RNN basics

상태	RNN
담당자	

RNN(Recurrent Neueral Network)이란?

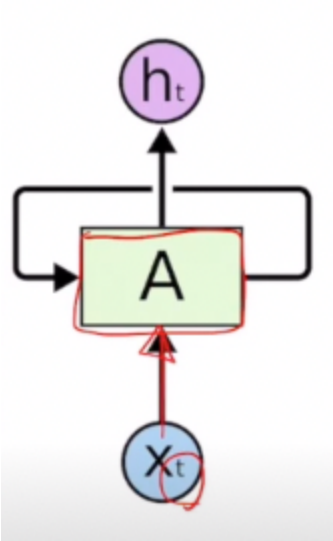
데이터의 순서가 주는 정보까지 인지해내는 새로운 형태의 신경망

'I live to eat'과 'I eat to love'은 같은 단어들의 조합으로 이루어져 있지만, 'live'와 'eat'의 위치가 바꾸면서 의미가 완전히 바뀌었다.

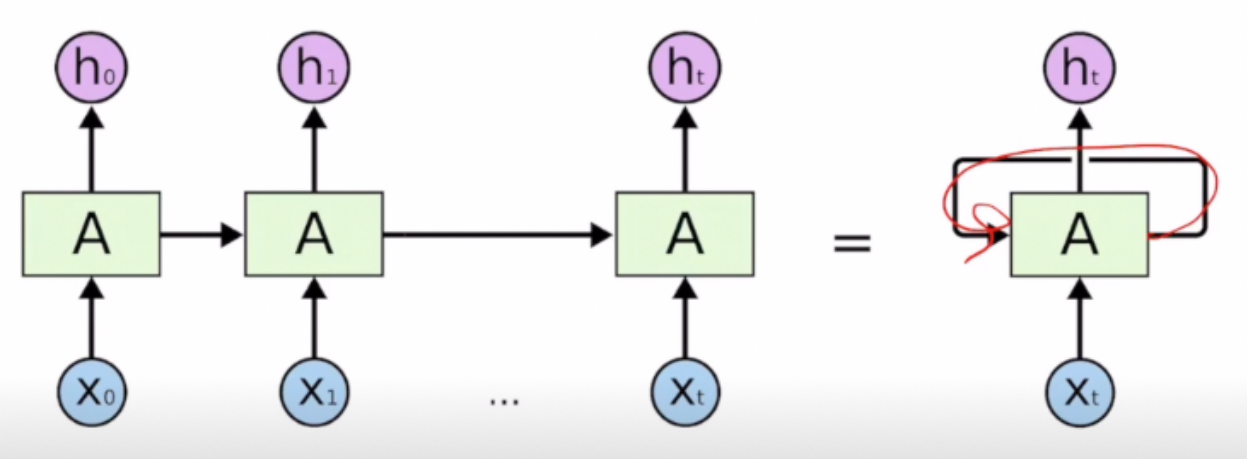
기존 신경망 vs RNN

기존 신경망은 이미지 같은 정적인 데이터를 인지하는 데 쓰였다. 즉, 데이터의 순서와 상호작용을 인식해 전체 상황을 이해하는 능력이 부족했다. (기존 신경망에서도 데이터에 position 정보를 가지는 position index을 추가해 학습할 수도 있지만 부족함이 있었다. 아래의 사진 참고)

RNN의 기본 구조



X_t가 cell A에 input으로 들어가면 A는 h_t을 출력하고, 다른 값을 다시 A에 집어넣는 구조(recurrent)이다. 학습 과정 내내 모델(A)의 파라미터를 공유하고 있다.



첫번째 입력값 X_0이 cell A에서 처리되어 h_0로 출력되며, 출력되지 않은 값이 다음 cell로 전달된다. 이렇게 출력되지 않고 전달되는 벡터를 hidden state (vector)라고 한다. 이러한 구조 덕분에 나중 값들은 이전 입력값들의 처리 결과를 반영할 수 있게 된다.

