RNN(Recurrent Neural Network)

이번 글에서는 RNN의 개요, Single-layered RNN과 Multi-layered RNN, 마지막으로 Bidirectional Multi-layered RNN(Bidirectional RNN)에 대해 살펴보려고 한다.

RNN 개요

RNN(Recurrent Neural Network)은 sequential한 data, 즉, time step이 존재하는 data를 다루는데에 최적화 된 아키텍쳐이다.

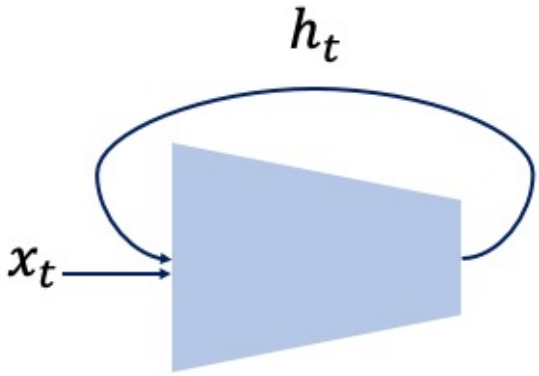
이러한 특성 때문에, RNN계열의 아키텍쳐(LSTM, GRU)는 시계열 데이터(주식 데이터, 센서 데이터), 텍스트 데이터, 영상 혹은 음성 데이터에 폭넓게 사용된다.

시계열 데이터의 경우, 데이터 안에 있는 특정 사건이 발생한 시각 정보가 매우 중요한 데이터이며, 텍스트 데이터, 영상 혹은 음성 데이터와 같은 sequential 데이터의 경우에도 순서가 매우 중요한 데이터이다.

텍스트 데이터를 예로 들자면, 같은 단어여도 문맥에 따라 의미가 달라지는 경우가 있기 마련이다. 이 때, 문맥을 고려하지 않고 데이터를 처리한다면, 실제 사용되는 뜻과 매우 다른 뜻으로 처리될 가능성이 있다. 그래서, 순서를 고려하여 데이터를 처리하는 것이 매우 중요하다.

그렇다면, RNN은 어떠한 이유로 인해 이러한 timestep, 혹은 순서를 잘 처리할까?

RNN을 간단하게 나타내면 다음과 같다.



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

RNN은 output y라는게 따로 존재하는 대신, h_t(해당 timestep의 hidden state)를 output으로 출력한다.

그런데, 입력으로는 x_t(해당 timestep의 input)과 h_{t-1}(이전 timestep의 hidden state)을 같이 받는다.

이렇게, 이전 timestep의 output(hidden state)를 현재 timestep의 입력으로 받는 특성 덕분에, 위에서 말한 timestep이 존재하는 데이터들을 효과적으로 다룰 수 있다.

조금 더 자세히 살펴보기 위해 수식과 함께 살펴보자.

$$\begin{aligned} \hat{y}_t = h_t &= f(x_t, h_{t-1}; \theta) \\ &= \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{t-1} + b_{hh}) \\ &\text{where } \theta = \{W_{ih}, b_{ih}, W_{hh}, b_{hh}\} \end{aligned} \tag{1}$$

위에서 살펴본 그림을 수식으로 풀어본 결과이다.

전에 언급한 대로, 입력으로는 x_t(해당 timestep의 input)과 h_{t-1}(이전 timestep의 hidden state)을 같이 받아서 h_t(해당 timestep의 hidden state)를 output으로 출력한다.

우리는 입력으로 x_t, h_{t-1} 2개를 받았다. 여기서 x_t에는 weight W_{ih}를 곱하고 bias b_{ih}를 더해주고,

h_{t-1}에는 weight W_{hh}를 곱하고 $biasb_{hh}$를 더해준다.

즉, $W_{ih}x_t + b_{ih} + W_{hh}h_{t-1} + b_{hh}$ 부분에서는 linear layer 2개가 있는것처럼 보여진다.

그런데, **이러한 과정만으로는 non-linearity(비선형성)을 담을 수 없기에, 탄젠트-하이퍼볼릭 함수를 non-linear activation functiond으로 적용시켜준다.**

결과적으로 RNN의 parameter θ는 $W_{ih}, b_{ih}, W_{hh}, b_{hh}$ 로 이루어져 있는 것이다.

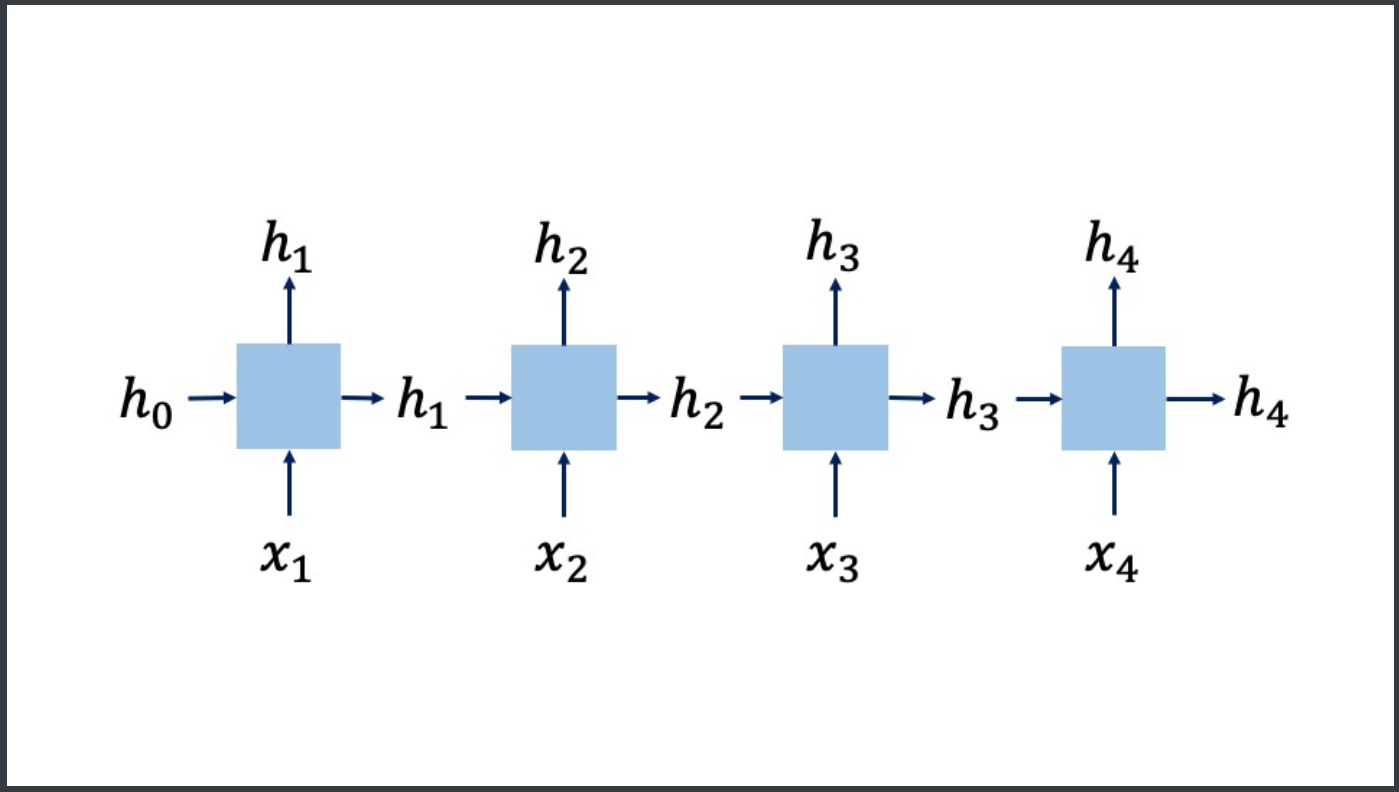
그렇다면, 여기서 weigth parameter의 shape은 어떻게 될까?

$$\begin{aligned} W_{ih} \in \mathbb{R}^{n \times d}, W_{hh} \in \mathbb{R}^{n \times n} \\ = W \in \mathbb{R}^{(n+d) \times d} \\ \text{where } h_t \in \mathbb{R}^d, x_t \in \mathbb{R}^n \end{aligned} \tag{2}$$

hidden stateh_t가 d차원, x_t가 n차원이라고 할 때, W_{ih} 는 (n, d), W_{hh} 는 (d, d)의 shape을 가지게 된다.

