Recurrent neural netowrk(rnn)

# example code

|  |
| --- |
| from google.colab import drive  drive.mount('/gdrive')  raw\_data\_path = '/content/drive/MyDrive/news.csv'  destination\_folder = '/content/drive/MyDrive'  import numpy as np  import pandas as pd  import sklearn  from sklearn.preprocessing import MinMaxScaler  import torch  import torch.nn as nn  import torch.optim as optim  import matplotlib  import matplotlib.pyplot as plt  df = pd.read\_csv(raw\_data\_path)  df.head()  scaler = MinMaxScaler()  df[['Open','High','Low','Close','Volume']] = scaler.fit\_transform(df[['Open','High','Low','Close','Volume']])  df.head()  device = torch.device('cuda:0' if torch.cuda.is\_available() else 'cpu')  print(f'{device} is available')  X = df[['Open','High','Low','Volume']].values  y = df['Close'].values  def seq\_data(x, y, sequence\_length):      x\_seq = []    y\_seq = []    for i in range(len(x) - sequence\_length):      x\_seq.append(x[i: i+sequence\_length])      y\_seq.append(y[i+sequence\_length])    return torch.FloatTensor(x\_seq).to(device), torch.FloatTensor(y\_seq).to(device).view([-1, 1]) # float형 tensor로 변형, gpu사용가능하게 .to(device)를 사용.    split = 200  sequence\_length = 5  x\_seq, y\_seq = seq\_data(X, y, sequence\_length)  x\_train\_seq = x\_seq[:split]  y\_train\_seq = y\_seq[:split]  x\_test\_seq = x\_seq[split:]  y\_test\_seq = y\_seq[split:]  print(x\_train\_seq.size(), y\_train\_seq.size())  print(x\_test\_seq.size(), y\_test\_seq.size())  train = torch.utils.data.TensorDataset(x\_train\_seq, y\_train\_seq)  test = torch.utils.data.TensorDataset(x\_test\_seq, y\_test\_seq)  batch\_size = 20  train\_loader = torch.utils.data.DataLoader(dataset=train, batch\_size=batch\_size, shuffle=False)  test\_loader = torch.utils.data.DataLoader(dataset=test, batch\_size=batch\_size, shuffle=False)  input\_size = x\_seq.size(2)  num\_layers = 2  hidden\_size = 8  class VanillaRNN(nn.Module):    def \_\_init\_\_(self, input\_size, hidden\_size, sequence\_length, num\_layers, device):      super(VanillaRNN, self).\_\_init\_\_()      self.device = device      self.hidden\_size = hidden\_size      self.num\_layers = num\_layers      self.rnn = nn.RNN(input\_size, hidden\_size, num\_layers, batch\_first=True)      self.fc = nn.Sequential(nn.Linear(hidden\_size \* sequence\_length, 1), nn.Sigmoid())    def forward(self, x):      h0 = torch.zeros(self.num\_layers, x.size()[0], self.hidden\_size).to(self.device) # 초기 hidden state 설정하기.      out, \_ = self.rnn(x, h0) # out: RNN의 마지막 레이어로부터 나온 output feature 를 반환한다. hn: hidden state를 반환한다.      out = out.reshape(out.shape[0], -1) # many to many 전략      out = self.fc(out)      return out  model = VanillaRNN(input\_size=input\_size,                     hidden\_size=hidden\_size,                     sequence\_length=sequence\_length,                     num\_layers=num\_layers,                     device=device).to(device)    criterion = nn.MSELoss()  lr = 1e-3  num\_epochs = 200  optimizer = optim.Adam(model.parameters(), lr=lr)  loss\_graph = [] # 그래프 그릴 목적인 loss.  n = len(train\_loader)  for epoch in range(num\_epochs):    running\_loss = 0.0    for data in train\_loader:      seq, target = data # 배치 데이터.      out = model(seq)   # 모델에 넣고,      loss = criterion(out, target) # output 가지고 loss 구하고,      optimizer.zero\_grad() #      loss.backward() # loss가 최소가 되게하는      optimizer.step() # 가중치 업데이트 해주고,      running\_loss += loss.item() # 한 배치의 loss 더해주고,    loss\_graph.append(running\_loss / n) # 한 epoch에 모든 배치들에 대한 평균 loss 리스트에 담고,    if epoch % 10 == 0:      print('[epoch: %d] loss: %.4f'%(epoch, running\_loss/n))  plt.figure(figsize=(20,10))  plt.plot(loss\_graph)  plt.show()  def plotting(train\_loader, test\_loader, actual):    with torch.no\_grad():      train\_pred = []      test\_pred = []      for data in train\_loader:        seq, target = data        out = model(seq)        train\_pred += out.cpu().numpy().tolist()      for data in test\_loader:        seq, target = data        out = model(seq)        test\_pred += out.cpu().numpy().tolist()      total = train\_pred + test\_pred    plt.figure(figsize=(20,10))    plt.plot(np.ones(100)\*len(train\_pred), np.linspace(0,1,100), '--', linewidth=0.6)    plt.plot(actual, '--')    plt.plot(total, 'b', linewidth=0.6)    plt.legend(['train boundary', 'actual', 'prediction'])    plt.show()  plotting(train\_loader, test\_loader, df['Close'][sequence\_length:]) |

# testing result

|  |
| --- |
| [epoch: 0] loss: 0.0370  [epoch: 10] loss: 0.0061  [epoch: 20] loss: 0.0056  [epoch: 30] loss: 0.0046  [epoch: 40] loss: 0.0029  [epoch: 50] loss: 0.0012  [epoch: 60] loss: 0.0011  [epoch: 70] loss: 0.0010  [epoch: 80] loss: 0.0010  [epoch: 90] loss: 0.0010  [epoch: 100] loss: 0.0010  [epoch: 110] loss: 0.0010  [epoch: 120] loss: 0.0010  [epoch: 130] loss: 0.0010  [epoch: 140] loss: 0.0009  [epoch: 150] loss: 0.0009  [epoch: 160] loss: 0.0009  [epoch: 170] loss: 0.0009  [epoch: 180] loss: 0.0009  [epoch: 190] loss: 0.0009 |