nn.lstm

```
def forward(self, input: Tensor, hx: Optional[Tensor] = None) -> Tuple[Tensor, Tensor]:
    is_packed = isinstance(input, PackedSequence)
    if is_packed:
        input, batch_sizes, sorted_indices, unsorted_indices = input
        max_batch_size = batch_sizes[0]
        max_batch_size = int(max_batch_size)
    else:
        batch_sizes = None
        max_batch_size = input.size(0) if self.batch_first else input.size(1)
        sorted_indices = None
        unsorted_indices = None

    if hx is None:
        num_directions = 2 if self.bidirectional else 1
        hx = torch.zeros(self.num_layers * num_directions,
                         max_batch_size, self.hidden_size,
                         dtype=input.dtype, device=input.device)
    else:
        # Each batch of the hidden state should match the input sequence that
        # the user believes he/she is passing in.
        hx = self.permute_hidden(hx, sorted_indices)

    self.check_forward_args(input, hx, batch_sizes)
    _impl = _rnn_impls[self.mode]
    if batch_sizes is None:
        result = _impl(input, hx, self._flat_weights, self.bias, self.num_layers,
                       self.dropout, self.training, self.bidirectional, self.batch_first)
    else:
        result = _impl(input, batch_sizes, hx, self._flat_weights, self.bias,
                       self.num_layers, self.dropout, self.training, self.bidirectional)
    output = result[0]
    hidden = result[1]

    if is_packed:
        output = PackedSequence(output, batch_sizes, sorted_indices, unsorted_indices)
    return output, self.permute_hidden(hidden, unsorted_indices)
```

nn.lstm

```
1 def forward(self, input: Tensor, hx: Optional[Tensor] = None) -> Tuple[Tensor, Tensor]:
2     is_packed = isinstance(input, PackedSequence)
```

1)x_torch만을 parameter로 넣어줬으므로
Forward에서 input만 받고 hx는 받지 않은 것

2)Input으로 들어온 x_torch가 PackedSequence인지 확인

```
from torch.nn.utils.rnn import PackedSequence
isinstance(x_torch, PackedSequence)
```

```
False
```

x_torch (2,20,28)의 2가 batch_size를 의미해서
얘도 mini batch 데이터기는 하지만
PackedSequence class 형식으로 묶인애는 아니라서

Isinstance(a, b)
: a의 type혹은 class가 b인지 확인
Isinstance(1, int)
: 1이 int형인지 확인 -> True
Isinstance(simclass, CSimple)
: simclass가 Csimple 클래스인지 확인

CLASS torch.nn.utils.rnn.PackedSequence

Holds the data and list of `batch_sizes` of a packed sequence.

All RNN modules accept packed sequences as inputs.

nn.lstm

```
if is_packed:
    input, batch_sizes, sorted_indices, unsorted_indices = input
    max_batch_size = batch_sizes[0]
    max_batch_size = int(max_batch_size)
else:
    batch_sizes = None
    max_batch_size = input.size(0) if self.batch_first else input.size(1)
    sorted_indices = None
    unsorted_indices = None
```

batch_sizes = None
max_batch_size = 2
sotred_indices = None
unstorted_indices = None

nn.lstm

```
    if hx is None:
        num_directions = 2 if self.bidirectional else 1
        hx = torch.zeros(self.num_layers * num_directions,
                         max_batch_size, self.hidden_size,
                         dtype=input.dtype, device=input.device)
    else:
        # Each batch of the hidden state should match the input sequence that
        # the user believes he/she is passing in.
        hx = self.permute_hidden(hx, sorted_indices)
```

지금은 hx가 None이지만 `rnn_out,(hn,cn) = R.rnn(x_torch)`

`class RecurrentNeuralNetworkClass`

의 forward에서는 직접 h0, c0만들고서 hx로 넣어줌

```
def forward(self,x):
    # Set initial hidden and cell states
    h0 = torch.zeros(self.n_layer, x.size(0), self.hdim
        # FILL IN HERE
    ).to(device)
    c0 = torch.zeros(self.n_layer, x.size(0), self.hdim
        # FILL IN HERE
    ).to(device)
    # RNN
    rnn_out,(hn,cn) = self.rnn(x, (h0,c0))
```

