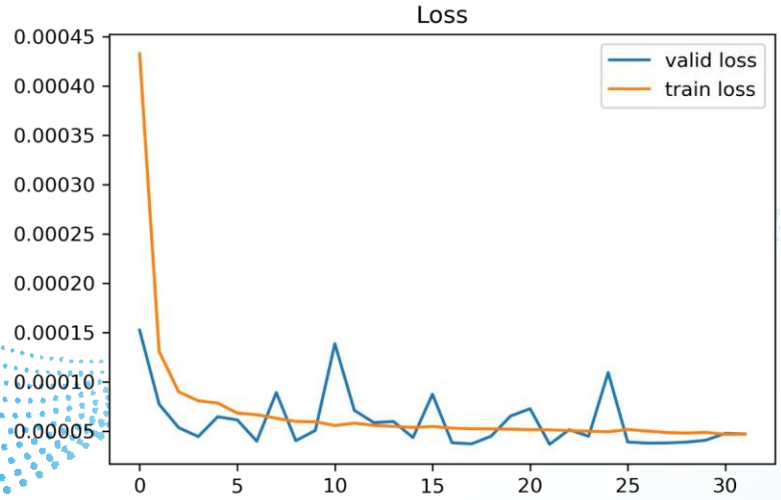
LSTM - Modelling

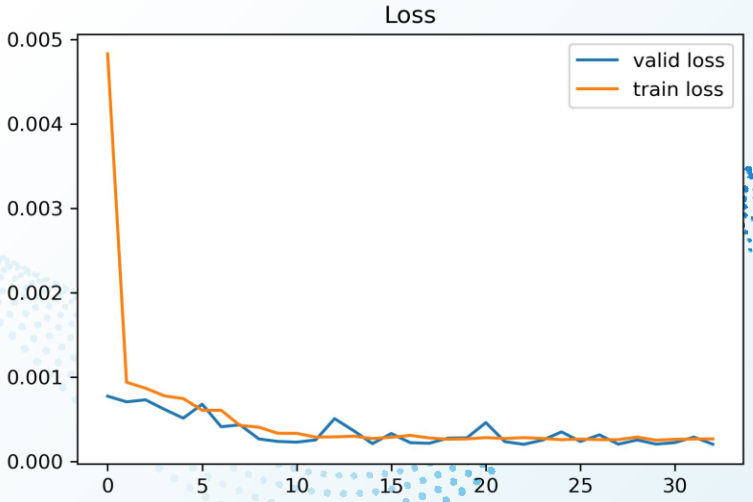
```
model=tf.keras.models.Sequential([
    tf.keras.layers.LSTM(64,return_sequences=True,input_shape=(X_train.shape[1],1)),
    tf.keras.layers.LSTM(32,return_sequences=False),
    tf.keras.layers.Dense(16),
    tf.keras.layers.Dense(1)
])
```

