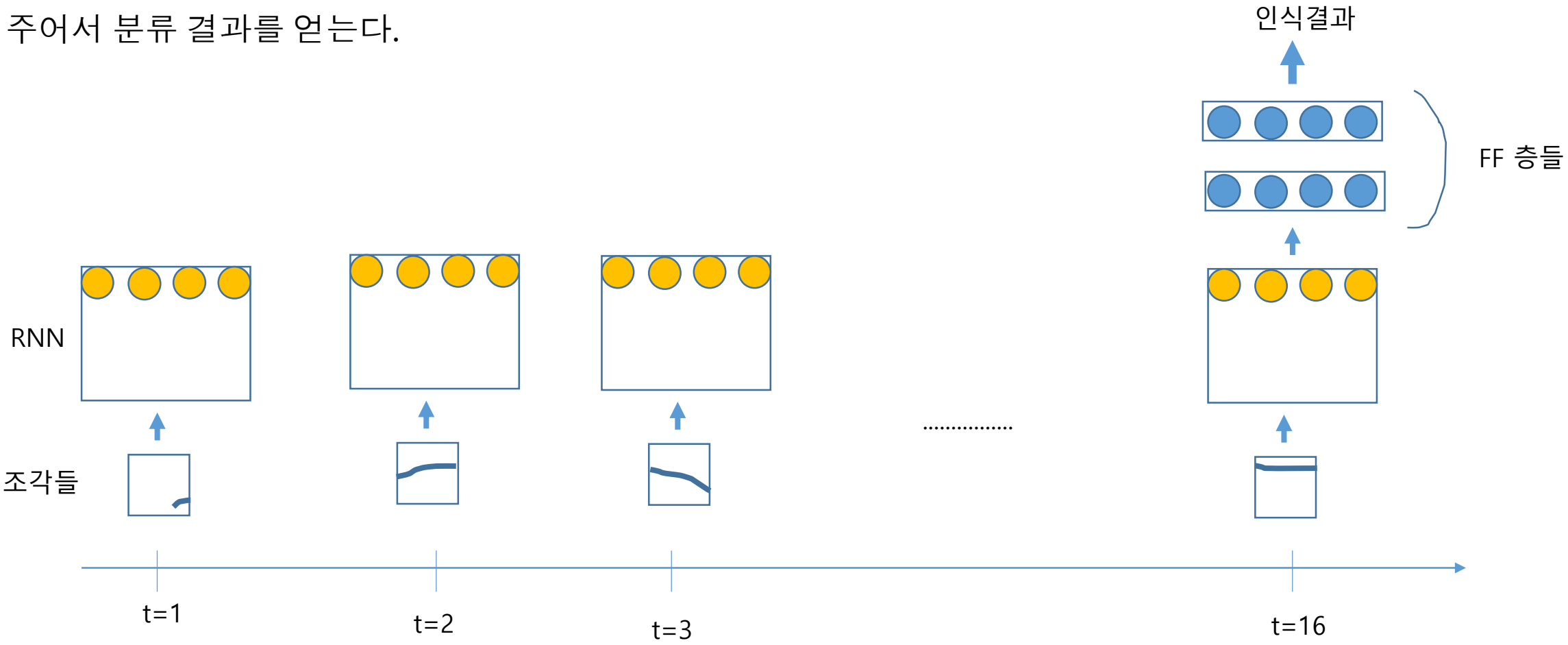
LSTM 을 이용한 MNIST 분류기(classifier) 개발

- 숫자 이미지 하나가 주어졌다 하자.

- 이의 분류에 대한 문제이다.
- 지금까지는 한 숫자 이미지를 FFNN, CNN 에게 주어서 한번에 인식한다.
- 이것을 RNN 이나 LSTM 을 이용해 보자.
  - 이 이미지 하나를 여러 시간대에 걸쳐 조각으로 나누어 입력해 준다.
  - 우측 그림과 같이 16개의 조각으로 나누어서 t=1....16 시간 대에 걸쳐
  - 각 조각을 LSTM 에게 입력해 준다.