nn.lstm

```
    if hx is None:
        num_directions = 2 if self.bidirectional else 1
        hx = torch.zeros(self.num_layers * num_directions,
                         max_batch_size, self.hidden_size,
                         dtype=input.dtype, device=input.device)
    else:
        # Each batch of the hidden state should match the input sequence that
        # the user believes he/she is passing in.
        hx = self.permute_hidden(hx, sorted_indices)
```

hx가 None이면
cell state와 hidden state를 담을 tensor를 만들어주는 것

self.bidirectional은 옵션에 따라서 cell state와 hidden state를 동일하게 보는 애도 있어서 그런 듯

nn.lstm

```
if hx is None:
    num_directions = 2 if self.bidirectional else 1
    hx = torch.zeros(self.num_layers * num_directions,
                     max_batch_size, self.hidden_size,
                     dtype=input.dtype, device=input.device)
else:
    # Each batch of the hidden state should match the input sequence that
    # the user believes he/she is passing in.
    hx = self.permute_hidden(hx, sorted_indices)
```

hx가 None이면
cell state와 hidden state를 담을 tensor를 만들어주는 것

self.bidirectional은 옵션에 따라서 cell state와 hidden state를 동일하게 보는 애도 있어서 그런 듯

우리가 직접 h0, c0 만들 때는 두 개를 따로 만들어서
첫번째 parameter에 n_layer만 들어감
두번째 parameter는 batch_size로 동일
세번째 parameter도 hidden_dimension으로 동일

```
h0 = torch.zeros(self.n_layer, x.size(0), self.hdim
).to(device)
```

X_torch : (2,20,28)

nn.lstm

```
self.rnn = nn.LSTM(
    input_size=self.xdim,hidden_size=self.hdim,num_layers=self.n_layer,batch_first=True)
```

```
    self.check_forward_args(input, hx, batch_sizes)
    _impl = _rnn_impls[self.mode]
```

```
def check_forward_args(self, input: Tensor, hidden: Tensor, batch_sizes: Optional[Tensor]):
    self.check_input(input, batch_sizes)
    expected_hidden_size = self.get_expected_hidden_size(input, batch_sizes)

    self.check_hidden_size(hidden, expected_hidden_size)
```

```
def check_input(self, input: Tensor, batch_sizes: Optional[Tensor]) -> None:
    expected_input_dim = 2 if batch_sizes is not None else 3
    if input.dim() != expected_input_dim:
        raise RuntimeError(
            'input must have {} dimensions, got {}'.format(
                expected_input_dim, input.dim()))
    if self.input_size != input.size(-1):
        raise RuntimeError(
            'input.size(-1) must be equal to input_size. Expected {}, got {}'.format(
                self.input_size, input.size(-1)))
```

Input : x_torch.dim()=3
에러 발생하지 않음

Self.input_size는
nn.Lstm 선언할 때 넣어준
x_dim과
x_torch[-1]의 dim이 같으면
에러 발생하지 않음

nn.lstm

```
    self.check_forward_args(input, hx, batch_sizes)
    _impl = _rnn_impls[self.mode]
```

```
def check_forward_args(self, input: Tensor, hidden: Tensor, batch_sizes: Optional[Tensor]):
    self.check_input(input, batch_sizes)
    expected_hidden_size = self.get_expected_hidden_size(input, batch_sizes)

    self.check_hidden_size(hidden, expected_hidden_size)
```

```
    def get_expected_hidden_size(self, input: Tensor, batch_sizes: Optional[Tensor]) ->
Tuple[int, int, int]:
        if batch_sizes is not None:
            mini_batch = batch_sizes[0]
            mini_batch = int(mini_batch)
        else:
            mini_batch = input.size(0) if self.batch_first else input.size(1)
        num_directions = 2 if self.bidirectional else 1
        expected_hidden_size = (self.num_layers * num_directions,
                                mini_batch, self.hidden_size)
        return expected_hidden_size
```

mini_batch = 2
Hidden, cell state tensor합쳐서 만든 size 계산
(2*2, 2, 256) return

nn.lstm

```
    self.check_forward_args(input, hx, batch_sizes)
    _impl = _rnn_impls[self.mode]
```

tensor들의 shape을 확인해서 예외처리 해주는 줄

```
def check_forward_args(self, input: Tensor, hidden: Tensor, batch_sizes: Optional[Tensor]):
    self.check_input(input, batch_sizes)
    expected_hidden_size = self.get_expected_hidden_size(input, batch_sizes)

    self.check_hidden_size(hidden, expected_hidden_size)
```

```
def check_hidden_size(self, hx: Tensor, expected_hidden_size: Tuple[int, int, int],
                      msg: str = 'Expected hidden size {}, got {}') -> None:
    if hx.size() != expected_hidden_size:
        raise RuntimeError(msg.format(expected_hidden_size, list(hx.size())))
```