LSTM - 훈련 결과

Loss function: MSE Loss

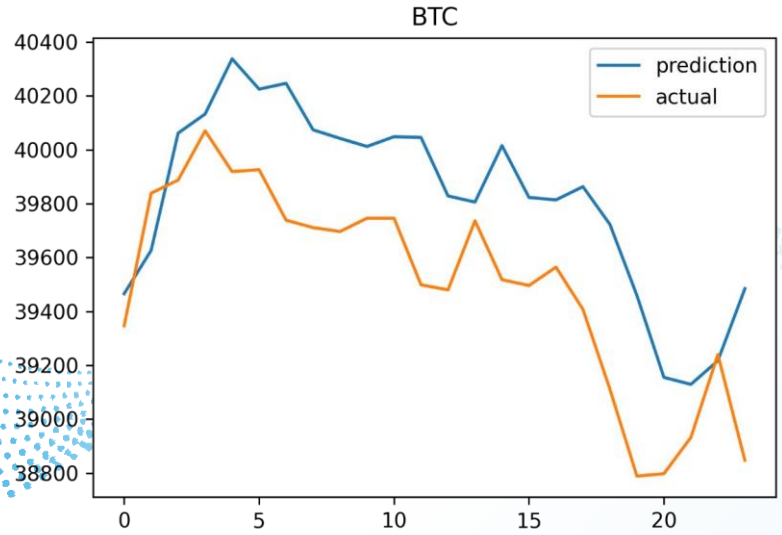
< 0.001

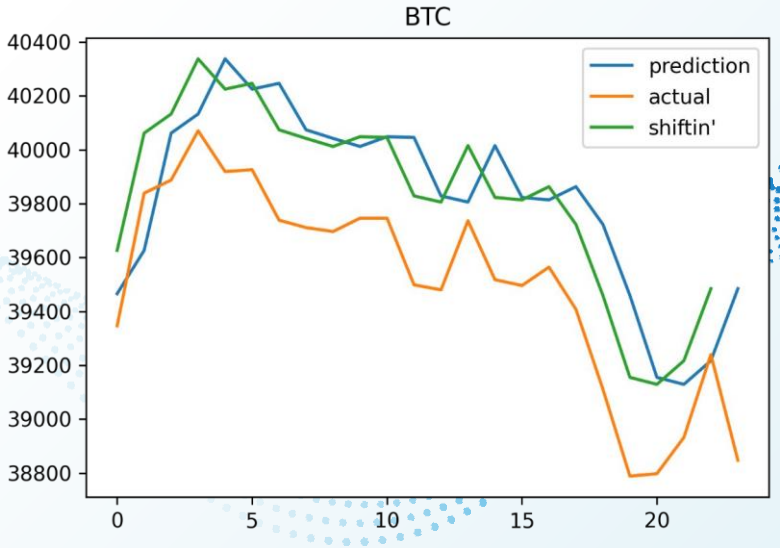




LSTM - BTC

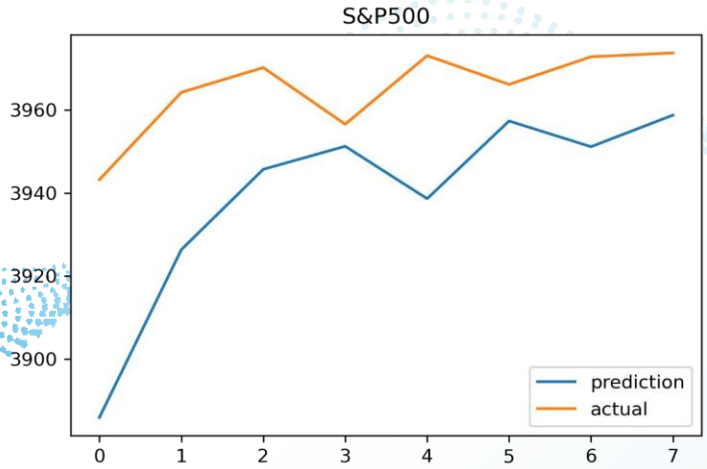
RMSE = 380.53
MAPE = 0.85%

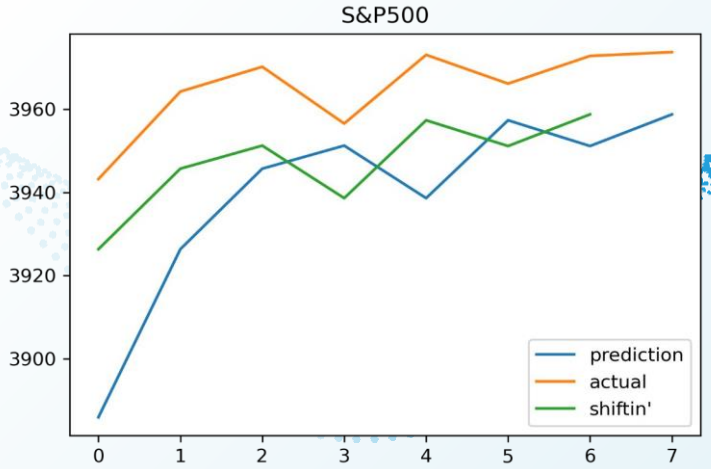




LSTM - S&P500

RMSE = 30.23 MAPE = 0.65%
예측값이 실제값에 대해 Shifting하는 경향
-> Network가 Test Data를 Mimicking하는 것으로 추측
-> multi-step forecast를 활용한 Seq2Seq 모델로 문제 해결 시도



