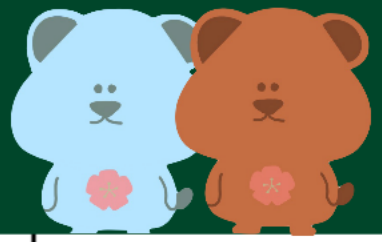
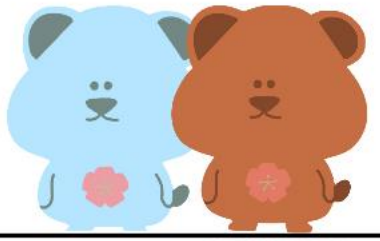


11주차 발표

손소현 오수진 최하경

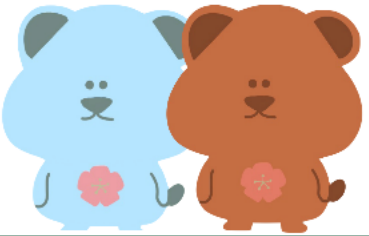
EWHA EURON

목차



EWHA EURON

#05 LSTM



EWHA EURON

#05 LSTM

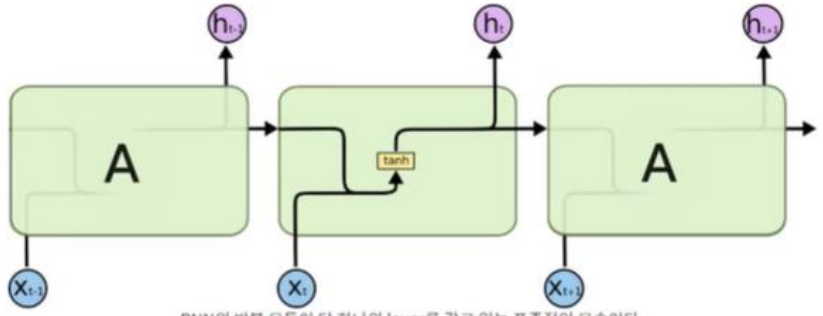
🚀 RNN vs LSTM

- ✓ RNN의 단점인, 가중치가 업데이트되는 과정에서 기울기가 1보다 작은 값이 계속 곱해지기 때문에, 기울기 소멸문제(Gradient Vanishing)를 극복하기 위해 고안
- ✓ Long Short Term Memory라는 이름에서 확인할 수 있듯, 장기 메모리와 단기메모리에 들어갈 정보를 나눠 학습

• RNN vs LSTM 구조 비교

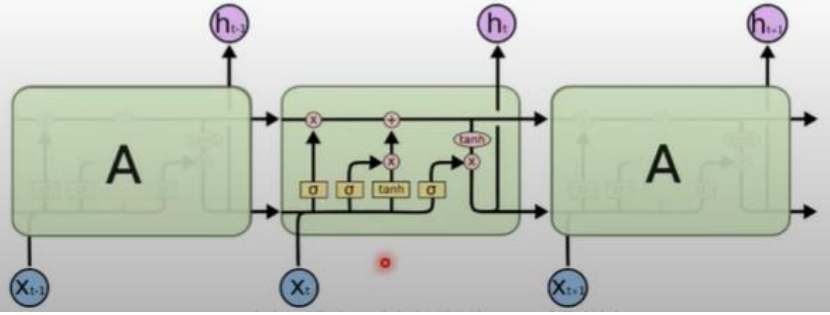
출처 : https://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN



RNN의 반복 모듈이 단 하나의 layer를 갖고 있는 표준적인 모습이다.