이전에 만든 hx가 expected_hidden_size와 같아야한다.

nn.lstm

```
self.check_forward_args(input, hx, batch_sizes)
_impl = _rnn_impls[self.mode]
```

```
_rnn_impls = {
    'RNN_TANH': _VF.rnn_tanh,
    'RNN_RELU': _VF.rnn_relu,
}
```

얘네가 진짜 RNN을 동작하는 애들
그런데 source code를 볼 수가 없음...ㅇㅅㅇ....

nn.lstm

```
if batch_sizes is None:
    result = _impl(input, hx, self._flat_weights, self.bias, self.num_layers,
                   self.dropout, self.training, self.bidirectional, self.batch_first)
else:
    result = _impl(input, batch_sizes, hx, self._flat_weights, self.bias,
                   self.num_layers, self.dropout, self.training, self.bidirectional)
output = result[0]
hidden = result[1]

if is_packed:
    output = PackedSequence(output, batch_sizes, sorted_indices, unsorted_indices)
return output, self.permute_hidden(hidden, unsorted_indices)
```

아마 _impl이 RNN의 구현을 해놓은 것이라
Input으로 (batch size를 잠깐 빼고 생각했을 때) (20,28)을 넣으면
(1,28)즉 단어를 1~20까지 순서대로 입력으로 넣으며 계산을 해서 결과를 내주는 것으로 보임

nn.lstm

```
    output = result[0]
    hidden = result[1]

    if is_packed:
        output = PackedSequence(output, batch_sizes, sorted_indices, unsorted_indices)
    return output, self.permute_hidden(hidden, unsorted_indices)
```

이렇게 나온 결과 result
Result[0]에는 우리가 원하는 output인 (2,20,256)

왜 이런 shape가 나왔는가?
단어 하나(1,28)이 lstm에 입력으로 들어가면 나오는 output은 (1,256)
nn.Lstm을 선언할 때 x_dim을 28, h_dim을 256으로 했으므로

그런데 한 개의 row에 각 단어의 input이 들어간 (20,28)를 입력으로 넣어주면
Lstm을 20번 돌면서 (1,256)인 결과가 20개가 나오고 이를 row 방향으로 합쳐서
결국 row에 각 단어의 output이 들어간 (20,256)이 나온다.

nn.lstm

```
    output = result[0]
    hidden = result[1]

    if is_packed:
        output = PackedSequence(output, batch_sizes, sorted_indices, unsorted_indices)
    return output, self.permute_hidden(hidden, unsorted_indices)
```

Batch_size를 고려해보면
Batch_size가 2이라는 것은 결국
서로 전혀 영향을 주지 않는 sequence 데이터 2개가 matrix로 묶여 있을 뿐
즉 여기서는 2개의 문장이 있다는 뜻

그래서 아마 예상컨대 lstm은 단어 하나가 순서대로 들어가는 구조지만
각 문장들끼리는 전혀 영향이 없으므로 병렬로 2개의 단어를 한번에 처리하는 것 같다.

즉, batch size를 고려했을 때
2개 문장의 단어 하나(2,1,28)가 lstm에 입력으로 들어가면 나오는 output은 (2,1,256)

nn.lstm

```
    output = result[0]
    hidden = result[1]

    if is_packed:
        output = PackedSequence(output, batch_sizes, sorted_indices, unsorted_indices)
    return output, self.permute_hidden(hidden, unsorted_indices)
```

이렇게 나온 결과 result
Result[1]에는 hn, cn이 들어있음 (2,2,256)

왜 이런 shape가 나왔는가?
일단 hn, cn은 각 단어마다의 결과가 나올 필요가 없음
그러니 단어의 개수인 x_torch.size(1)은 정말 무쓸모

nn.lstm

```
    output = result[0]
    hidden = result[1]

    if is_packed:
        output = PackedSequence(output, batch_sizes, sorted_indices, unsorted_indices)
    return output, self.permute_hidden(hidden, unsorted_indices)
```

Hn, cn는 lstm layer마다 있을거고 한 개의 sequence 데이터마다 있을거다
(한 개의 sequence 데이터가 batch 하나?라고 보면 되겟죠?)