Single-layered RNN

그렇다면, 이러한 RNN을 사용한 Single-layered RNN부터 살펴보도록 하겠다. 위 예시처럼 RNN을 나타낼 수 있지만, RNN layer의 내부 동작을 좀 더 직관적으로 볼 수 있게끔 unrolling하여 나타낼 수도 있다.



다음과 같이 RNN을 나타내보았다. 전에 설명한 것과 같이, input으로 x_t와 이전 timestep의 hidden state h_{t-1}를 받는 것을 확인할 수 있다.

그런데, 위의 표현 방식으로 RNN을 나타낼 때 주의해야 할 점이 있다. unrolling하다보니, 각각의 RNN이 다른 parameter θ를 가지는 다른 RNN으로 보일 수 있다는 점이다.

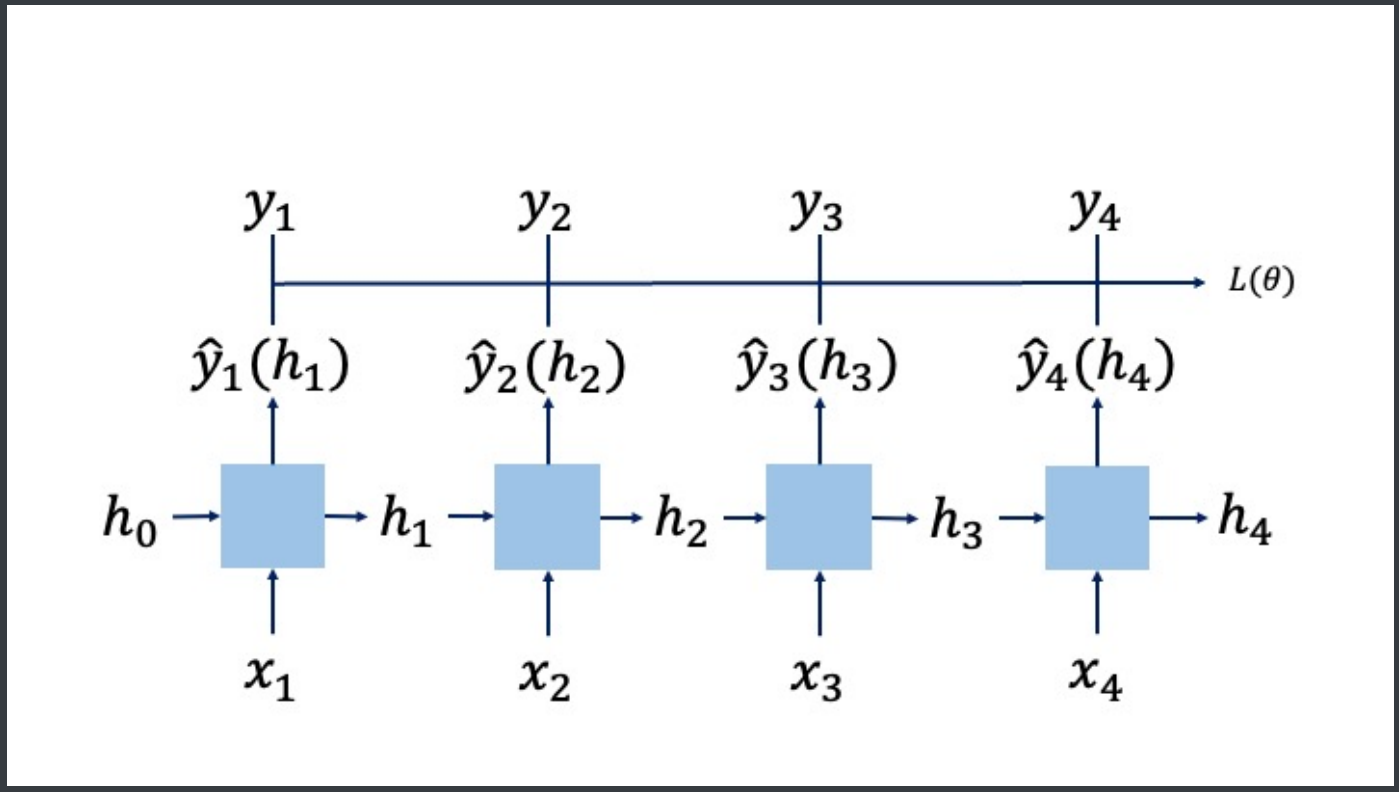
$$\begin{aligned} \hat{y}_t = h_t &= f(x_t, h_{t-1}; \theta) \\ &= \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{t-1} + b_{hh}) \\ &\text{where } \theta = \{W_{ih}, b_{ih}, W_{hh}, b_{hh}\} \end{aligned} \tag{3}$$

위 수식에서 확인할 수 있는것처럼, RNN의 parameter θ는 timestep에 영향을 받지 않는다.

즉, parameter θ는 timestep에 관계없이 같다는 것이다.

위 예시에서는 파란색 사각형을 의미하는 것이며, **각각의 파란색 사각형은 여러개처럼 보이지만 실제로는 같은 parameter를 가지는, 같은 RNN이다.**

그렇다면, 이제 RNN이 동작하는 전반적인 과정을 살펴보겠다.



우선, input으로 x_t(해당 timestep의 input)과 h_{t-1}(이전 timestep의 hidden state)을 같이 받는다.

input은 parameter θ를 가진 RNN을 거쳐 output을 내놓게 되는데, RNN은 output y라는게 따로 존재하는 대신, h_t(해당 timestep의 hidden state)를 output으로 출력한다고 언급했었다.

즉, 각 timestep에서의 output $\hat{y}(h_1)$은 각 timestep의 hidden state h_t이다.

이러한 각 timestep에서의 output $\hat{y}(h_1)$은 다음 timestep의 input으로 들어감과 동시에, 해당 timestep에서의 ground truth인 y_1과 함께 loss를 계산하게 된다.

이후, 각 timestep t의 output $\hat{y}(h_t)$과 해당 timestep에서의 ground truth인 y_t 차이인 loss를 모든 timestep에 대해 최소화하도록 하여 학습한다

정리하자면 다음과 같다.