LSTM

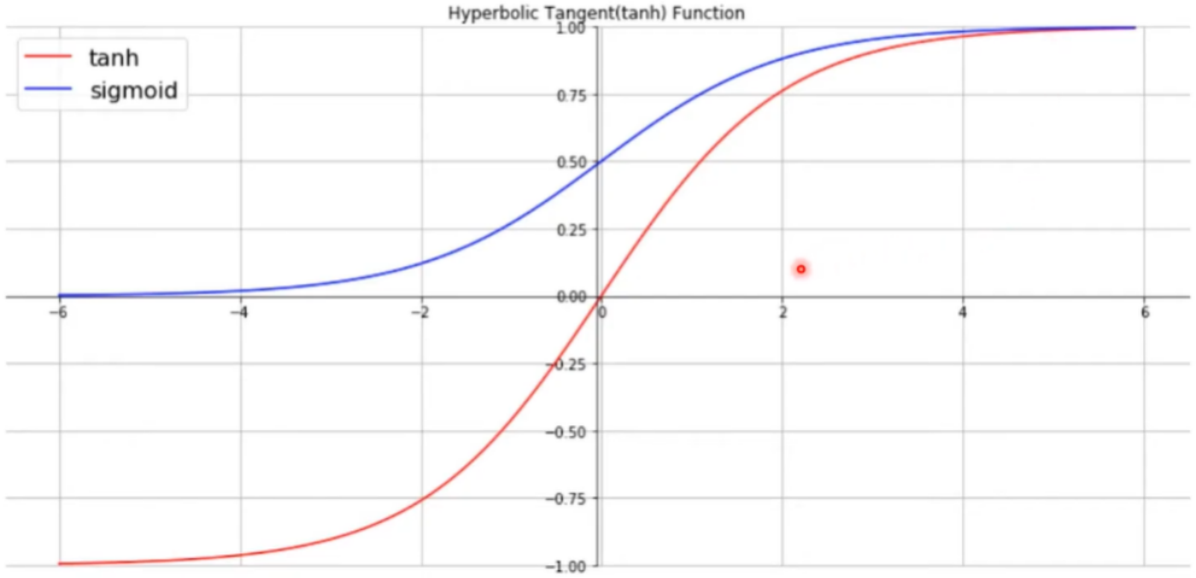


#05 LSTM

🚀 sigmoid vs tanh

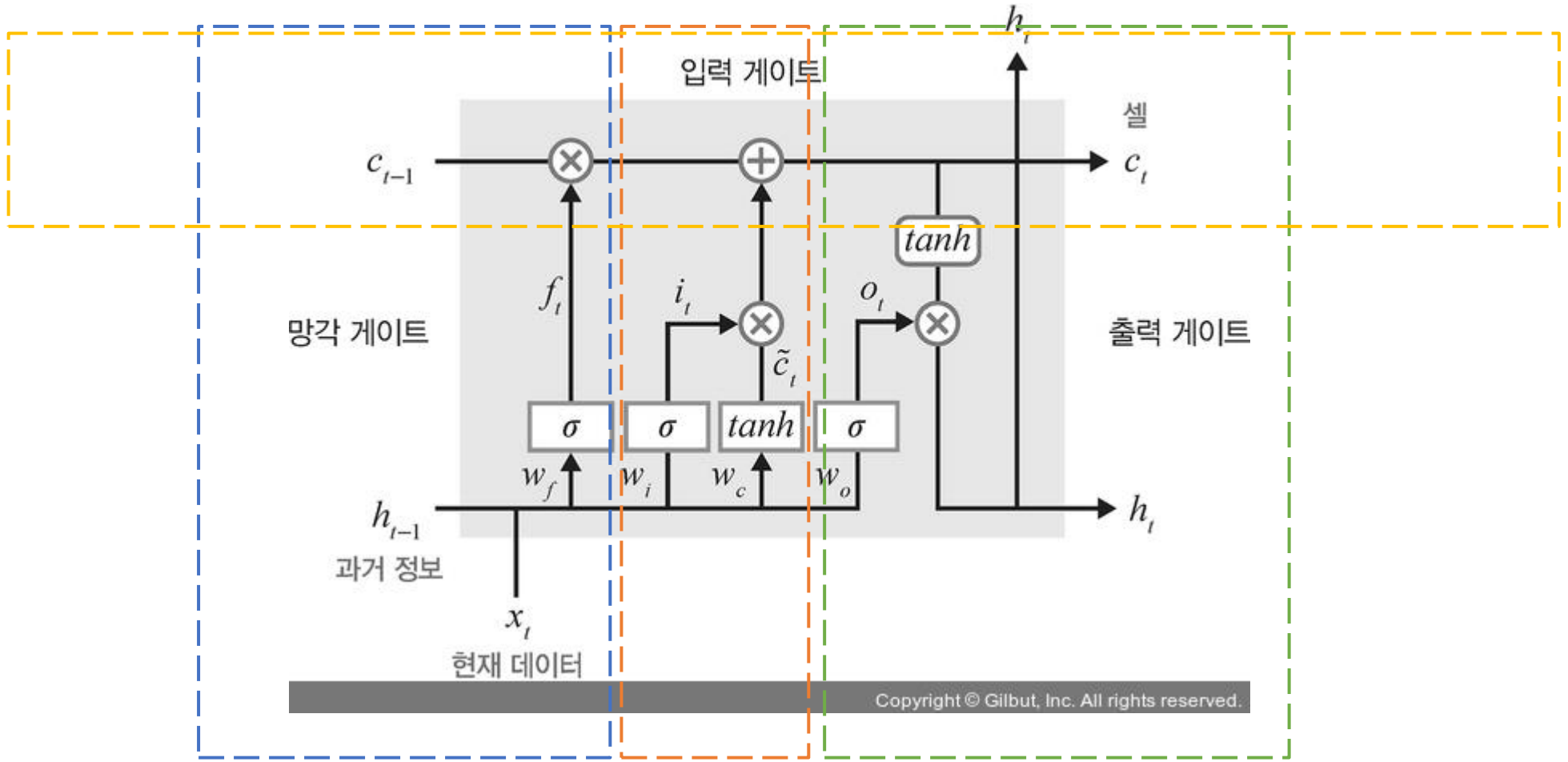
- ✓ 둘 다 비선형이고, 수렴하는 것도 비슷하다.
- ✓ Sigmoid : 0 ~ 1 출력. 스위치 역할. 과거나 현재의 정보를 '얼마나' 저장할지. 삭제 과정을 거친 **정보의 양**