즉, 마지막 hidden state (vector) t는 모든 벡터들의 내용을 압축하고 있는 것이다.

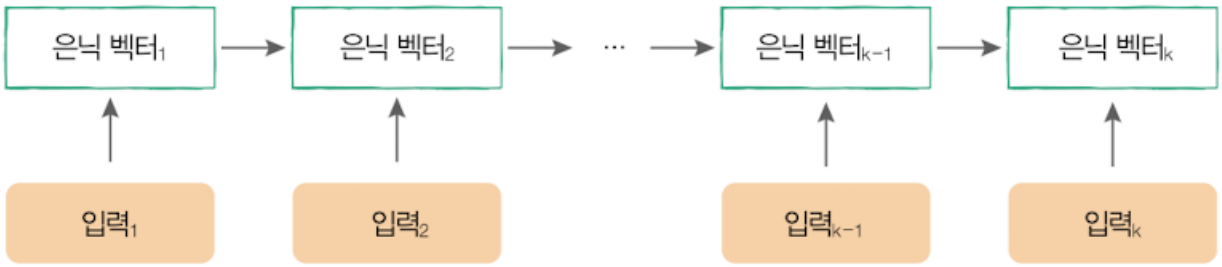


그림 7-1 RNN 구조

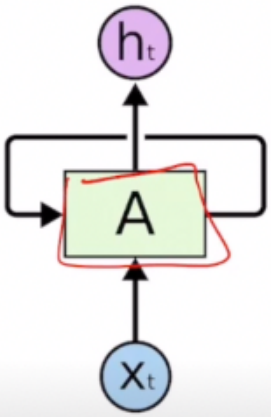
RNN 내부에서 일어나는 일

In A,

$$h_t = f(h_{t-1}, x_t)$$

For example,

$$h_t = \tanh(W_h h_{t-1} + W_x x_t)$$



이전 단계의 hidden state(h_{t-1})와 현재 단계의 입력값(x_t)을 함수의 입력값으로 받는다.

