KU KONKUK UNIVERSITY

Recurrent Neural Network

자연어처리연구실

PART-I

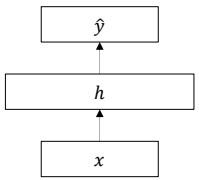
Recurrent Neural Network (RNN)

- 순환 신경망
  - 은닉 노드(node)가 방향을 가진 엣지(edge)로 연결되어 순환 구조를 이루는 인공신경망의 한 종류
  - 음성이나 문자와 같이 시간 축을 따라서 순차적으로 등장하는 데이터 처리에 적합함
  - 장점
    - 입출력 길이가 자유로움
    - 필요에 따라 다양하고 유연하게 구조를 만들 수 있음

FNN vs. RNN

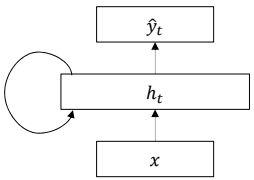
Feed-forward NN

- $h = g(Vx + c)$
- $\hat{y} = Wh + b$

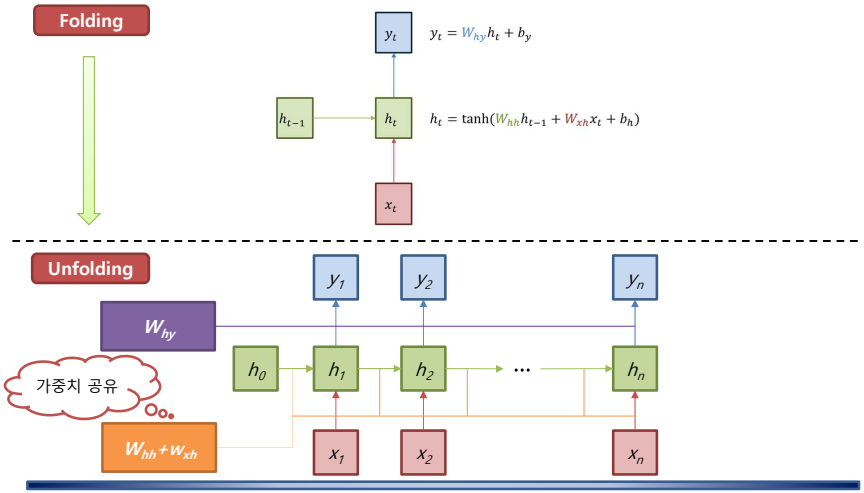


Recurrent NN

- $h_t = g(Vx_t + Uh_{t-1} + c)$
- $\hat{y}_t = Wh_t + b$



RNN 기본 구조



RNN 응용 구조

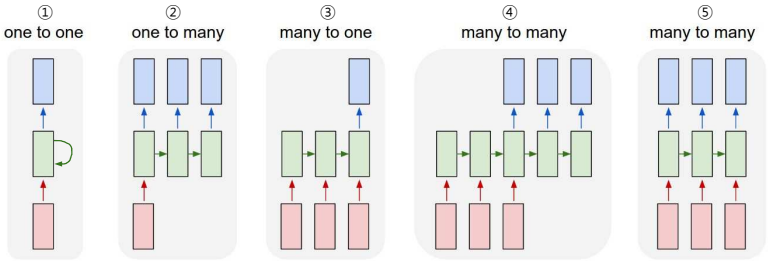
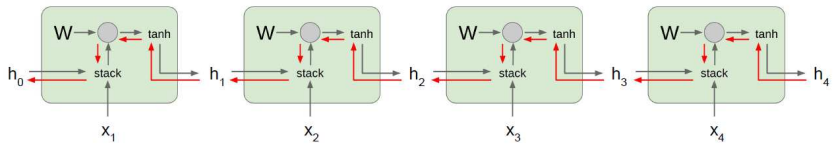


그림 참조: Stanford Lecture CS2031N, 2017 (Fei-Fei Li & Justin Johnson & Serena Yeung)

Blue : output vector
Green : RNN's state
Red : input vector

[활용 예]
① 일반적인 NN (FNN과 유사함)
② 자막 생성 (image captioning)
③ 분류 및 예측 (prediction)
④ 기계 번역
⑤ 순차 표지 부착

From Vanilla RNN To LSTM



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix}h_{t-1}\\x_t\end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix}h_{t-1}\\x_t\end{pmatrix}\right)$$

Error Backpropagation:
"*tanh(tanh(tanh(tanh(W)*W)*W)*W)*" 미분

Largest singular value < 1:
Vanishing gradients → Change RNN architecture

그림 출처: Stanford Lecture CS2031N, 2017 (Fei-Fei Li & Justin Johnson & Serena Yeung)

Long Short-Term Memory (LSTM)

Vanilla RNN

$$h_t = \tanh\left(W\begin{pmatrix}h_{t-1}\\x_t\end{pmatrix}\right)$$

vs.