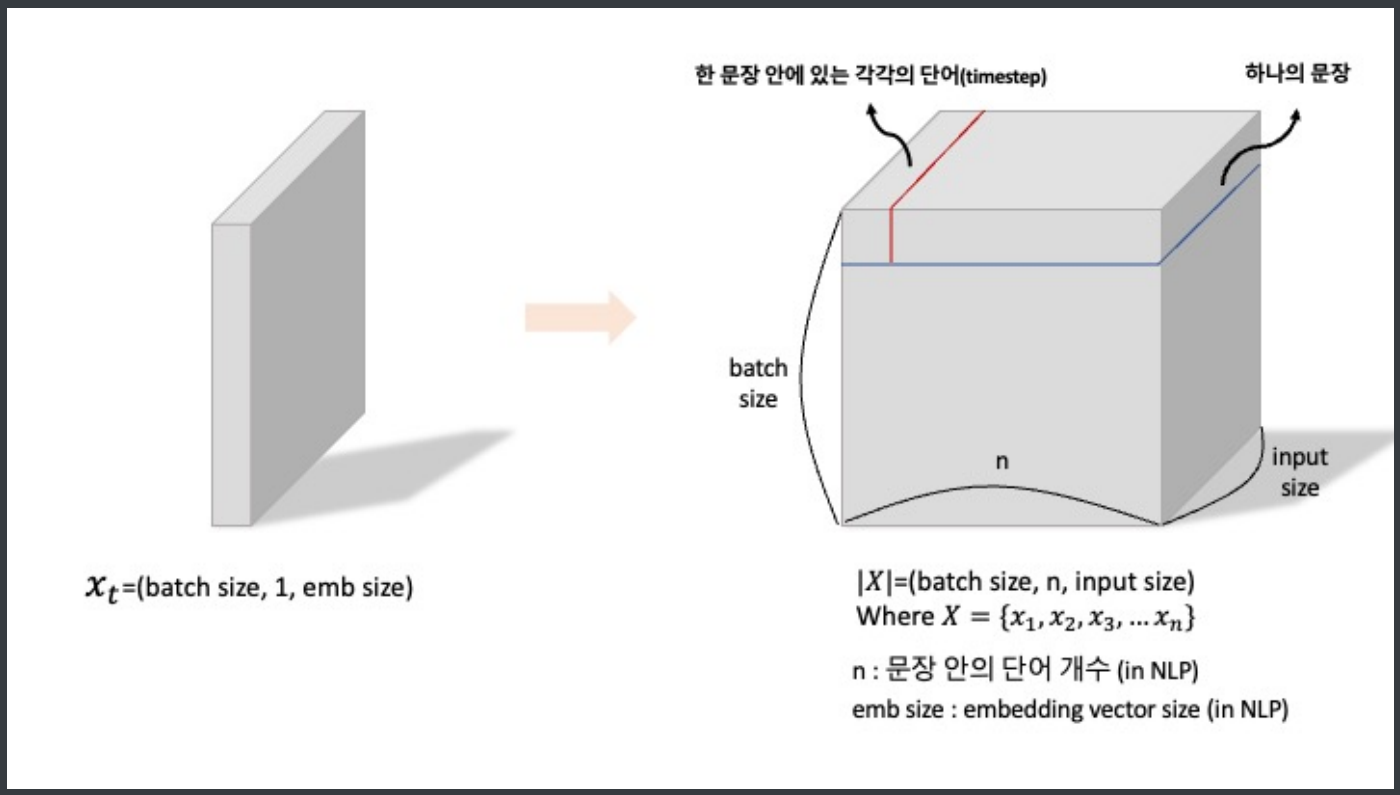
1. input으로 x_t(해당 timestep의 input)과 h_{t-1}(이전 timestep의 hidden state)을 같이 받는다.
2. input은 parameter θ를 가진 RNN을 거쳐 h_t(해당 timestep의 hidden state)를 output으로 출력
3. 각 timestep에서의 output $\hat{y}(h_t)$은 다음 timestep의 input으로 들어감과 동시에, 해당 timestep에서의 ground truth인 y_t과 함께 loss를 계산
4. loss를 모든 timestep에 대해 최소화하도록 하여 학습

Input tensor와 Hidden tensor의 shape

금방까지 RNN이 동작하는 전반적인 과정을 살펴보았다. 그런데, 이렇게 동작하는 과정에서 우리의 데이터들은 어떤 형태를 가지고 입력되어지며, 어떠한 형태로 출력되어질까? 그림과 함께 살펴보도록 하겠다.

(예시는 NLP를 기반으로 작성되어졌다)

먼저 Input tensor이다.



한 문장 안에는 여러 단어들이 있다. 즉, 하나의 문장은 여러 개의 단어들로 구성되어 있다는 것이다. 이 때, 문장을 구성하는 단어의 개수를 n개라고 한다. 이 단어의 개념은 timestep의 개념과 동일하다. 또한, emb size의 경우, 한 단어를 나타내는 embedding vector의 size를 나타낸다.

그림의 왼쪽은 하나의 timestep에서의 input tensor, 오른쪽은 모든 timestep에서의 input tensor를 나타낸다.

왼쪽은 emb size를 가진 하나의 단어(하나의 timestep)가 batch size만큼 쌓여있고,

오른쪽 input tensor는 한 문장 안에 emb size를 가진 단어 n개가 있고, 이 각각의 문장들이 차곡차곡 batch size만큼 쌓여있다.

즉, input tensor는 하나의 timestep에서는

(batch size, 1개의 단어(하나의 timestep), 한 단어를 나타내는 embedding vector의 size)의 shape인

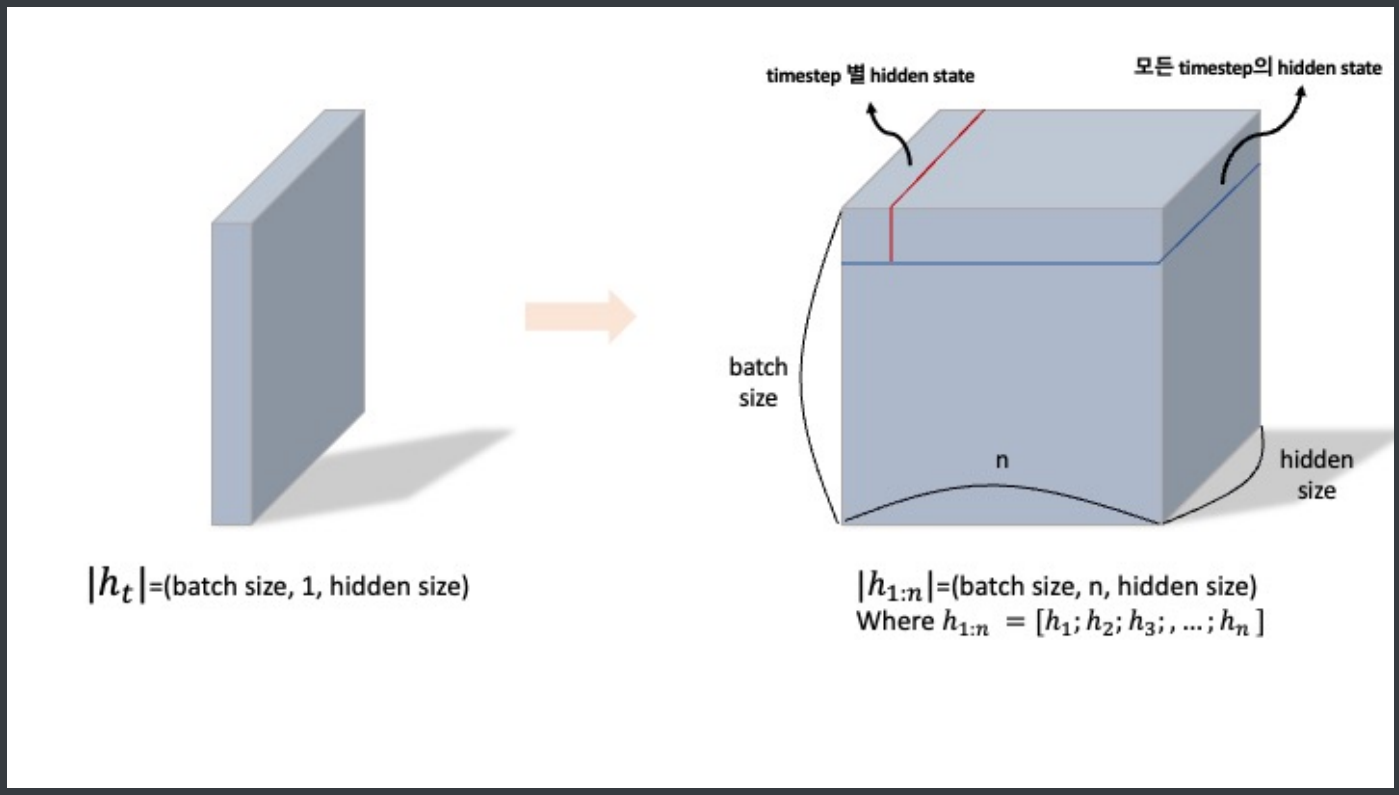
(batch size, 1, emb size)의 shape을 가지고 있으며,