일단 layer 마다 있는건 이해가 되는데
왜 sequence date 마다?
Cell state와 hidden state는 현재 문장의 정보를 갖고 있는거지
다른 문장의 정보와는 관련이 없음
그래서 다른 문장과 cell state, hidden state를 섞어서 쓰면 안되니
한 개의 sequence 데이터마다 만들어준다.

nn.lstm

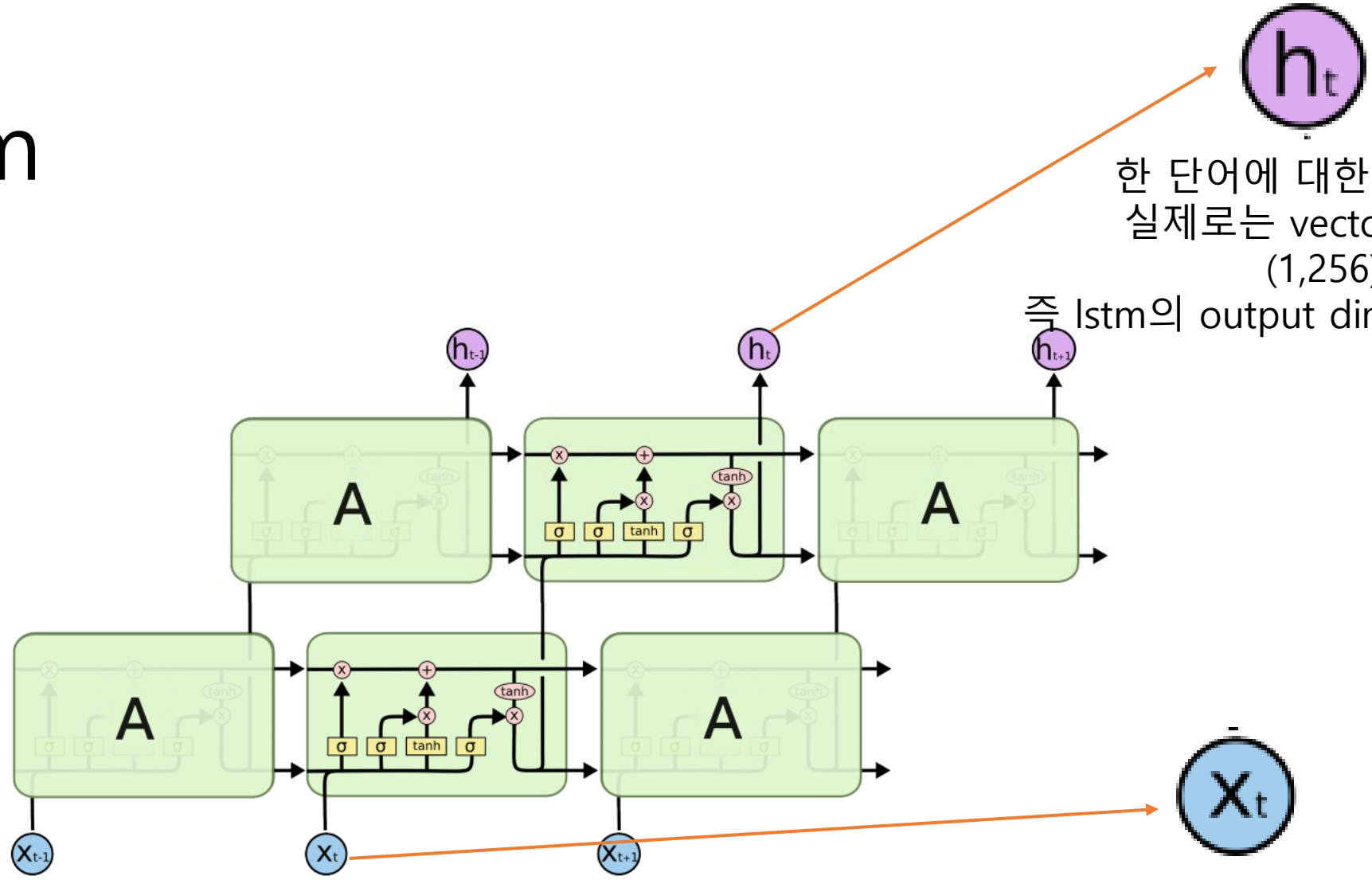
```
    output = result[0]
    hidden = result[1]

    if is_packed:
        output = PackedSequence(output, batch_sizes, sorted_indices, unsorted_indices)
    return output, self.permute_hidden(hidden, unsorted_indices)
```