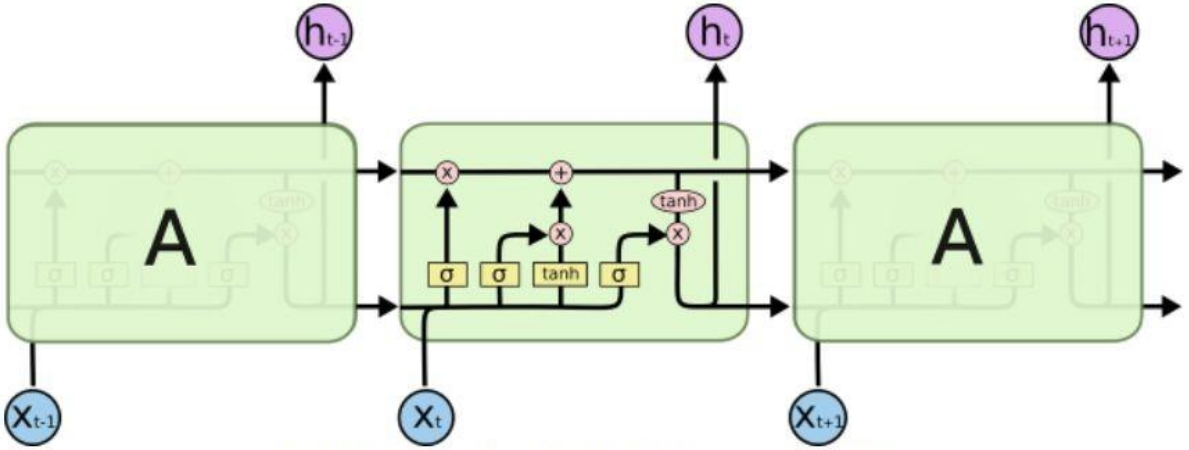


6. Seq2Seq

Seq2Seq (LSTM의 구조)

output: hidden state & cell state

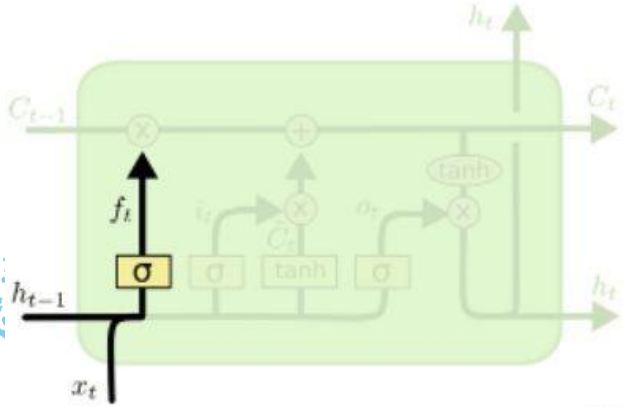
다음 time stamp 시점의 layer 로 두 개의 output 이 그대로 전달



LSTM의 반복 모듈에는 4개의 상호작용하는 layer가 들어있다.

Seq2Seq (LSTM-forget gate)

- t-1의 hidden state와 t의 x데이터를 통해 weight, bias를 계산
- Sigmoid 적용 > 0~1 사이 결과값
- 0~1 사이의 결과값과 t-1 cell state 값의 행렬곱을 통해 기억 유지 정도를 결정 (0: forget, 1: keep)

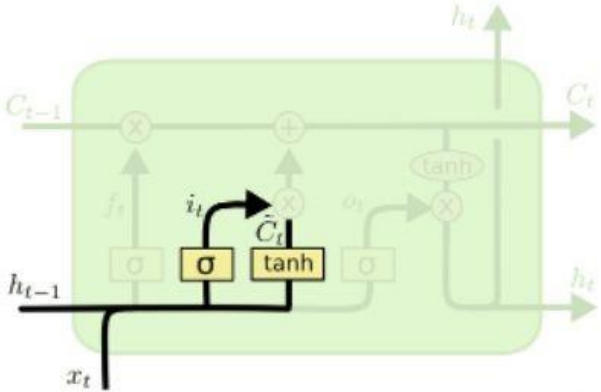


LSTM의 forget gate layer

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Seq2Seq (input gate & cell state)

- Hidden state(t-1) & X(t) > sigmoid > 0~1
- tanh applied Cell state(t)
- 두 값의 행렬곱
- Forget gate 지난 C(t-1) 과 위의 행렬곱 값의 합 = Cell state output



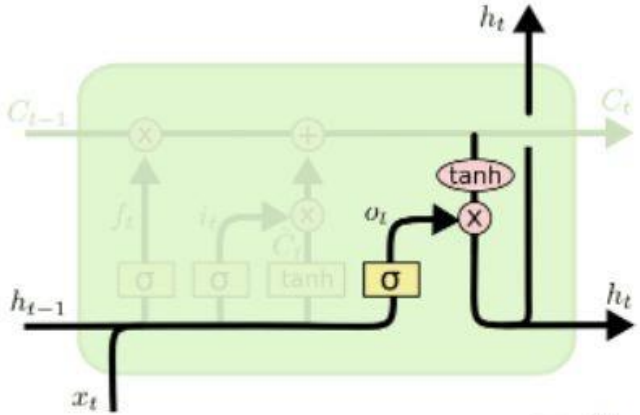
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