모든 timestep에서는

(batch size, 문장을 구성하는 단어의 개수, 한 단어를 나타내는 embedding vector의 size)의 shape인

(batch size, n, emb size)의 shape을 가지고 있다.

이러한 상태로 RNN에 입력되어지면 Hidden tensor가 나오게 되는데, hidden tensor의 shape에 대해서도 알아보자.



input tensor때와 마찬가지로,

그림의 왼쪽은 하나의 timestep에서의 hidden tensor, 오른쪽은 모든 timestep에서의 hidden tensor를 나타낸다.

hidden tensor는 하나의 timestep에서는

(batch size, 하나의 timestep, hidden state의 size)의 shape인

(batch size, 1, hidden size)의 shape을 가지고 있으며,

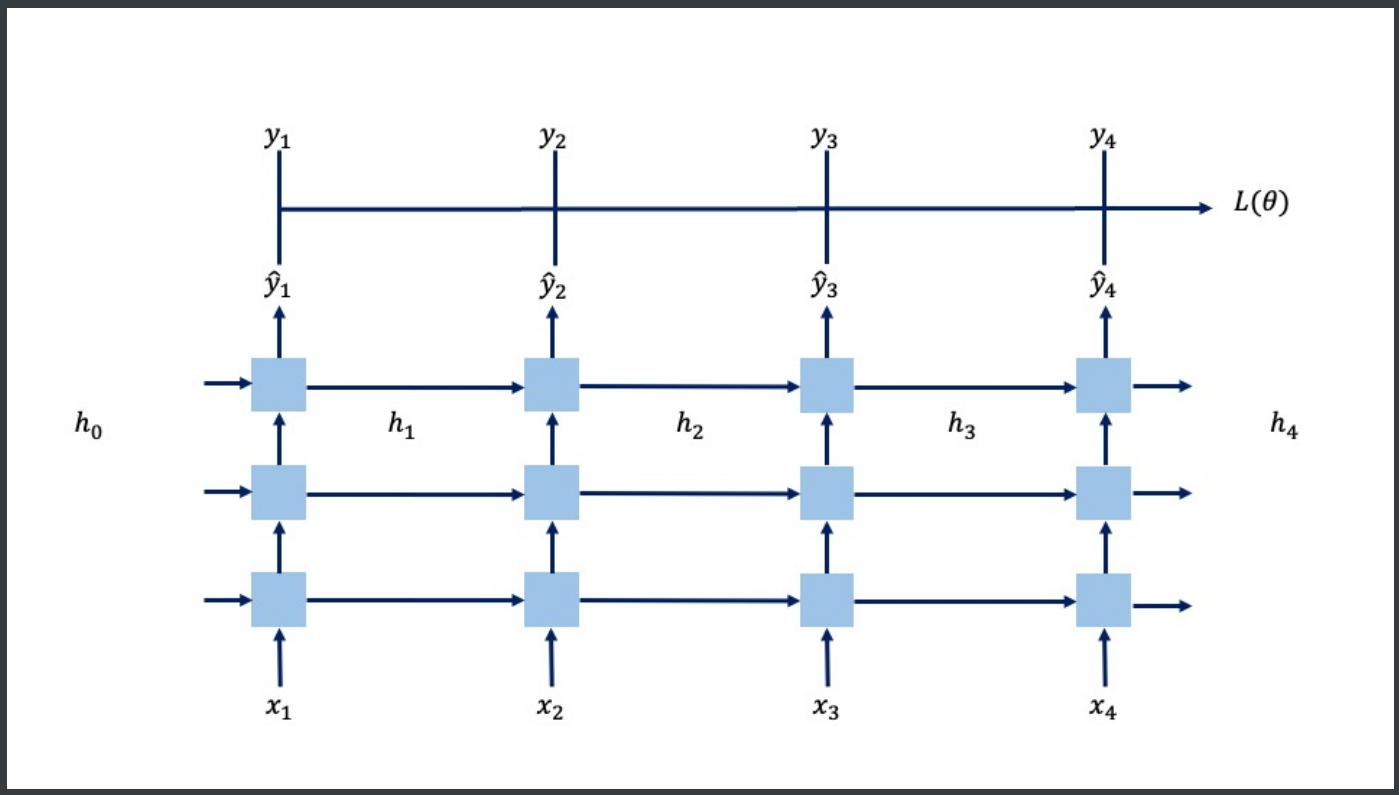
모든 timestep에서는

(batch size, 모든 timestep, hidden state의 size)의 shape인

(batch size, n, hidden size)의 shape을 가지고 있다.

Multi-layered RNN

이제까지 하나의 layer를 가진 Single-layered RNN에 대해 살펴보았다. 지금부터는 여러 layer를 가진 Multi-layered RNN에 대해 살펴보려고 한다. 언제나 그랬듯이 그림과 함께 살펴보자.



사실, Multi-layered RNN이라 해서 거창한 무언가가 있는 것이 아니라, Single-layered RNN을 여러 겹 쌓은 것이다.

다만 주의해야할 점이 있다.

위에서 언급했던 내용들 중, parameter θ는 timestep에 관계없이 같으며, 각각의 파란색 사각형은 여러 개처럼 보이지만 실제로는 같은 parameter를 가지는, 같은 RNN이라는 내용이 있었다.

여기서도 마찬가지이긴 한데,

같은 층(layer)끼리는 parameter θ가 같지만, 다른 층(layer)끼리는 서로 parameter θ가 다르다!