분류 결과 구하기

- 마지막 시간의 LSTM 의 최상층 hidden state 를 FFNN 층들에게 주어서 분류 결과를 얻는다.



Importing and contant 선언:

```
# MNIST classifer implemented by LSTM

import os
import time
import numpy as np
import torch
from torch import nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
import torch.optim as optim
from torch.utils.data import Dataset
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler

# GPU 를 사용하기 위한 선언.
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print("device=", device)

import torchvision.datasets as dsets
from torch.utils.data import DataLoader

# 데이터 다운로드. (주어진 디렉토리에 이미 다운로드 데이터가 없다면 다운로드 수행함. 디렉토리는 미리 만들어 놓을 것.)
mnist_train = dsets.MNIST(root='./MNIST_data/', train=True, transform=transforms.ToTensor(), download=True)
mnist_test = dsets.MNIST(root='./MNIST_data/',train=False, transform=transforms.ToTensor(), download=True)

BATCH_SIZE = 32
NUM_CLASS = 10
NUM_HIDDEN_LAYERS = 2
HIDDEN_STATE_SIZE = 256
num_epochs = 8
```

모델설계:

```
32 # 모델 설계
33 class MNIST_LSTM_model(nn.Module):
34     def __init__(self, num_hidden_layers, hidden_state_size):
35         super(MNIST_LSTM_model, self).__init__()
36 
37         self.lstm = nn.LSTM(input_size=49, hidden_size=hidden_state_size,  #
38                             num_layers=num_hidden_layers, batch_first=True, bidirectional=True)
39 
40         self.linear1 = nn.Linear(2*hidden_state_size, 128, bias=True)
41         self.relu = nn.ReLU()
42         self.linear2 = nn.Linear(128, NUM_CLASS)    # NUM_CLASS : 분류할 클래수 개수로 여기서는 10.
43 
44     def forward(self, X):
45         # X : shape (batch_size, 16, 49)
46 
47         x1 = self.lstm(X)       # 출력은 list 이다.
48 
49         x1 = x1[0]              # 위 출력의 첫 원소가 우리가 원하는 출력. shape:  (batch_size, seq_length, 2*hidden_size).
50         x2 = x1[:, 15]          # 마지막 시간(t=15)의 hidden_state 을 가져옴. x2의 shape: (batch_size,2*hidden_size)
51         x3 = self.linear1(x2)   # output shape is (batch, 128).
52         x4 = self.relu(x3)
53         x5 = self.linear2(x4)   # output shape is (batch, num_class).
54         return x5
55 
56 model = MNIST_LSTM_model(NUM_HIDDEN_LAYERS, HIDDEN_STATE_SIZE )
57 model = model.to(device)
58 
59 # loss 를 계산할 객체 준비 (cross entropy 이용). CrossEntropyLoss 가 softmax 수행함.
60 criterion = nn.CrossEntropyLoss().to(device) # 가능하다면 gpu 로 보냄.
61 
62 # 훈련을 수행한 optimizer 선정: 파라미테 셋과 learning rate 설정.
63 optimizer = torch.optim.AdamW(model.parameters(), lr=0.001)
64 
65 # 훈련과 테스트에 사용할 배치 리스트 준비.
66 data_loader = DataLoader(dataset=mnist_train, batch_size=BATCH_SIZE, shuffle=True, drop_last=True)
67 data_loader_test = DataLoader(dataset=mnist_test, batch_size=BATCH_SIZE, shuffle=True, drop_last=True)
68 
69 total_batch = len(data_loader)
70 total_batch_test = len(data_loader_test)
71 print("Total number of batches for train and test = ", total_batch, " ,  ", total_batch_test)
72 
73 a_batch = torch.zeros((BATCH_SIZE, 4, 4, 7, 7))
74 b_batch = torch.zeros((BATCH_SIZE, 16, 49))
75 
76 model.train()
```