여러 형태의 RNN

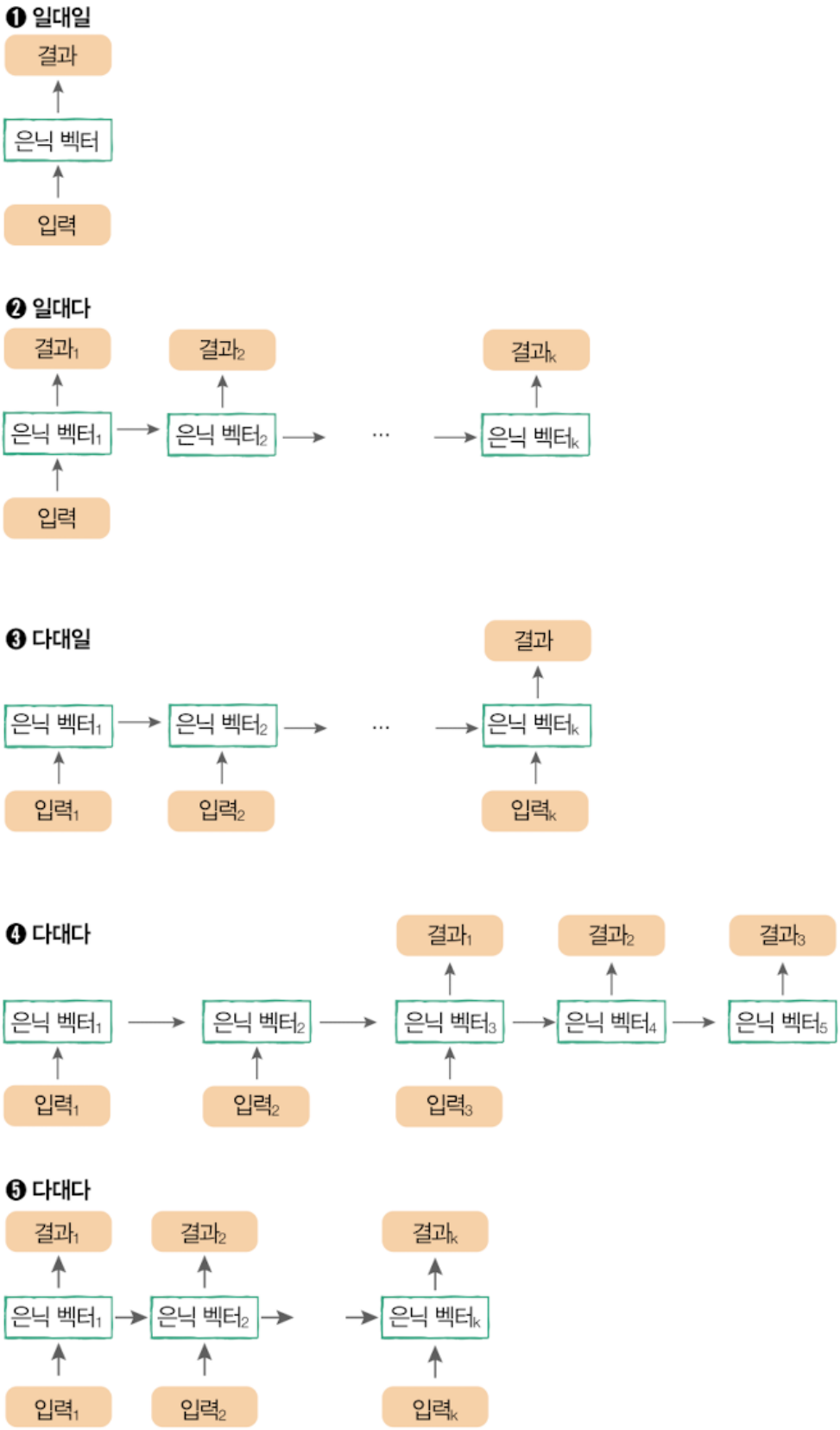
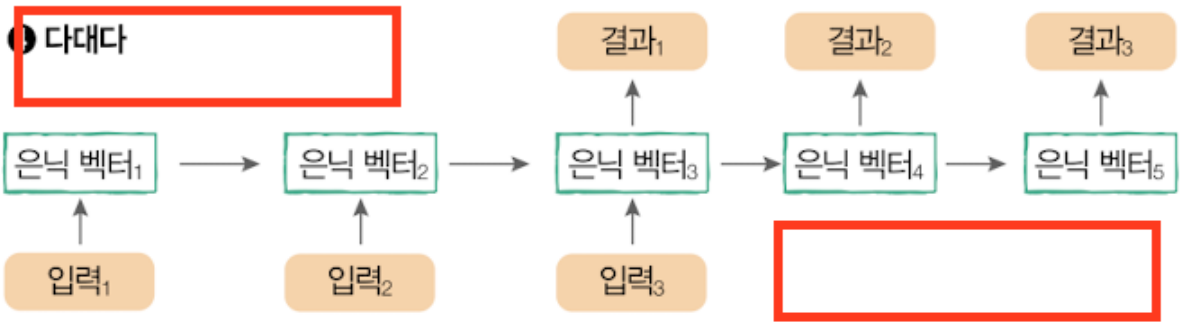