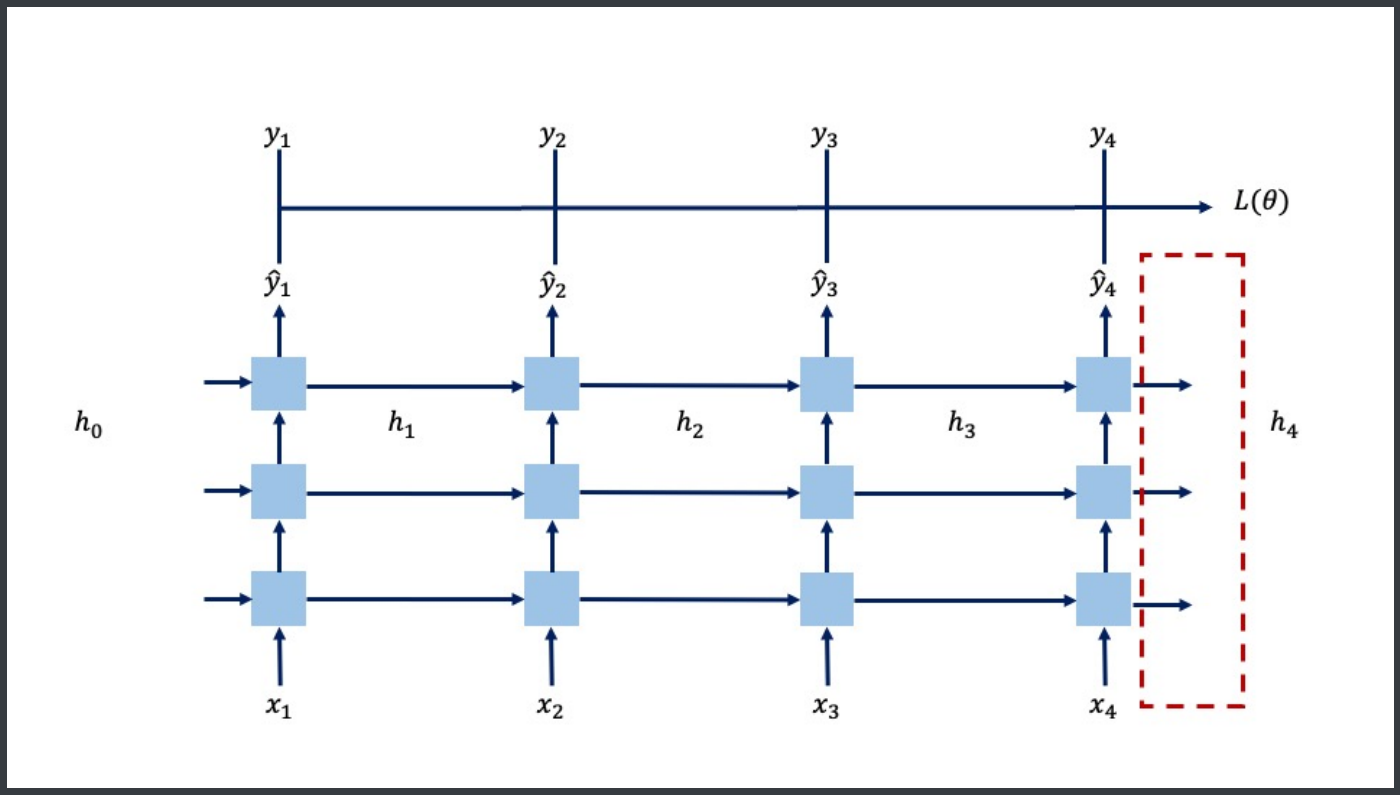
또한, **Multi-layered RNN에서의 hidden state는, 각 timestep별 모든 layer의 hidden state를 의미하며,**

Multi-layered RNN에서의 output은, 모든 timestep별 마지막 layer의 hidden state를 의미한다.

이 부분은 혼동될 수 있으니 그림과 함께 하나씩 차근차근 살펴보겠다.

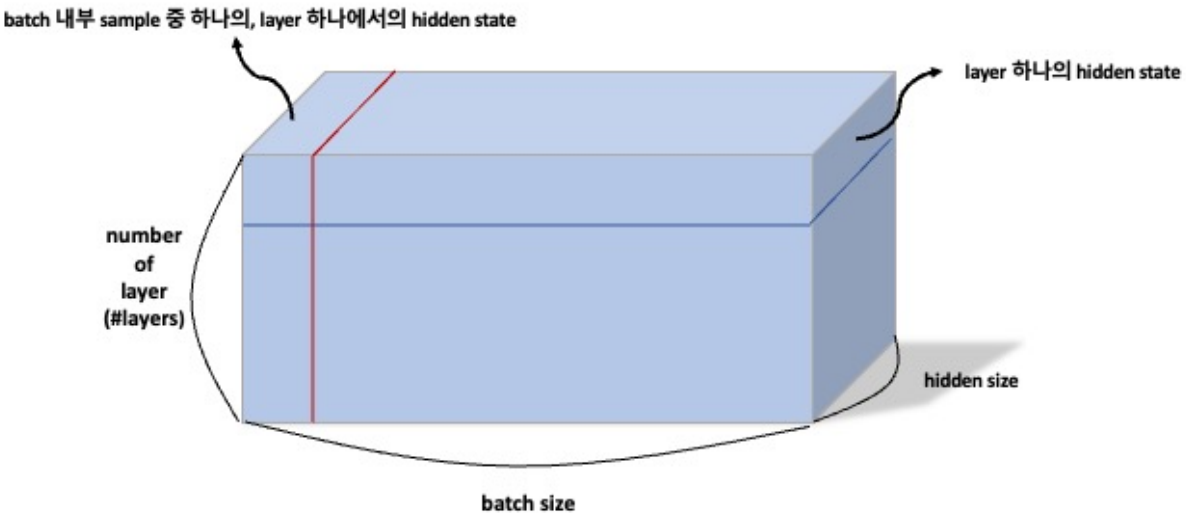
Hidden tensor와 Output tensor의 shape

먼저, hidden state부터 살펴보도록 하겠다.



빨간색 점선 박스가 **Multi-layered RNN에서의 hidden state이다.**

shape 모형은 다음과 같다.



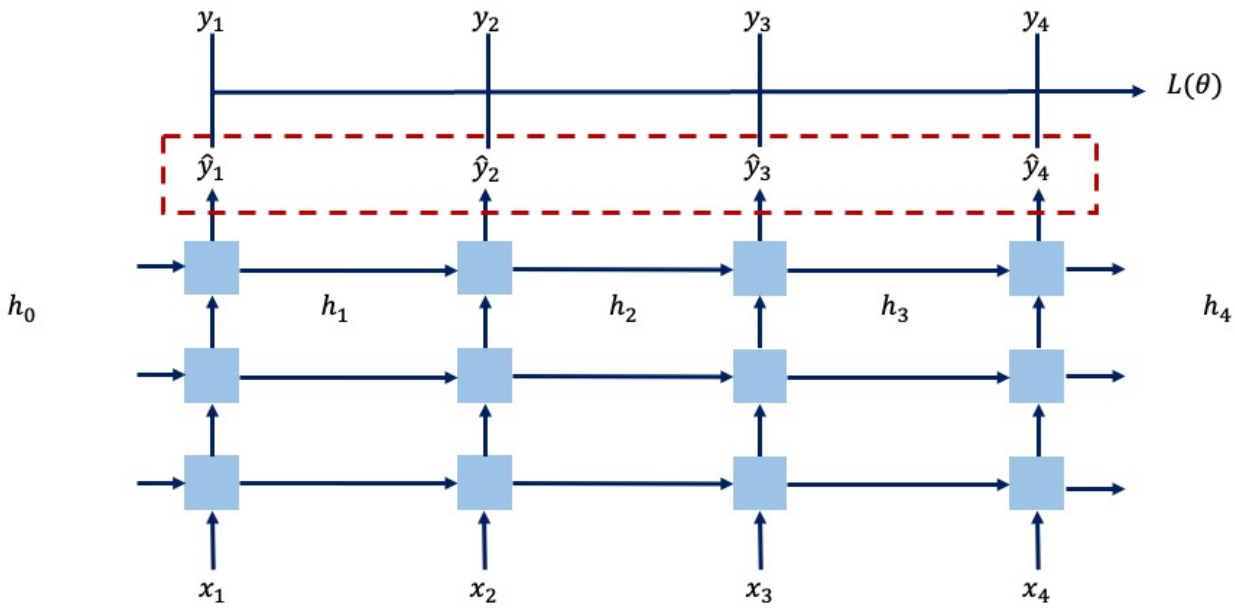
$|h_t|$=(#layers, batch size, hidden size)

