



Type something...

01) RNN hihello and charseq

01_RNN hihello

02_RNN charseq

01) RNN hihello and charseq

01_RNN hihello

One-hot encoding

- We need to encode using one-hot encoding!

```
# list of available characters
char_set = ['h', 'i', 'e', 'l', 'o']
x_data = [[0, 1, 0, 2, 3, 3]]
x_one_hot = [[[1, 0, 0, 0, 0],
              [0, 1, 0, 0, 0],
              [1, 0, 0, 0, 0],
              [0, 0, 1, 0, 0],
              [0, 0, 0, 1, 0],
              [0, 0, 0, 1, 0]]]
y_data = [[1, 0, 2, 3, 3, 4]]
```

- 문자열을 `one-hot encoding` 을 해준다.
- 다른 문자가 5개이므로 각각 size는 5이다.

하지만 RNN이나 LSTM 같은 경우는 batch_size가 아닌 sequence가 가장 먼저 나옵니다.

input : [sequence, batch_size, input_size]

model에서 나오는 output 또한 sequence가 가장 먼저 나옵니다.

output : [sequence, batch_size, hidden_size]

이렇게 sequence가 가장 앞에서 사용되면 가끔 데이터 차원이 헷갈립니다.

그럴 때 사용하는 것이 `batch_first=True` 입니다. default는 False입니다.



`batch_first`를 True로 설정하면 `batch_size`가 제일 먼저 앞으로 이동합니다.

`input : [batch_size, sequence, input_size]`

`output : [batch_size, sequence, hidden_size]`

```
rnn = torch.nn.RNN(input_size, hidden_size, batch_first = True)

criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.Adam(rnn.parameters(), learning_rate)

for i in range(100):
    optimizer.zero_grad()
    outputs, _status = rnn(X)
    # outputs = (1, 6, 5) -> (6, 5)
    # Y = (1, 6) -> (6, )
    loss = criterion(outputs.view(-1, input_size), Y.view(-1))
    loss.backward()
    optimizer.step()
    result = outputs.data.numpy().argmax(axis = 2) # (1, 6)
    #print(np.squeeze(result).shape) #-> (6, )

    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)
```

- `batch_first = True` 를 통해 `batch_size` 를 제일 먼저 앞으로 옮긴다.

- 마지막 축을 접어야하니 `axis = 2` 를 사용하였다. (3차원)
- `np.squeeze()` 를 통해서 문자열을 저장해주었다.

02_RNN charseq

- hihello를 일반화한 코드를 살펴보자.

```
[4] sample = 'if you want you'

[10] char_set = list(set(sample)) # 집합으로 받아서 1개씩만 빨기
     char_dic = {c: i for i, c in enumerate(char_set)}
     print(char_dic)

→ {'u': 0, 'w': 1, 't': 2, 'f': 3, 'a': 4, 'n': 5, ' ': 6, 'i': 7, 'o': 8, 'y': 9}
```

- `set` 을 통해 문자들 고유하게 받는다.
- `enumerate` 를 통해서 index와 값에 동시에 접근한 후 dic로 만든다.

2. np.eye 함수



```
In [2]: np.eye(N=3, M=5, # 총 행렬의 크기는 3x5로 만들어 달라
           k=1, # identity matrix의 시작은 1열부터 시작하라
           dtype=np.int8) # dtype은 int8로 설정

Out[2]: array([[0, 1, 0, 0, 0],
               [0, 0, 1, 0, 0],
               [0, 0, 0, 1, 0]],
              dtype=int8)
```

- eye 함수로 생성되는 배열은 identity와 유사하다.
- 차이는 Identity 행렬은 항상 $n \times n$ 의 행렬이라면, eye는 원하는 $N \times M$ 으로 행렬을 만들 수 있다.
- k 변수를 통해서 identity matrix가 시작할 열을 지정해줄 수 있다.
- 위의 예제에서 보면, N=3/M=5/k=1이기 때문에
3x5 행렬을 만들고, 1번 열부터 시작하여 1을 대각선으로 입력하고, 나머지는 0으로 채워
진 행렬을 array 형태로 반환한다.

- np.eye를 통해서 one-hot encoding을 구현하자.

▶ sample_idx = [char_dic[c] for c in sample]
x_data = [sample_idx[:-1]] # 마지막 전까지 가져오기
print(np.eye(dic_size))

➡ [[1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]

- np.eye의 행을 뽑아서 데이터로 만들어보자.

▶ sample_idx = [char_dic[c] for c in sample]
x_data = [sample_idx[:-1]] # 마지막 전까지 가져오기
x_one_hot = [np.eye(dic_size)[x] for x in x_data]
y_data = [sample_idx[1:]] # 두번째부터 끝까지

- x_one_hot = [np.eye(dic_size)[x] for x in x_data]
- 뒷 부분은 hihello와 같으므로 생략하겠다.
 - one-hot encoding 구현부를 주의깊게 살펴보자.