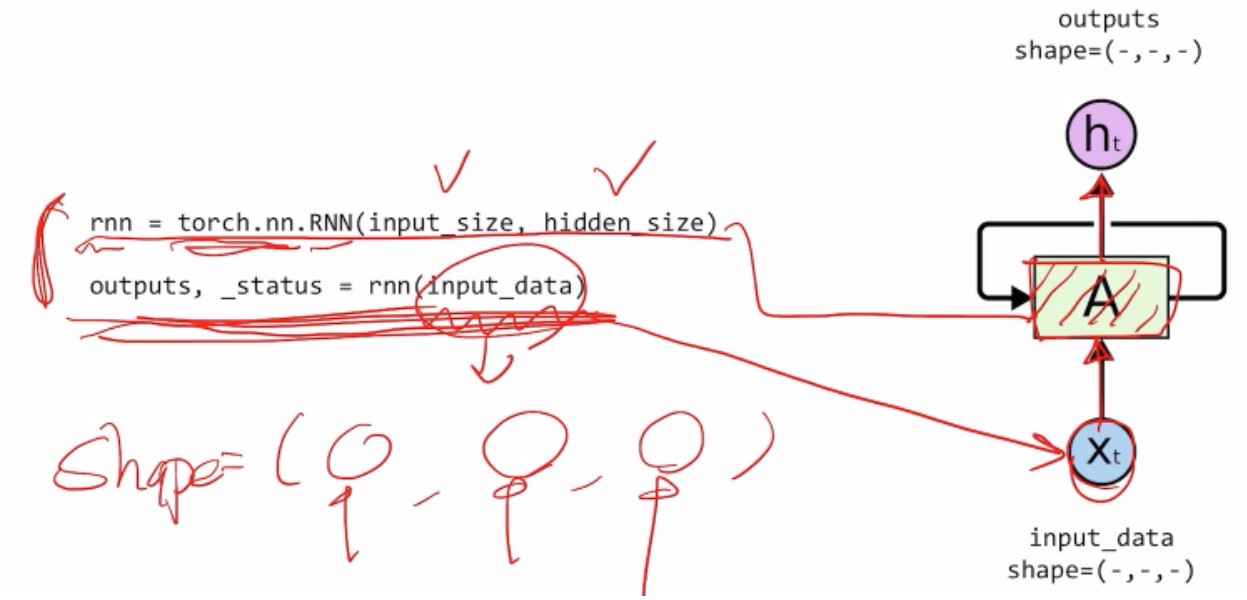
그림 7-2 여러 형태의 RNN

- 일대일(one to one) : 일반적인 인공신경망 (RNN 아님)
- 일대다(one to many) : e.g. 하나의 이미지 → 문장 (Image captioning)
- 다대일(many to one) : e.g 문장 → 감정 레이블 (sentiment analysis)
- 다대다(many to many) : e.g. 문장 → 문장 (translation, 문장을 끝까지 들은 다음 → 그 문장을 번역하기)



  - 사실 앞뒤 비어있는 부분에도 출력값이 있는데, 사용하지 않는다는 점이 특징이다.
- 다대다(many to many) : e.g. 비디오

PyTorch



- `torch.nn.RNN(input_size, hidden_size)` : cell A
- `rnn(input_data)` : X_t를 입력받아 h_t를 출력하는 과정 (여기서 input data는 (a, b, c)의 shape을 가지는 텐서)

문자열 'hello'를 벡터 형태로 만들어 보자

```
h = [1, 0, 0, 0]
e = [0, 1, 0, 0]
l = [0, 0, 1, 0]
o = [0, 0, 0, 1]
```

