



# RNN hihello and charseq

▼ 상태	RNN
👤 담당자	

[풀고자 하는 문제](#)

[예상되는 문제](#)

[데이터 세팅](#)

[Cross Entropy Loss](#)

[Declare RNN](#)

[학습](#)

[학습 결과](#)

## 풀고자 하는 문제



'h' → 'i' → 'h' → 'e' → 'l' → 'l' → 'o' 순으로 다음 문자의 출현을 예측해 보자.

## 예상되는 문제

'h' 다음에는 'i'가 오는 경우도, 'e'가 오는 경우도 존재한다.

## 데이터 세팅

We can represent them by index

- 'h' -> 0
- 'i' -> 1
- 'e' -> 2
- 'l' -> 3
- 'o' -> 4

알파벳을 0-4의 인덱스로 표현할 수 있다.

```
# declare dictionary
char_set = ['h', 'i', 'e', 'l', 'o']

# hyper parameters
input_size = len(char_set)
hidden_size = len(char_set)
learning_rate = 0.1

# data setting
```

```

x_data = [[0, 1, 0, 2, 3, 3]] # hihell
x_one_hot = [[1, 0, 0, 0, 0], # h
              [0, 1, 0, 0, 0], # i
              [1, 0, 0, 0, 0], # h
              [0, 0, 1, 0, 0], # e
              [0, 0, 0, 1, 0], # l
              [0, 0, 0, 1, 0]] # l
y_data = [[1, 0, 2, 3, 3, 4]] # ihello

# transform as torch tensor variable
X = torch.FloatTensor(x_one_hot)
Y = torch.LongTensor(y_data)

```

`x_data` 는 문자열에서 맨 마지막 문자만 뺀 데이터이다(hihell). 이를 원핫 인코딩한 `x_one_hot` 은 각각 h, i, h, e, l, l을 원핫 인코딩한 것이다.

`y_data` 는 문자열에서 맨 첫 번째 문자만 뺀 데이터이다(ihello).

## Cross Entropy Loss

Categorical output에 대한 loss를 계산하기 위해 cross entropy loss를 사용한다.

```

# loss & optimizer setting
criterion = torch.nn.CrossEntropyLoss()
loss = criterion(outputs.view(-1, input_size), Y.view(-1))

```

## Declare RNN

```

# declare RNN
rnn = torch.nn.RNN(input_size, hidden_size, batch_first=True) # batch_first guarantees the order of output = (B, S, F)

```

```

optimizer = optim.Adam(rnn.parameters(), learning_rate)

```

## 학습

```

# start training
for i in range(100):
    optimizer.zero_grad() # gradient를 축적하지 않고 매번 새로 구하기
    outputs, _status = rnn(X)
    # outputs : 모델에 독립변수를 넣어 구한 예측값
    # _status : hidden state
    loss = criterion(outputs.view(-1, input_size), Y.view(-1))
    # batch dimension을 제거해 꼭 펼쳐주기 위함
    loss.backward()
    optimizer.step()

    result = outputs.data.numpy().argmax(axis=2)
    result_str = ''.join([char_set[c] for c in np.squeeze(result)])
    print(i, "loss: ", loss.item(), "prediction: ", result, "true Y: ", y_data, "prediction str: ", result_str)

```

## 학습 결과

```
0 loss: 1.7802648544311523 prediction: [[1 1 1 1 1]] true Y: [[1, 0, 2, 3, 3, 4]] prediction str: iiii
1 loss: 1.4931954145431519 prediction: [[1 4 1 1 4 4]] true Y: [[1, 0, 2, 3, 3, 4]] prediction str: ioioo
2 loss: 1.3337129354476929 prediction: [[1 3 2 3 1 4]] true Y: [[1, 0, 2, 3, 3, 4]] prediction str: ilelio
3 loss: 1.215295433998108 prediction: [[2 3 2 3 3 3]] true Y: [[1, 0, 2, 3, 3, 4]] prediction str: elelll
4 loss: 1.1131411790847778 prediction: [[2 3 2 3 3 3]] true Y: [[1, 0, 2, 3, 3, 4]] prediction str: elelll
5 loss: 1.0241888761520386 prediction: [[2 3 2 3 3 4]] true Y: [[1, 0, 2, 3, 3, 4]] prediction str: elello
6 loss: 0.9573155045509338 prediction: [[2 3 2 3 3 4]] true Y: [[1, 0, 2, 3, 3, 4]] prediction str: elello
..
96 loss: 0.5322802066802979 prediction: [[1 3 2 3 3 4]] true Y: [[1, 0, 2, 3, 3, 4]] prediction str: ilello
97 loss: 0.5321123003959656 prediction: [[1 3 2 3 3 4]] true Y: [[1, 0, 2, 3, 3, 4]] prediction str: ilello
98 loss: 0.5319531559944153 prediction: [[1 3 2 3 3 4]] true Y: [[1, 0, 2, 3, 3, 4]] prediction str: ilello
99 loss: 0.5317898392677307 prediction: [[1 3 2 3 3 4]] true Y: [[1, 0, 2, 3, 3, 4]] prediction str: ilello
```

초반에는 학습을 잘 못하는 것처럼 보이지만(iiiii, ioioo, ilelio 등) 학습이 거듭되면서 정답인 ihello에 근접해지는 것을 확인할 수 있다.