- ✓ Tanh : -1 ~ +1. 현재 정보를 '얼마나' 더할지, 최종 cell state 값을 '얼마나' 빼낼지 결정

  - Hyperbolic Tangent(tanh) 함수를 사용하는 이유



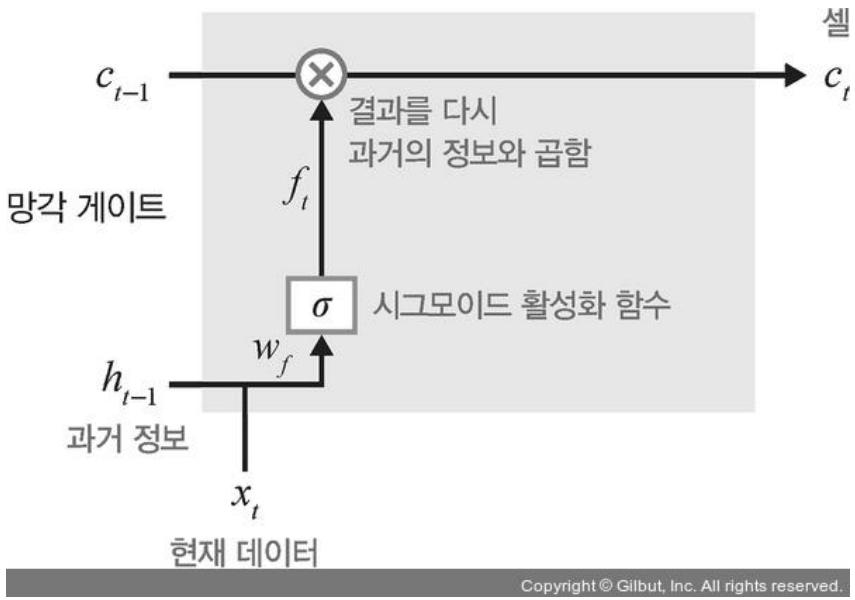
#05 LSTM

전체적인 게이트 구조



#05 LSTM

순전파 - 망각게이트



Copyright © Gilbut, Inc. All rights reserved.