훈련단계:


```
77  for epoch in range(num_epochs):
78      total_loss = 0.0
79      cnt_batch = 0
80
81      for X, Y in data_loader:
82          #  배치에서 가져온 X는 shape = (BATCH_SIZE, 1, 28,28)의 텐서이다.
83          #       "   "   Y는 shape = (BATCH_SIZE, 1) 로서 각 원소는 레이블로서 0 ~ 9의 정수.
84
85          Y = Y.to(device)
86
87          # 배치 내의 예제마다 다음 작업을 수행한다:
88          #    (1, 28, 28) 형태의 예제를 16 개의 (7, 7)형태의 조각으로 나누고 조각을 (49) 형태로 펴서
89          # 결국 (16, 49) 형태로 만든다. 이를 16 시간 대에 걸쳐서 길이 49인 벡터를 입력으로 넣는다.
90
91          for r in range(4):
92              for c in range(4):
93                  a_batch[:, r, c, :, :] = X[:, 0, r*7:(r+1)*7, c*7:(c+1)*7]
94
95          t_batch = torch.reshape(a_batch, (BATCH_SIZE, 4, 4, 49))   # 마지막 2 개 차원을 flatten하여 얻음.
96
97          for r in range(4):
98              for c in range(4):
99                  b_batch[:, 4*r+c, :] = t_batch[:, r, c, :]
100
101         b_batch = b_batch.float().to(device)
102
103         optimizer.zero_grad()
104
105         hypothesis = model(b_batch)   # 모델의 출력
106
107         loss = criterion(hypothesis, Y)     # loss 결정
108         total_loss += loss.item()   # loss 값 축적
109
110         # backpropagation 에 의한 gradient 계산
111         loss.backward( )
112
113         # gradient 로 parameter 갱신신
114         optimizer.step( )
115
116         cnt_batch += 1
117         #if cnt_batch % 600 == 0:
118             #print("batch count done = ", cnt_batch)
119
120     avg_loss = total_loss / total_batch
121     print('Epoch:', '%04d' % (epoch + 1), '.  loss per example =', '{:.9f}'.format(avg_loss))
122
123 print('Learning has finished')
```

```
126 print("\nTesting begins.\n")
127 model.eval()
128 with torch.no_grad( ):          ## torch.no_grad()를 하면 gradient 계산을 수행하지 않는다.
129     total_hit = 0
130     total_num_examples = 0
131
132     for X, Y in data_loader_test:
133         Y = Y.to(device)
134
135         for r in range(4):
136             for c in range(4):
137                 a_batch[:, r, c, :, :] = X[:, 0, r * 7:(r + 1) * 7, c * 7:(c + 1) * 7]
138
139         t_batch = torch.reshape(a_batch, (BATCH_SIZE, 4, 4, 49))  # 마지막 2 개 차원을 flatten하여 얻음.
140
141         for r in range(4):
142             for c in range(4):
143                 b_batch[:, 4 * r + c, :] = t_batch[:, r, c, :]
144
145         b_batch = b_batch.float().to(device)
146
147         optimizer.zero_grad()
148
149         prediction = model(b_batch)  # 모델의 출력
150
151         pr_label = torch.argmax(prediction, 1)
152         correct = pr_label == Y
153         hit_cnt = correct.sum()
154         total_hit += hit_cnt
155         total_num_examples += BATCH_SIZE
156
157 accuracy = float(total_hit) / total_num_examples
158 print("test accuracy = ", accuracy)
159 print("\nprogram terminates.")
```

실행 결과

```
device= cuda
Total number of batches for train and test =  1875  ,  312
Epoch: 0001 .  loss per example = 0.439984521
Epoch: 0002 .  loss per example = 0.147714125
Epoch: 0003 .  loss per example = 0.104264925
Epoch: 0004 .  loss per example = 0.080226218
Epoch: 0005 .  loss per example = 0.062347282
Learning has finished

Testing begins.

test accuracy =  0.977363782051282

program terminates.
```