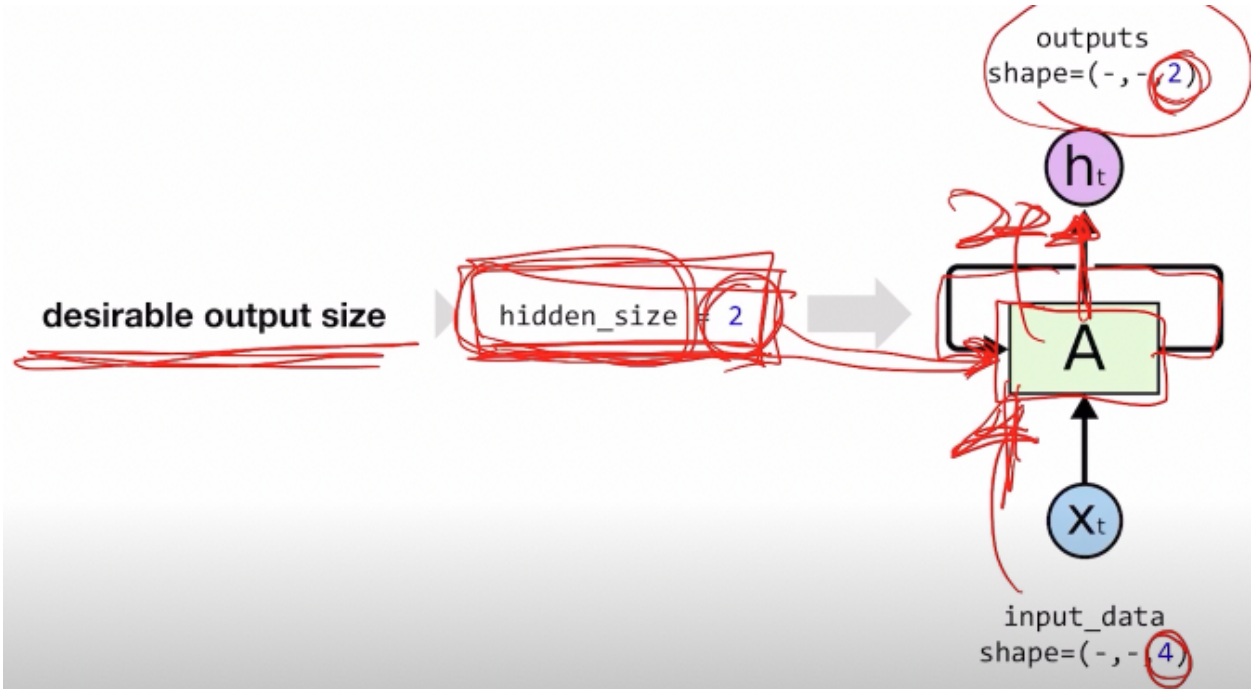
4개의 차원을 input으로 받는다는 점에서 `input_size = 4`이다. 여기에서는 one-hot-encoding을 사용했고, 실제로 다른 word embedding 같은 것을 사용하는 경우에는 그 embedding vector의 dimension이 `input_size`가 되겠다.

`torch.nn.RNN`이 `input_data`로 받는 데이터의 구조는 (batch size, sequence length, input_size)이다. 마지막 `input_size` 파라미터의 값은 방금 구한 셈! (정확한 설명은 다음의 문서 참고https://pytorch.org/docs/stable/generated/torch.nn.RNN.html)

