本コンテンツは自己学習のみということでしたので、少し興味があったPytorchを使用した
RNN・LSTMの実装について学びました。
為替の予測などとても興味深い講義やサンプルもご提供いただいておりましたので、そちらも読み込みます。

【自己学習】
※参考資料：3_9_rnn_torch.ipynb
サンプルを写経

```
import numpy as np
import torch
import matplotlib.pyplot as plt
% matplotlib inline
import seaborn as sns
sns.set()

np.random.seed(1)
torch.manual_seed(1)
```

```
# sin曲線+ノイズ
ts = np.linspace(0, 10 * np.pi, 500)
ys = np.sin(ts) + np.random.normal(scale=0.1, size=len(ts))
```

```
# 学習設定
batch_size = 32     # ミニバッチサイズ
n_steps = 50        # 入力系列の長さ
input_size = 1      # 入力の次元
hidden_size = 50    #  中間層の次元
output_size = 1     # 出力層の次元

lr = 0.005          # 学習率(SGD)
n_iter = 500        # イテレーション回数
```

```
# 訓練データとテストデータに分割
train_ratio = 0.8
data = []

for i in range(len(ys) - n_steps - 1):
    data.append(ys[i: i+n_steps+1])
data = np.array(data, dtype=np.float32)
n_train = int(train_ratio * len(data))
x_train, y_train = np.split(data[:n_train], [-1], axis=1)
x_test, y_test = np.split(data[n_train:], [-1], axis=1)
```

```
x_train = np.reshape(x_train, [-1, n_steps, input_size])
x_test = np.reshape(x_test, [-1, n_steps, input_size])
```

```
import torch.nn as nn

class RNN_Net(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN_Net, self).__init__()

        self.rnn = nn.RNN(input_size, hidden_size, num_layers=1,
batch_first=True)
        self.l = nn.Linear(hidden_size, output_size)

    def forward(self, x, h_0=None):
        output, h_n = self.rnn(x, h_0)
        return self.l(output[:, -1])
```

```
model = RNN_Net(input_size, hidden_size, output_size)
# 最適化手法（SGD）
optimizer = torch.optim.Adam(model.parameters())
# 損失関数
loss_fun = nn.MSELoss()
```

```
# 訓練データのインデックスをランダムにする
perm = np.random.permutation(len(x_train))

for i in range(n_iter):
    idx = (i * batch_size) % len(x_train)
    batch_x, batch_y = x_train[perm[idx: idx+batch_size]],
y_train[perm[idx: idx+batch_size]]
    batch_x, batch_y = torch.tensor(batch_x), torch.tensor(batch_y)

    y = model(batch_x)
    loss = loss_fun(y, batch_y)

    # パラメータ更新（勾配のリセット+誤差逆伝播+更新）
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if i % 50 == 0:
        print("step: {}, loss {:.5f}".format(i, loss.data))
```

```
# テストデータで予測
x_test = torch.tensor(x_test)
```

```
prediction = model(x_test)

# 1次元配列に変換
prediction = prediction.detach().numpy().reshape(-1)
true_y = y_test.reshape(-1)
```

```
# モデルの予測を利用し再帰的に予測
curr_x = x_test[0]
predicted = []

# 予測するステップ数
N = 200
for i in range(N):
    # 予測
    pred = model(curr_x[None])
    predicted.append(pred.item())
    # 入力を更新
    curr_x = torch.cat([curr_x[1:], pred], dim=0)
```