우선, shape을 살펴보기 전에 염두에 둬야할 점이 있다. hidden state의 경우, 하나의 timestep의 모든 layer의 hidden state인데,

즉, 이 말은 hidden state는 결국 하나의 timestep을 정해놓고 조회한다는 것이다. 하나의 hidden state의 shape 모형에는 timestep에 대한 정보가 없다.

그래서, 이전까지 봐왔던 shape 3차원 예제들에서 가로(row)가 timestep을 담당했기에, 이 Multi-layered RNN에서의 hidden state의 shape 모형을 처음 봤을 때 혼란스러울 수 있다.

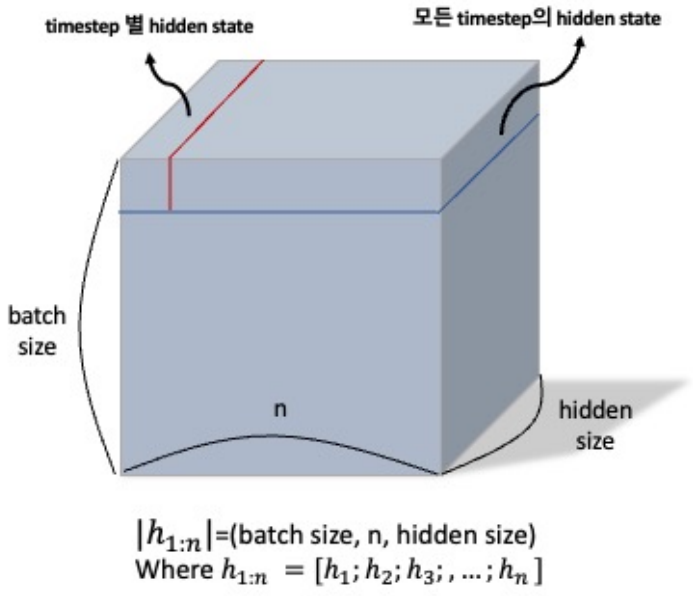
다음으로는 output을 살펴보겠다.



마찬가지로, 빨간색 점선 박스가 **Multi-layered RNN에서의 output이다.**

output의 shape의 경우에는, 결국 마지막 layer의 모든 timestep에서의 hidden state이기에 Single-layered RNN와 차이가 없다.

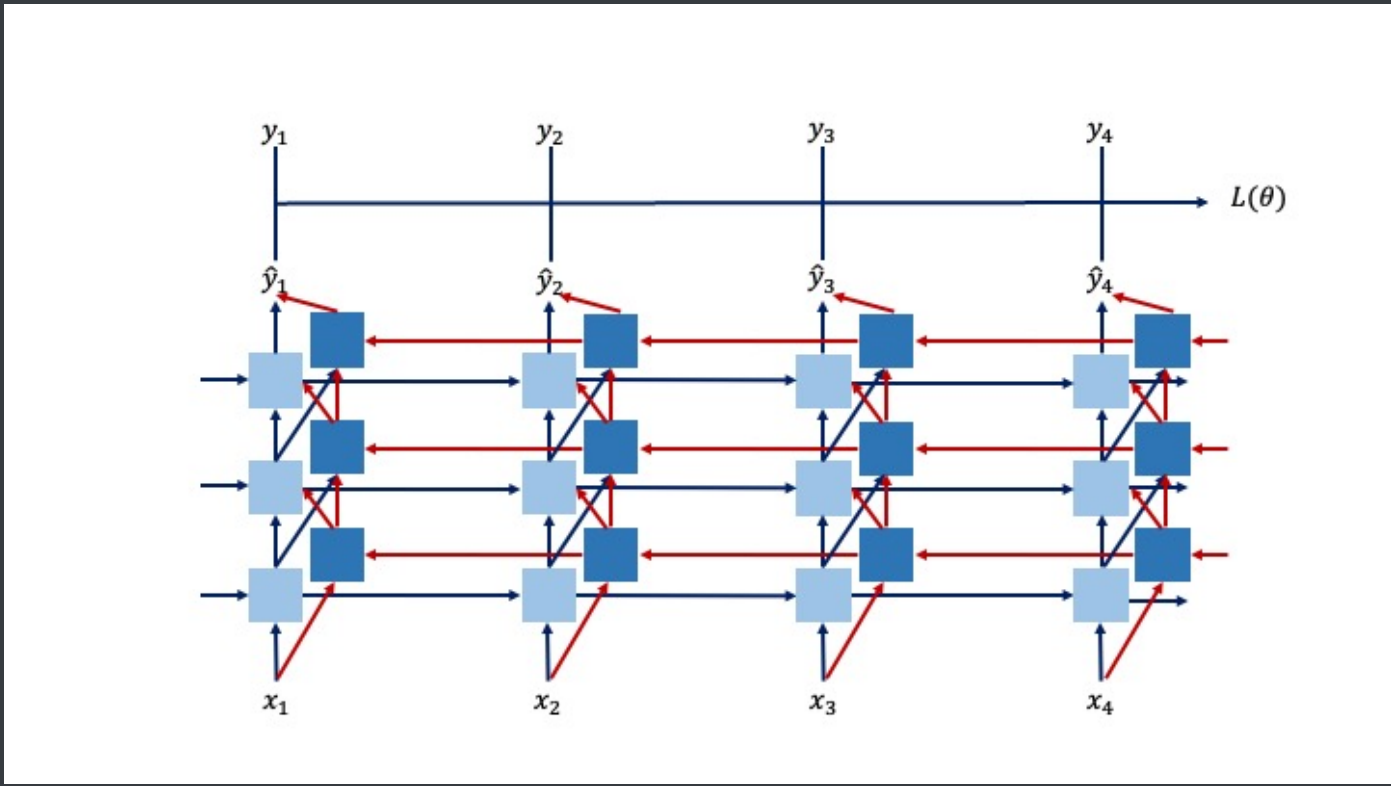
즉 다음과 같다.



Bidirectional Multi-layered RNN(Bidirectional RNN)

이제까지의 RNN들은 이전 timestep의 output이 다음 timestep의 input으로 전달되는 방향이 오로지 하나의 방향, 단방향이었다.

Bidirectional RNN은, 이러한 단방향을 넘어서 정방향과 역방향 총 2개의 방향으로 hidden state를 전달한다.



그림을 보면, 두 가지 방향으로 서로 연결되어 있음을 확인할 수 있다. (파란색 화살표가 정방향, 빨간색 화살표가 역방향이다.)