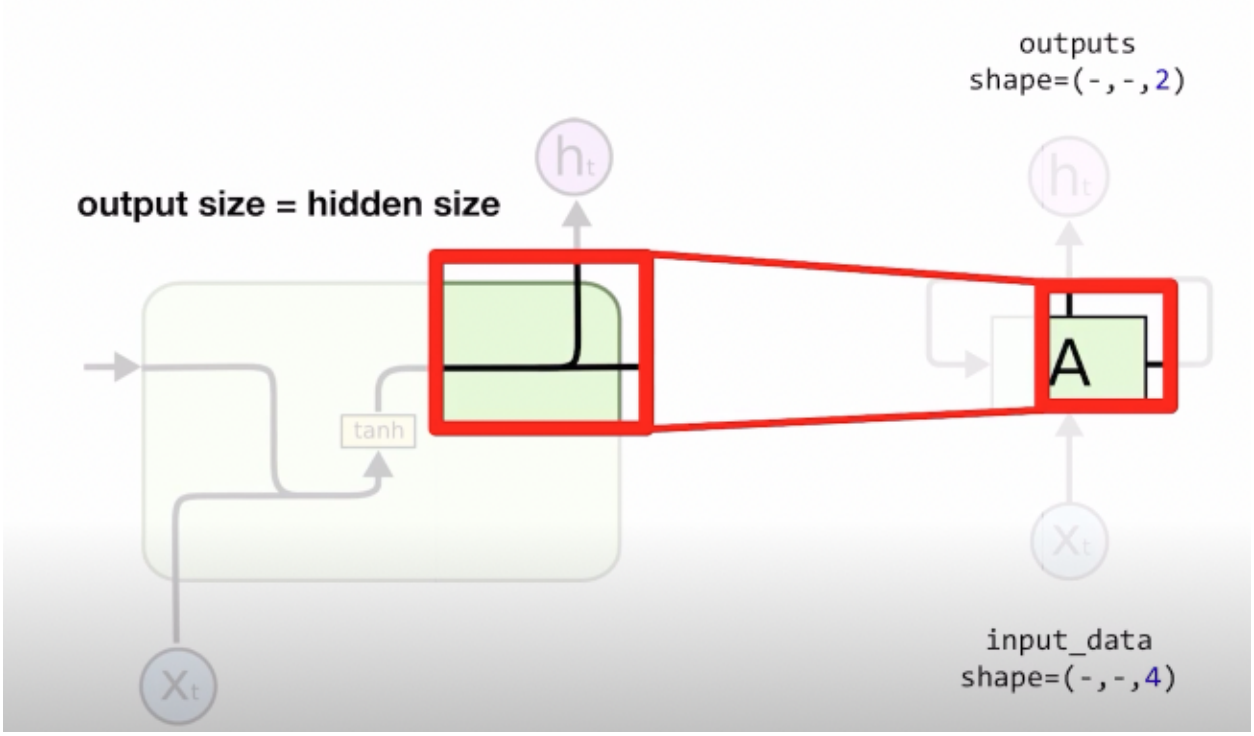
`torch.nn.RNN`이 `output_data`로 뱉는 데이터의 구조는 (batch size, sequence_lenth, hidden_size)이다. 몇차원의 벡터를 원하는지(예를 들어 몇 개의 레이블을 원하는지)에 따라 `hidden_size`파라미터를 조정하면 된다.

hidden state와 output size?

어떻게 hidden size가 output size와 동일한 값을 가지게 되는 것일까?



이는 RNN 셀 내부를 보면 쉽게 이해할 수 있다.