LSTM의 input gate layer

Seq2Seq (LSTM-output gate)

- Input gate와 forget gate가 공통으로 받는 값은 C(t-1) & x(t)
- x(t)로 weight, bias를 구한 뒤, sigmoid를 취해 output으로 내보낼 부분을 결정
- tanh applied cell state와 행렬곱
- Output으로 의도한 일부분만 출력



LSTM의 output gate layer

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

7. Seq2Seq Model

Seq2Seq-model-Conv1d

BTC 데이터에 대한 Encoder 부분의 Conv1d 모델

우선적으로 24시간 차원을 구성
-> input value= 24

Dilation: data interval
-> 가격을 건너뛰면서 학습 시 값 예측에 효과적일 수 있다는 판단

```
class Encoder(nn.Module):
    def __init__(self,input_size,hidden_size):
        super(Encoder,self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size

        self.conv1 = nn.Conv1d(24,128,1,dilation=1)
        self.bn1 = nn.BatchNorm1d(128)
        self.conv2 = nn.Conv1d(128,256,1,dilation=2)
        self.bn2 = nn.BatchNorm1d(256)
        self.conv3 = nn.Conv1d(256,512,1,dilation=1)
        self.bn3 = nn.BatchNorm1d(512)
        self.swish = nn.Hardswish()
    def forward(self,input):
        x = self.swish(self.bn1(self.conv1(input)))
        x = self.swish(self.bn2(self.conv2(x)))
        x = self.swish(self.bn3(self.conv3(x)))
        x = x.permute((2,0,1))
        return x
```

Seq2Seq-model-GRU

LSTM 대신 Weight 계산이 가벼운 GRU 채택
Attention 기법: output 값과 함께 이전의 encoder layer 에서 받은 마지막 input 값도 decoder 모델에 넣어 훈련
-> 직전 시점의 데이터를 참조하는 원리

```
class Decoder(nn.Module):
    def __init__(self,input_size,hidden_size):
        super(Decoder,self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size

        self.gru = nn.GRU(1,512,1,batch_first=True)
        self.fc = nn.Linear(512,1)

    def forward(self,input,encoder_hidden):
        lstm_output, self.hidden = self.gru(input.unsqueeze(-1),encoder
_hidden)
        output = self.fc(lstm_output)
        return output, self.hidden
```

Seq2Seq-model-AutoEncoder

- 앞의 두 모델의 병합
- 출력 개수 지정 (target length): for 구문을 통해 t 시점마다 output 값 입력
- Teacher forcing ratio: for 구문 통해 target_len 씩 선택 시, decoder 의 output 값을 t-1 데이터로 정하고, decoder_input 을 output 으로 변환하여 실행
- output값이 틀렸을 경우 t+1시점에 output값이 적절하게 나오지 않을 것을 우려, 확률적으로 Decoder input을 t시점의 target값으로 바꾸어 for문이돌아갈때,
  t-1시점의 Decoder input을 target_t-1값으로 바꾸어서 진행