- ✓ 과거의 정보를 어느정도 기억할지 결정
- ✓ 과거정보와 현재 데이터에 시그모이드를 취한 후, 과거 정보에 곱해줌
- ✓ 즉 0이면 과거정보 버리고, 1이면 과거정보 유지

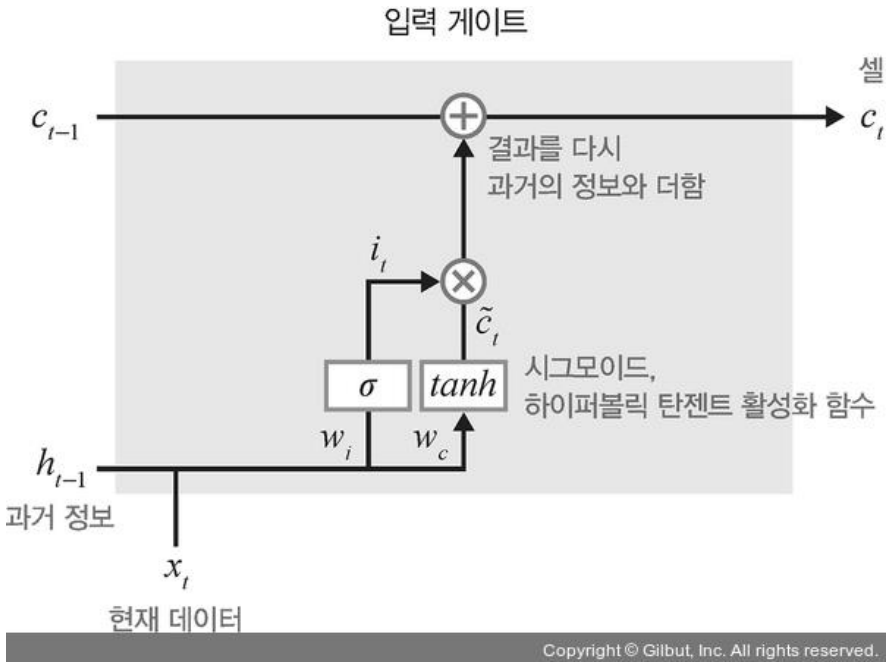
$$f_t = \sigma(w_f[h_{t-1}, x_t])$$

$$c_t = f_t \cdot c_{t-1}$$

Copyright © Gilbut, Inc. All rights reserved.

#05 LSTM

순전파 - **입력게이트**



Copyright © Gilbut, Inc. All rights reserved.

$$i_t = \sigma(w_i[h_{t-1}, x_t])$$

$$\tilde{c}_t = \tanh(w_c[h_{t-1}, x_t])$$

$$c_t = c_{t-1} + i_t \cdot \tilde{c}_t$$

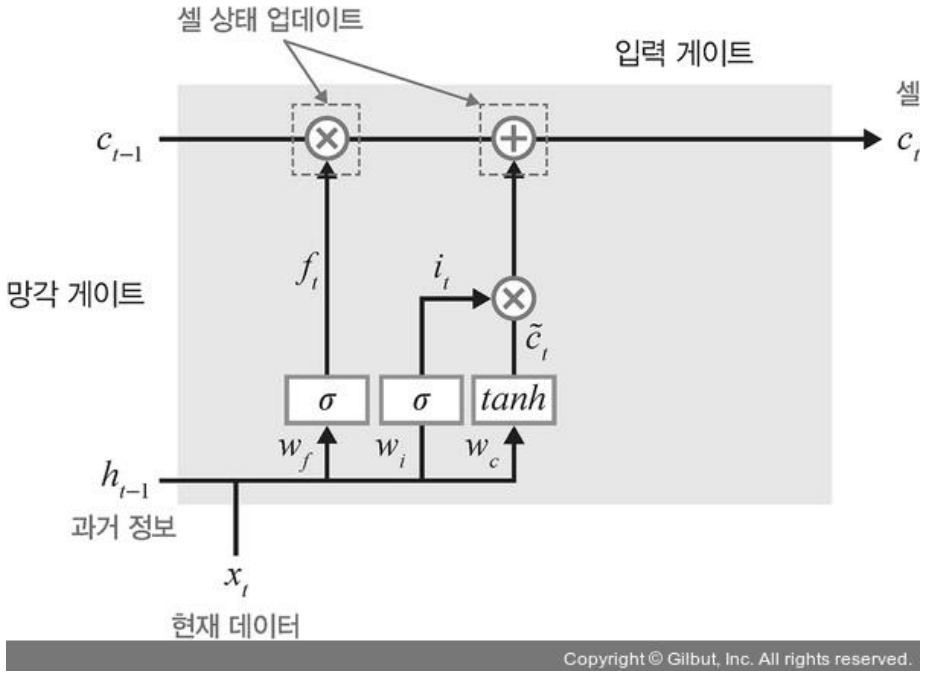
Copyright © Gilbut, Inc. All rights reserved.