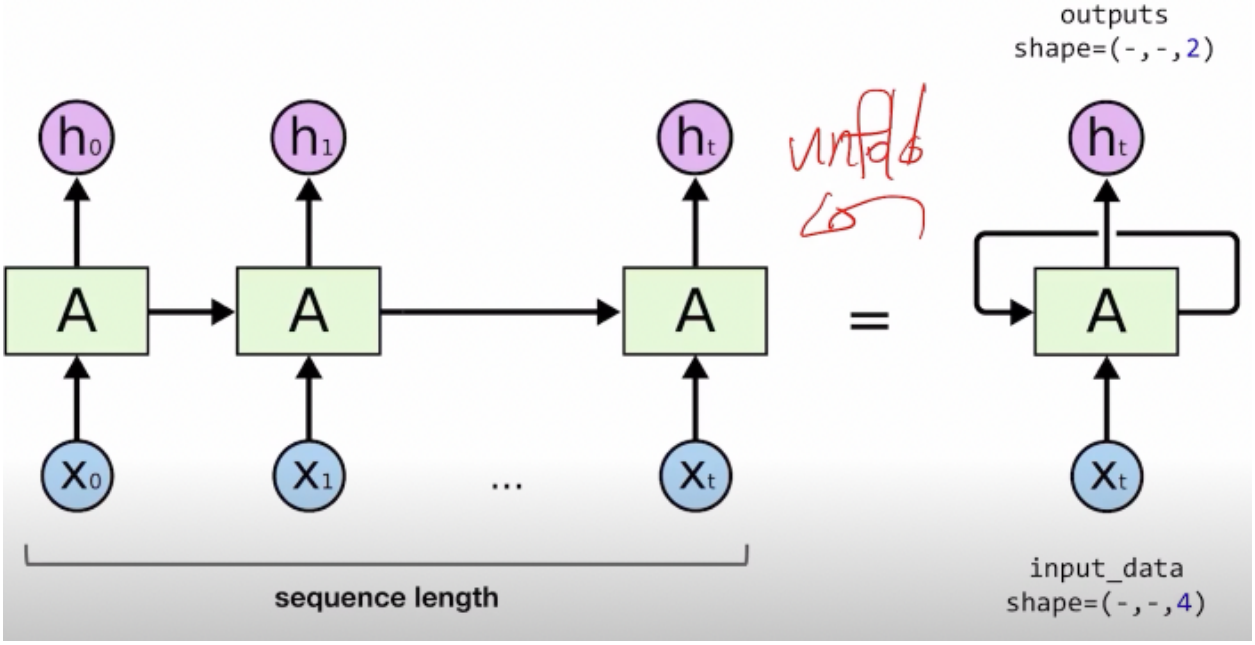


출력 직전에 같은 값이 2개의 가지로 분기된다(output, hidden state). Hidden state는 output으로 출력되지 않고 다시 input으로 들어오기 때문에 다시 유입된다. 따라서 output size =

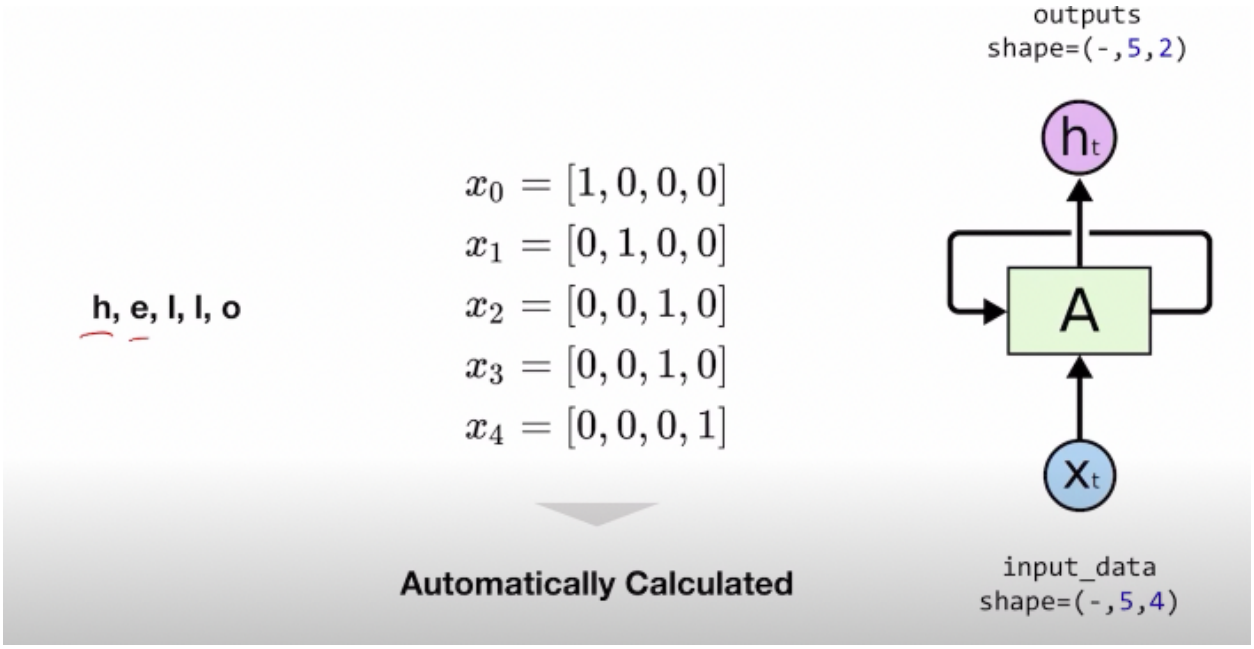
hidden size가 된다.

Sequence length?

RNN이 sequence data를 입력받는다는 점에서 sequence length 파라미터가 존재한다. RNN의 구조를 다음의 사진처럼 펼쳐보면 총 $t+1$만큼의 sequence length가 생긴다.