Seq2Seq-model-AutoEncoder

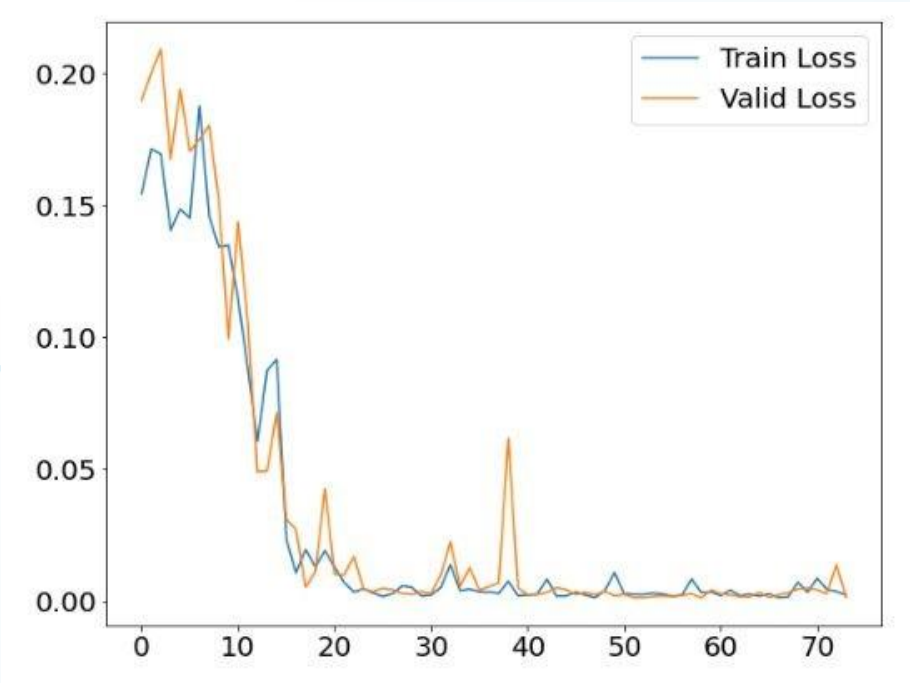
```
class AutoEncoder(nn.Module):
    def __init__(self,input_size=1,hidden_size=64):
        super(AutoEncoder,self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size

        self.encoder = Encoder(input_size,hidden_size)
        self.decoder = Decoder(input_size,hidden_size)
    def forward(self,input,target,target_len,tf_ratio):
        batch_size = input.shape[0]
        input_size = input.shape[2]

        outputs = torch.zeros(batch_size,target_len,input_size)
        hidden = self.encoder(input)
        decoder_input = input[:,-1,:]
        for t in range(target_len):
            output, hidden = self.decoder(decoder_input,hidden)
            output = output.squeeze(1)
            if torch.rand(1) < tf_ratio:
                decoder_input = target[:,t,:]
            else:
                decoder_input = output
            outputs[:,t,:] = output
        return outputs
    def predict(self, inputs, target_len):
        self.eval()
        inputs = inputs.unsqueeze(0)
        batch_size = inputs.shape[0]
        input_size = inputs.shape[2]
        outputs = torch.zeros(batch_size, target_len, input_size)
        hidden = self.encoder(inputs)
        decoder_input = inputs[:,-1, :]
        for t in range(target_len):
            out, hidden = self.decoder(decoder_input, hidden)
            out =  out.squeeze(1)
            decoder_input = out
            outputs[:,t,:] = out
        return outputs.detach().numpy()[0,:,0]
```

Seq2Seq - 훈련 결과

Loss function:
Huber Loss (MAE + MSE)