LSTM

$$\begin{pmatrix}i\\f\\o\\g\end{pmatrix} = \begin{pmatrix}\sigma\\\sigma\\\sigma\\\tanh\end{pmatrix} W\begin{pmatrix}h_{t-1}\\x_t\end{pmatrix}$$
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

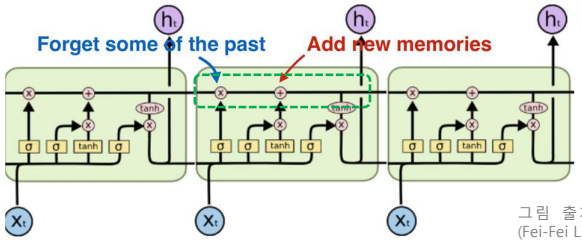


그림 출처: Stanford Lecture CS2031N, 2017 (Fei-Fei Li & Justin Johnson & Serena Yeung)

Many-to-One Model (Classification and Regression)

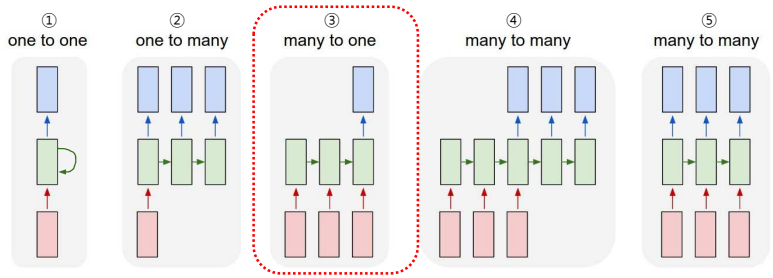


그림 참조: Stanford Lecture CS2031N, 2017
(Fei-Fei Li & Justin Johnson & Serena Yeung)

Blue : output vector
Green : RNN's state
Red : input vector

[활용 예]
① 일반적인 NN (FNN)
② 자막 생성 (image captioning)
③ 분류 및 예측 (prediction)
④ 기계 번역
⑤ 순차 표지 부착

Sequence → Classes (Values)
(영화평 분류, 주가 예측 등)

실습

- 문제
  - 일일 주가 데이터를 학습하여 특정일 종가를 예측
- 입력 데이터
  - CSV(comma-separated values) 형식의 일일 주가 데이터

	A	B	C	D	E	F
1	날짜	종가	오픈	고가	저가	거래량
2	2020년 01월 02일	55.2	55.5	56	55	12760
3	2020년 01월 03일	55.5	56	56.6	54.9	15310
4	2020년 01월 05일	55.5	55.5	55		
5	2020년 01월 06일	55.5	54.9	55		
6	2020년 01월 07일	55.8	55.7	56		
7	2020년 01월 08일	56.8	56.2	57		

samsung-2020.csv - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
2020년 02월 28일,54.2,55,55.5,54.2,29910
2020년 03월 01일,54.2,54.2,54.2,54.2,0
2020년 03월 02일,55,54.3,55.5,53.6,29210
2020년 03월 03일,55.4,56.7,56.9,55.1,29810
2020년 03월 04일,57.4,54.8,57.6,54.6,24320
```


실습

실습 코드 다운로드:
https://github.com/KUNLP/Lecture

```python
import os
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import (DataLoader, RandomSampler, TensorDataset)
import csv
from sklearn.preprocessing import MinMaxScaler

class STOCK_RNN(nn.Module):

  def __init__(self, config):

    super(STOCK_RNN, self).__init__()

    self.input_size = config["input_size"]
    self.hidden_size = config["hidden_size"]
    self.output_size = config["output_size"]
    self.num_layers = config["num_layers"]
    self.batch_size = config["batch_size"]

    # LSTM 설계
    self.lstm = nn.LSTM(self.input_size, self.hidden_size, self.num_layers, bidirectional=False, batch_first=True)
    # 출력층 설계
    self.linear = nn.Linear(self.hidden_size,self.output_size)

  def forward(self, input_features):

    # LSTM 리턴 = output (배치, 시퀀스, 은닉 상태), (hidden_state, cell_state)
    x, (h_n, c_n) = self.lstm(input_features)

    # output에서 마지막 시퀀스의 (배치, 은닉 상태) 정보를 가져옴
    h_t = x[:,-1,:]

    # 출력층: (배치, 출력)
    hypothesis = self.linear(h_t)

    return hypothesis
```

LSTM의 출력 형태

Many-to-one 모델

실습

```python
# 데이터 읽기 함수
def load_dataset(fname):

  f = open(fname, 'r', encoding='cp949')

  # CSV 파일 읽기
  data = csv.reader(f,delimiter=',')

  # 헤더 건너뛰기
  next(data)

  data_X = []
  data_Y = []

  for row in data:
    # 오픈, 고가, 저가, 거래량 -> 숫자 변환
    data_X.append([float(i) for i in row[2:]])
    # 종가 -> 숫자 변환
    data_Y.append(float(row[1]))

  # MinMax 정규화 (예측하려는 종가 제외)
  scaler = MinMaxScaler()
  scaler.fit(data_X)
  data_X = scaler.transform(data_X)

  data_num = len(data_X)
  sequence_len = config["sequence_len"]
  seq_data_X, seq_data_Y = [], []