결론 hn, cn 의 shape (2,2,256)

첫번째 2는 lstm layer의 개수
두번째 2는 batch_size 즉 sequence data의 개수
세번째 256은 그냥 output dimension이랑 동일 state가 담고 있는 값이 256개라는 것

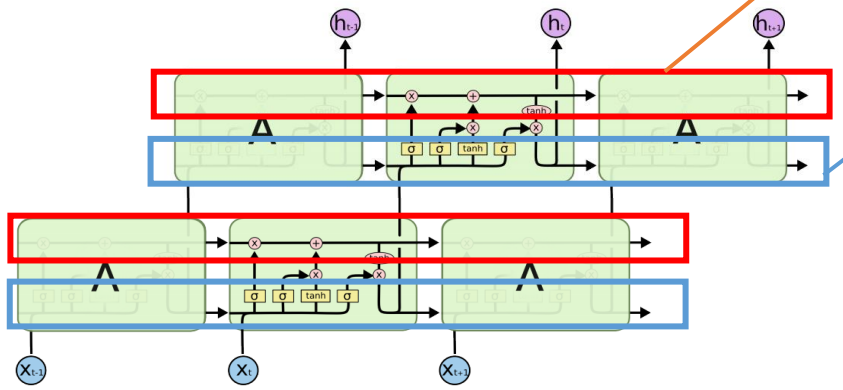
nn.lstm


h_t
한 단어에 대한 lstm결과
실제로는 vector로 존재
(1,256)
즉 lstm의 output dimension = 256
h_{t-1}
h_t
h_{t+1}
A
A
A
A
tanh
σ
σ
tanh
σ
X_{t-1}
X_t
X_{t+1}
X_t
한 개의 단어
실제로는 vector로 존재
(1,28)
즉 lstm의 input dimension = 28
Layer의 개수
n_layer =2
단어의 개수
x_torch.size(1)=20

nn.lstm

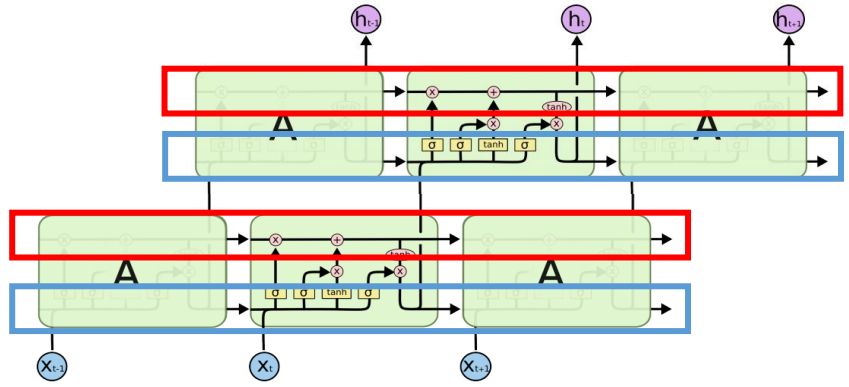
Cell state 개수 : 2*2 = 4

Hidden state 개수 : 2*2 =4



Layer의 개수
n_layer =2

단어의 개수
x_torch.size(1)=20



Layer의 개수
n_layer =2

단어의 개수
x_torch.size(1)=20

Batch size
X_torch.size(0)=2
Sequence data별로 lstm을 돌린다.

nn.lstm

그래서 하고 싶은 말

실제로 lstm은 단어 하나씩을 순차적으로 입력 받아 진행되지만
코드 구현에서는 matrix로 만들어서 한번에 넣으면
Batch는 병렬적으로, sequence 데이터는 순차적으로 단어를 넣으며 진행된 결과를 준다.