MAE: outlier 민감도가 낮음 MSE: outlier 민감도가 높음

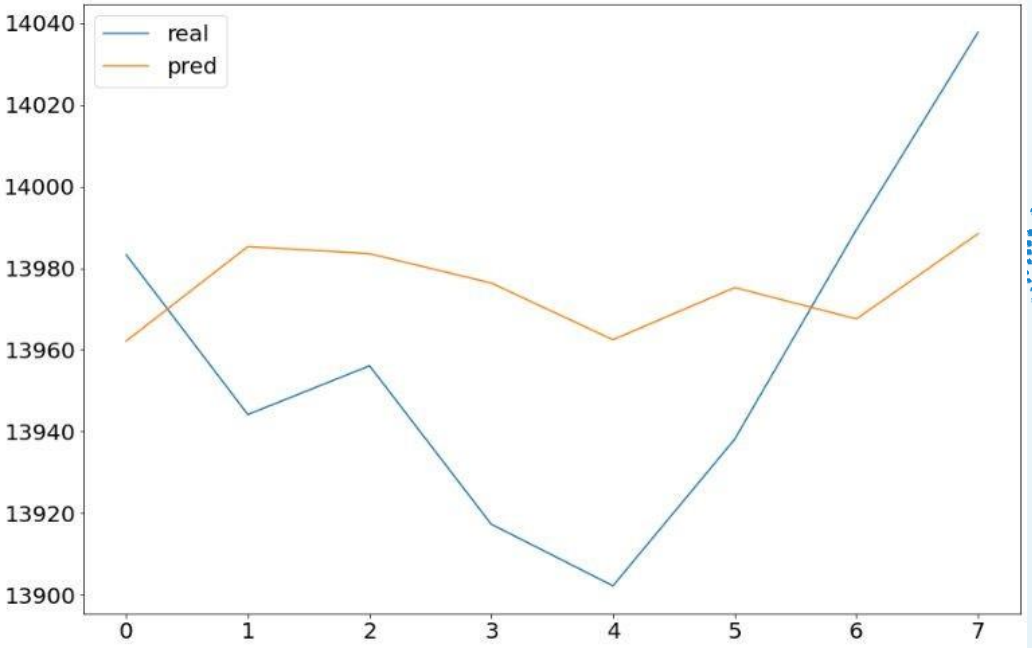
> 두 특성을 모두반영해 최적화를 기대할 수 있음



Seq2Seq - S&P500 훈련 결과

Teacher Forcing 적용

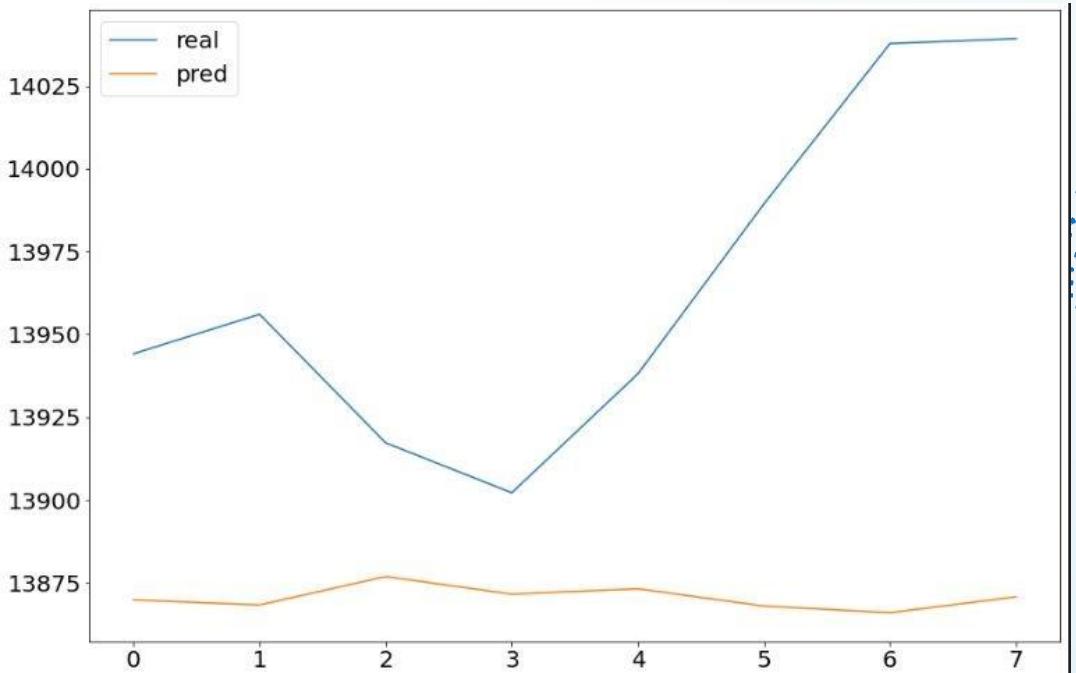
RMSE = 64.5818



Seq2Seq - S&P500 훈련 결과

Teacher Forcing 적용

X RMSE = 107.5751



8. Conclusion

Shifting 현상 해결 후 예측의 정확도 감소

Seq2Seq 모델 연구를 통해 향후 모델 고도화와

Time-Series Data같은 경우 Time Dependency가 강하기 때문에, sliding window dataset 을 적용하여서 Validation이나 test set사용으로 인한 Continuous한 학습으로 인한 성능개선의 여지가 보임.

“

- 시계열 데이터 예측의 난이도 체감
- **3**개 모델 성능 비교를 위해 **1**개 **column** 적용 및 훈련
- 주식, 가상화폐 데이터가 아닌 에너지 소비량처럼 **seoson ality**
  가 명확한 데이터에 적용 시 더 높은 성능 기대
- 뚜렷한 패턴이 없는 주가예측에 **DL** 모델 적용 시도로 성능 테스트에 기여

Reference

- Paper: https://paperswithcode.com/paper/sequence-to-sequence-lea rning-with-neural
- LSTM 설명: https://dgkim5360.tistory.com/entry/understanding-long-short-term-memory-lstm-kr

GitHub Link

- code: https://github.com/Lee-junseok1025/seq2seq_fintech_team1_time-sreies

Thank You