  # 윈도우 크기만큼 슬라이딩 하면서 데이터 생성
  for i in range(data_num-sequence_len):
    window_size = i+sequence_len
    seq_data_X.append(data_X[i:window_size])
    seq_data_Y.append([data_Y[window_size-1]])

  (train_X, train_Y) = (np.array(seq_data_X[:]),np.array(seq_data_Y[:]))
  train_X = torch.tensor(train_X, dtype=torch.float)
  train_Y = torch.tensor(train_Y, dtype=torch.float)

  print(train_X.shape) # (73,3,4)
  print(train_Y.shape) # (73,1)

  return (train_X, train_Y)
```

```
torch.Size([72, 3, 4])
torch.Size([72, 1])
```

CSV 파일 읽기

데이터 범위가 매우 상이한 경우 → 정규화 필수

입력 데이터: 시퀀스 길이 만큼씩 구성
Sequence length = 3
[[1,2],[3,4],[5,6],[7,8]]
→ [[[1,2],[3,4],[5,6]], [[3,4],[5,6],[7,8]]]

실습

```python
# 평가 수행 함수
def do_test(model, test_dataloader):

  # 평가 모드 셋팅
  model.eval()

  ...

      input_features, labels = batch
      hypothesis = model(input_features)

      x = tensor2list(hypothesis[:,0])
      y = tensor2list(labels)

      # 예측값과 정답을 리스트에 추가
      predicts.extend(x)
      golds.extend(y)

    # 소숫점 이하 1자리로 변환
    predicts = [round(i,1) for i in predicts]
    golds = [round(i[0],1) for i in golds]

    print("PRED=",predicts)
    print("GOLD=",golds)
```

```python
# 모델 평가 함수
def test(config):

  model = STOCK_RNN(config).cuda()
```

회귀 모델
(regression model)
→ argmax 필요 없음

실습

```python
# 모델 학습 함수
def train(config):

  # 모델 생성
  model = STOCK_RNN(config).cuda()

  # 데이터 읽기
  (input_features, labels) = load_dataset(config["file_name"])

  # TensorDataset/DataLoader를 통해 배치(batch) 단위로 데이터를 나누고 셔플(shuffle)
  train_features = TensorDataset(input_features, labels)
  train_dataloader = DataLoader(train_features, shuffle=True, batch_size=config["batch_size"])

  # MSE (Mean Square Error) 비용 함수
  ?
  # 옵티마이저 함수 (역전파 알고리즘을 수행할 함수)
  optimizer = torch.optim.Adam(model.parameters(), lr=config["learn_rate"])

  for epoch in range(config["epoch"]+1):

    # 학습 모드 셋팅
    model.train()

    # epoch 마다 평균 비용을 저장하기 위한 리스트
    costs = []
```

회귀 모델 → MSE

실습

```python
if(__name__=="__main__"):

    root_dir = "/gdrive/My Drive/colab/rnn/stock"
    output_dir = os.path.join(root_dir, "output")
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    config = {"mode": "train",
              "model_name":"epoch_{0:d}.pt".format(10),
              "output_dir":output_dir,
              "file_name": "{0:s}/samsung-2020.csv".format(root_dir),
              "sequence_len": 3,
              "input_size": 4,
              "hidden_size": 10,
              "output_size": 1,
              "num_layers": 1,
              "batch_size": 1,
              "learn_rate": 0.1,
              "epoch": 10,
              }

    if(config["mode"] == "train"):
        train(config)
    else:
        test(config)
```

```
Average Loss= 907.702039
PRED= [53.1, 53.1, 53.1, 53.1, 53.1, 53.1, 53.1, 53.1, 53.1, 53.1,
GOLD= [45.4, 47.3, 55.9, 60.7, 43.0, 56.5, 59.9, 61.8, 57.9,
Average Loss= 30.854506
PRED= [54.5, 54.5, 54.5, 54.5, 54.5, 54.5, 54.5, 54.5, 54.5, 54.5,
GOLD= [61.3, 54.6, 56.5, 54.2, 50.8, 60.7, 58.9, 54.2, 61.8,
Average Loss= 29.129469
PRED= [55.7, 55.8, 55.3, 55.9, 56.0, 56.2, 55.7, 56.1, 56.0,
GOLD= [55.5, 59.5, 42.5, 61.1, 56.4, 60.8, 55.0, 59.1, 59.5,
Average Loss= 12.220411
PRED= [58.1, 58.1, 58.1, 58.0, 58.1, 58.1, 58.1, 58.1, 58.1,
GOLD= [57.4, 55.8, 60.2, 50.8, 58.9, 60.7, 57.8, 54.6, 59.5,
Average Loss= 7.938679
PRED= [58.7, 58.7, 58.7, 58.7, 58.7, 58.7, 58.7, 58.7, 53.4,
GOLD= [55.4, 60.0, 61.3, 59.2, 56.8, 59.5, 57.2, 60.0, 48.9,
```


PART-II에 계속됩니다!