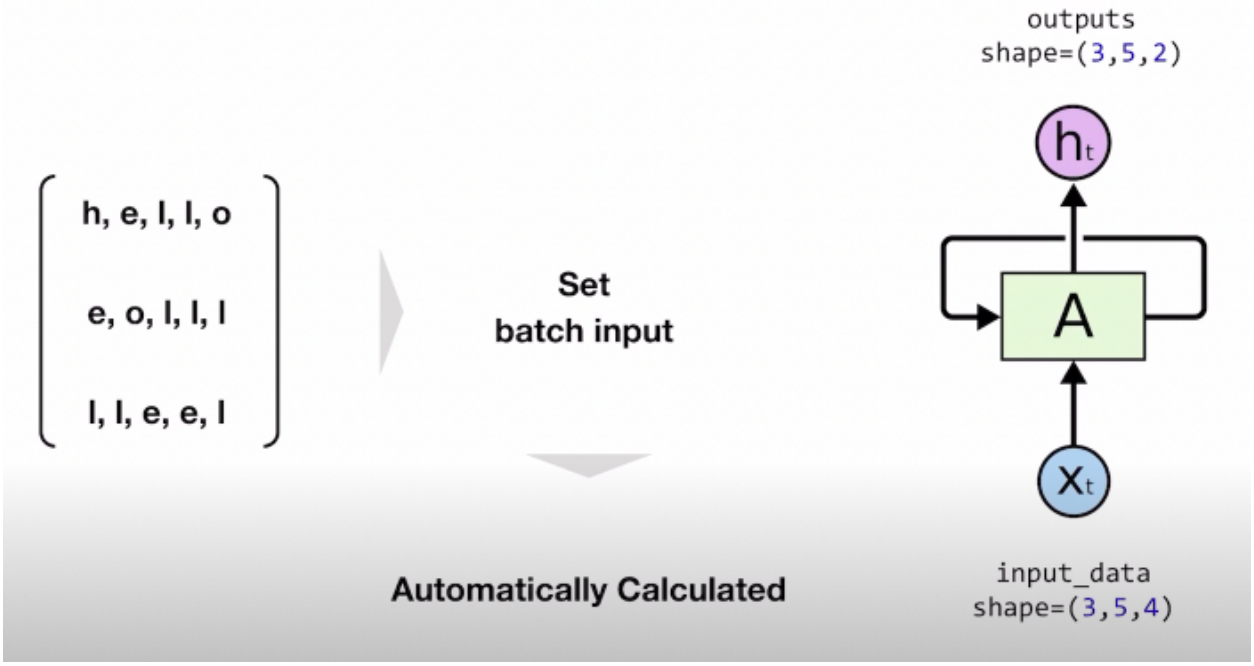


이를 hello 예시에 적용해 보자.



- `input_data` : sequence length 5, input size 4
- `outputs` : sequence length 5, hidden size 2

Batch size



```
input_data_np = np.array([[h, e, l, l, o],
                          [e, o, l, l, l],
                          [l, l, e, e, l]], dtype=np.float32)
```

PyTorch RNN

```
# declare dimension
input_size = 4
hidden_size = 2


# transform as torch tensor
input_data = torch.Tensor(input_data_np)


# declare RNN
rnn = torch.nn.RNN(input_size, hidden_size)


# check output
outputs, _status = rnn(input_data)


print(outputs)
print(outputs.size())


>>>
tensor([[[-0.7497, -0.6135],
         [-0.5282, -0.2473],
         [-0.9136, -0.4269],
         [-0.9136, -0.4269],
         [-0.9028,  0.1180]],

        [[-0.5753, -0.0070],
         [-0.9052,  0.2597],
         [-0.9173, -0.1989],
         [-0.9173, -0.1989],
         [-0.8996, -0.2725]],

        [[-0.9077, -0.3205],
         [-0.8944, -0.2902],
         [-0.5134, -0.0288],
         [-0.5134, -0.0288],
         [-0.9127, -0.2222]]], grad_fn=<StackBackward>)
torch.Size([3, 5, 2]) # batch size, sequence length, hidden state
```