Bidirectional Multi-layered RNN에서의 hidden state도, 각 timestep별 모든 layer의 hidden state를 의미하며,

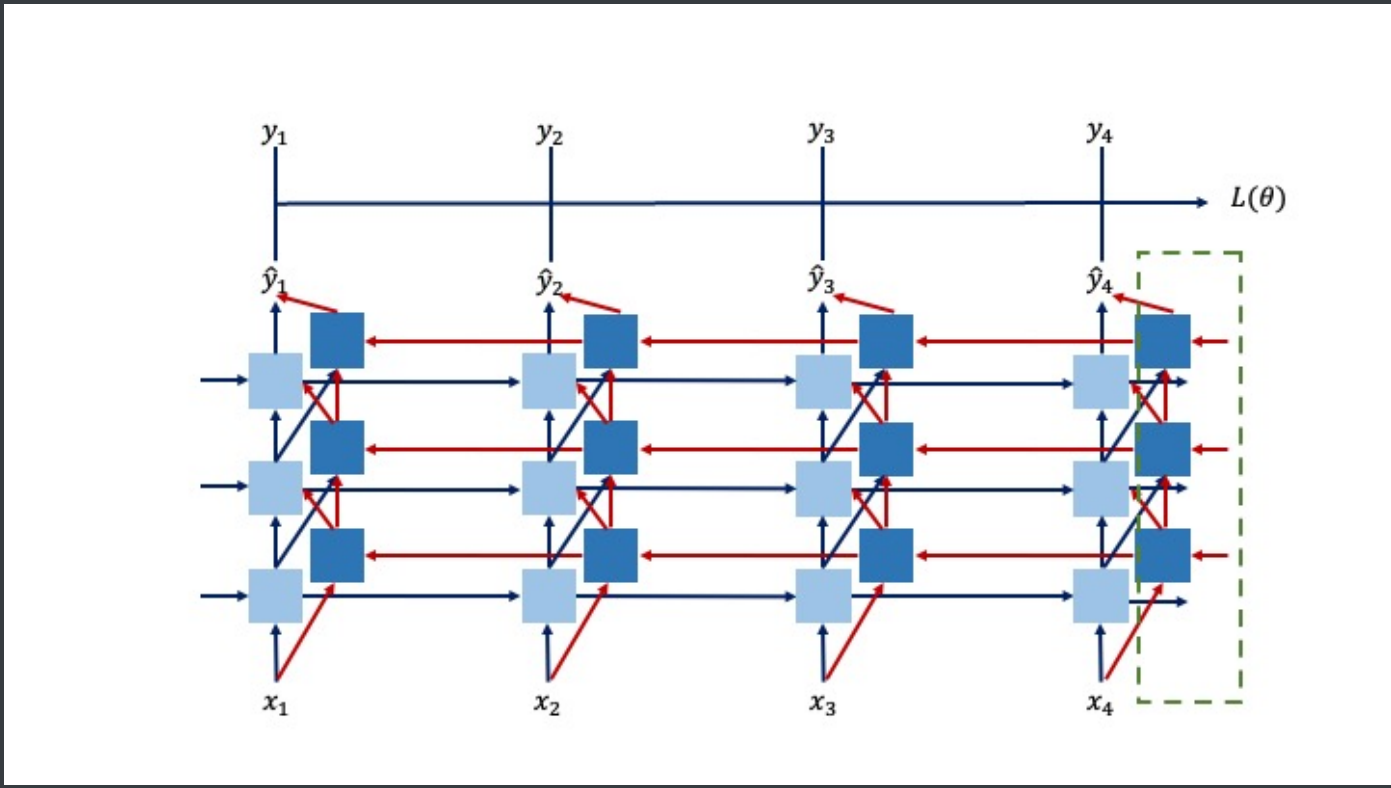
Multi-layered RNN에서의 output은, 모든 timestep별 마지막 layer의 hidden state를 의미한다.

그러나, 일반 **Multi-layered RNN 살짝 다른 점이 있는데 이 부분은 아래에서 더 살펴보겠다.**

Hidden tensor와 Output tensor의 shape

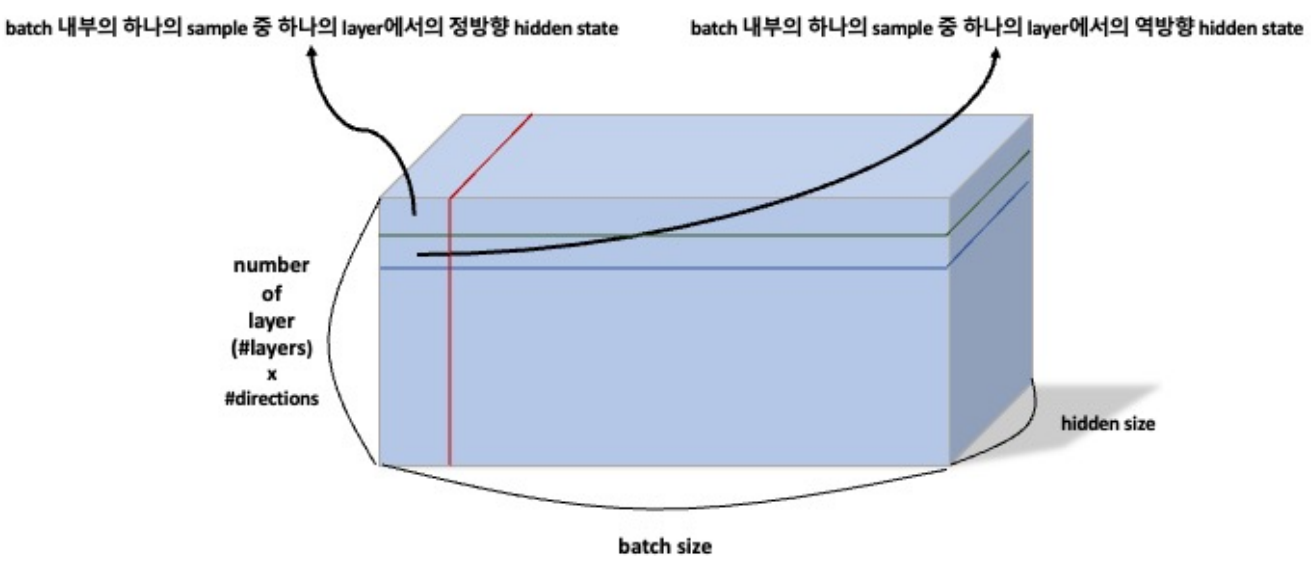
Bidirectional Multi-layered RNN에서는 기본적으로 Multi-layered RNN과 비슷하지만, 정방향과 역방향이 추가되었기 때문에 각각의 요소가 2배가 되었다는 차이가 있다.

먼저 hidden state이다.



초록색 점선 박스가 **Bidirectional Multi-layered RNN***에서의 hidden state이다.**

shape 모형은 다음과 같다.

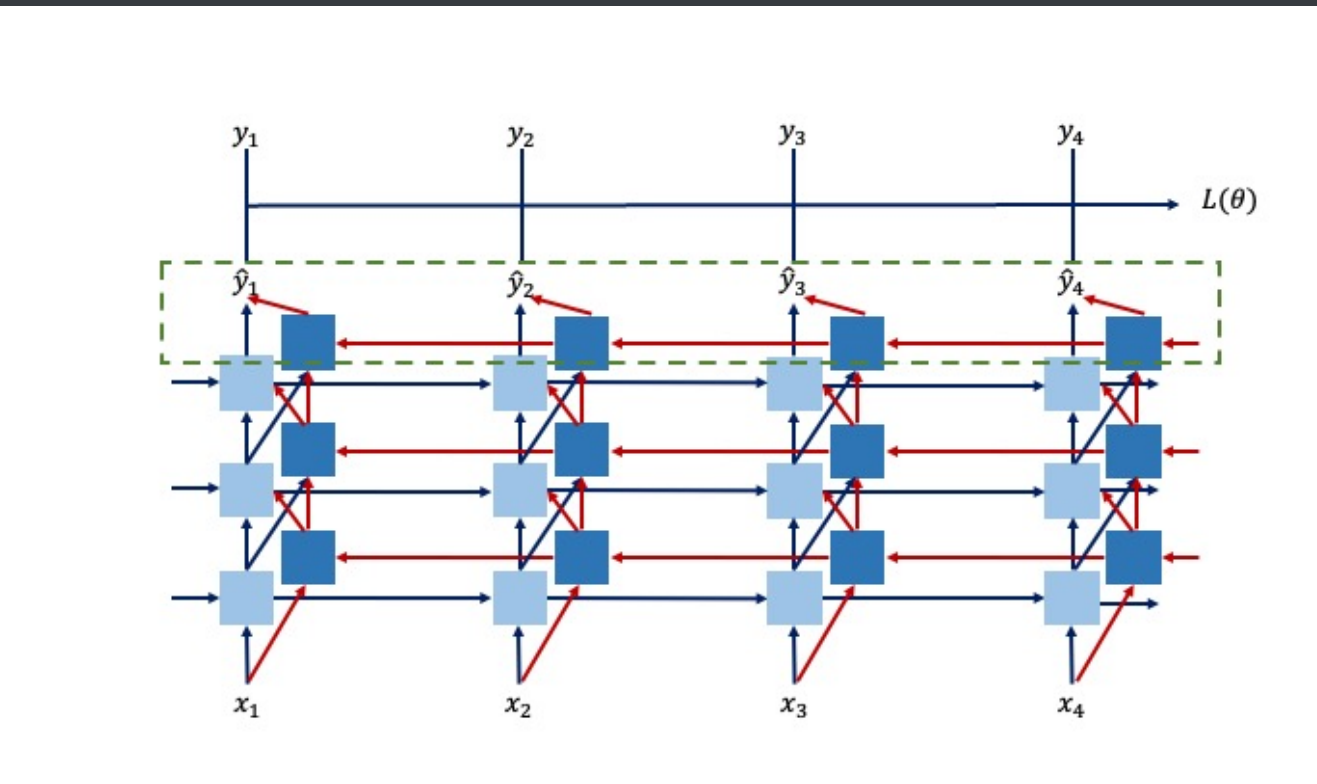


$|h_t|$=(#directions x #layers, batch size, hidden size)

기본적으로 Multi-layered RNN과 비슷하지만

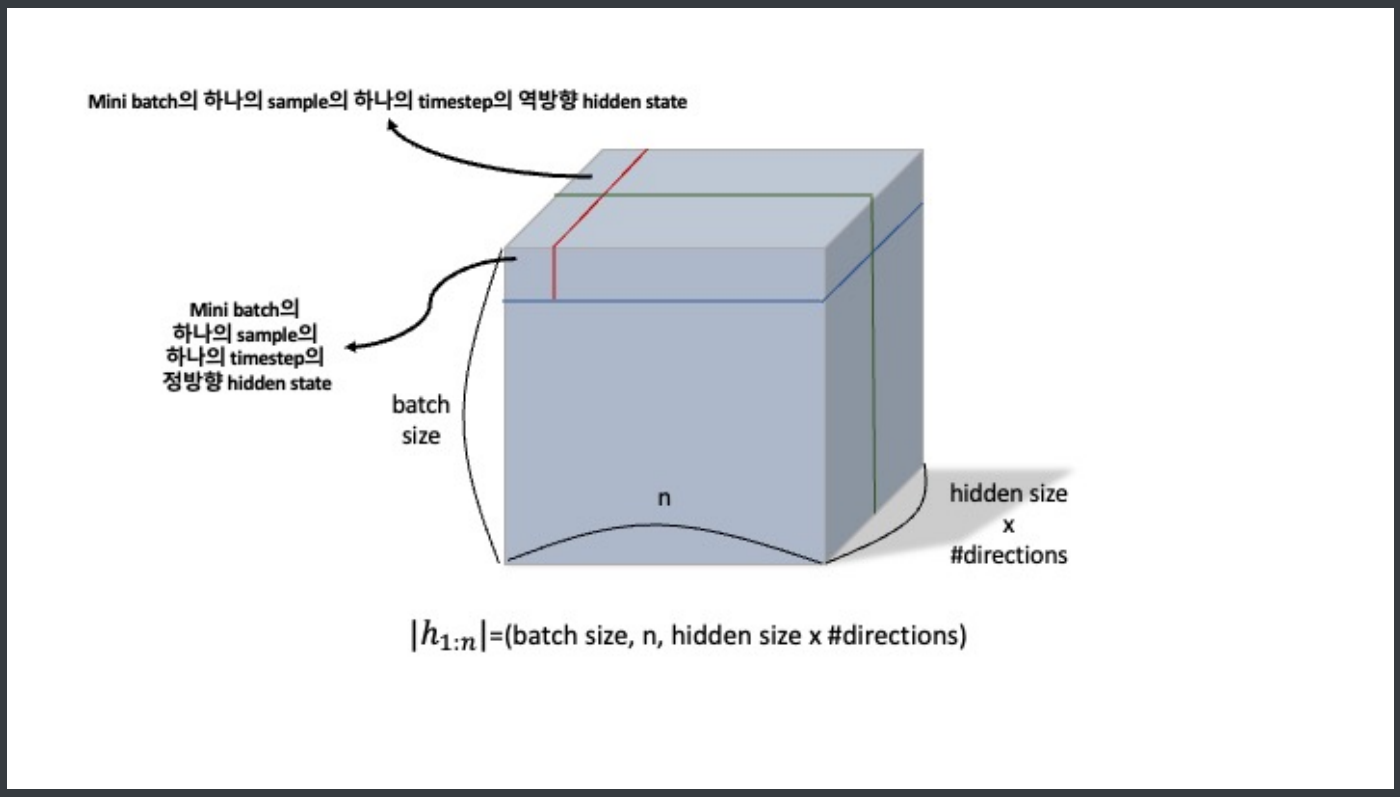
hidden state에서는 늘어난 방향으로 인해 Multi-layered RNN에 비해 layer의 개수가 2배가 된다 (정방향 + 역방향)

다음으로는 output을 살펴보겠다.



마찬가지로, 초록색 점선 박스가 **Bidirectional Multi-layered RNN***에서의 output이다.**

shape 모형은 다음과 같다.



output에서는 늘어난 방향으로 인해 Multi-layered RNN에 비해 hidden state의 개수가 2배가 된다 (정방향 + 역방향)

이러한 Bidirectional RNN은 Non-autoregressive한 task(현재 상태가 앞, 뒤 상태를 통해 정해지는 경우)에서는 좋은 성능을 이끌어낼 수 있지만, Ex) : text classification, POS tagging

Autoregressive(현재 상태가 과거 상태에 의존하여 정해지는 경우)한 task에서는 모든 timestep의 입력이 sequence 끝까지 한번에 주어지지 않기 때문에 사용할 수 없다. Ex) : NLG task