- ✓ 현재의 정보를 어느정도 기억할지 결정
- ✓ 시그모이드 : 1이면 현재정보 허용, 0이면 차단
- ✓ 하이퍼볼릭탄젠트 : 현재정보를 '얼마나' 받아들일지 결정

EWHA EURON

#05 LSTM

순전파 – 셀 상태 업데이트



Copyright © Gilbut, Inc. All rights reserved.

$$f_t = \sigma(w_f[h_{t-1}, x_t])$$

$$c_t = c_{t-1} + i_t \cdot \tilde{c}_t$$

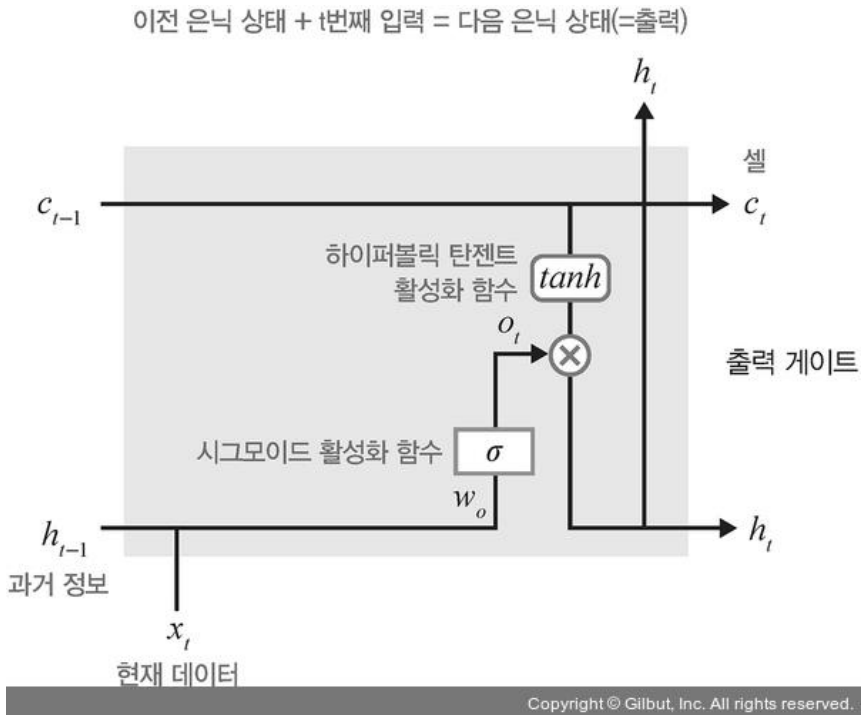
Copyright © Gilbut, Inc. All rights reserved.

✓ 앞에서 계산한 결과를 셀에 업데이트

✓ 첫번째 식은 망각게이트 결과
✓ 두번째 식은 입력게이트 결과

#05 LSTM

순전파 - **출력게이트**



$$o_t = \sigma(w_o [h_{t-1}, x_t])$$

$$h_t = o_t \cdot \tanh(c_{t-1})$$

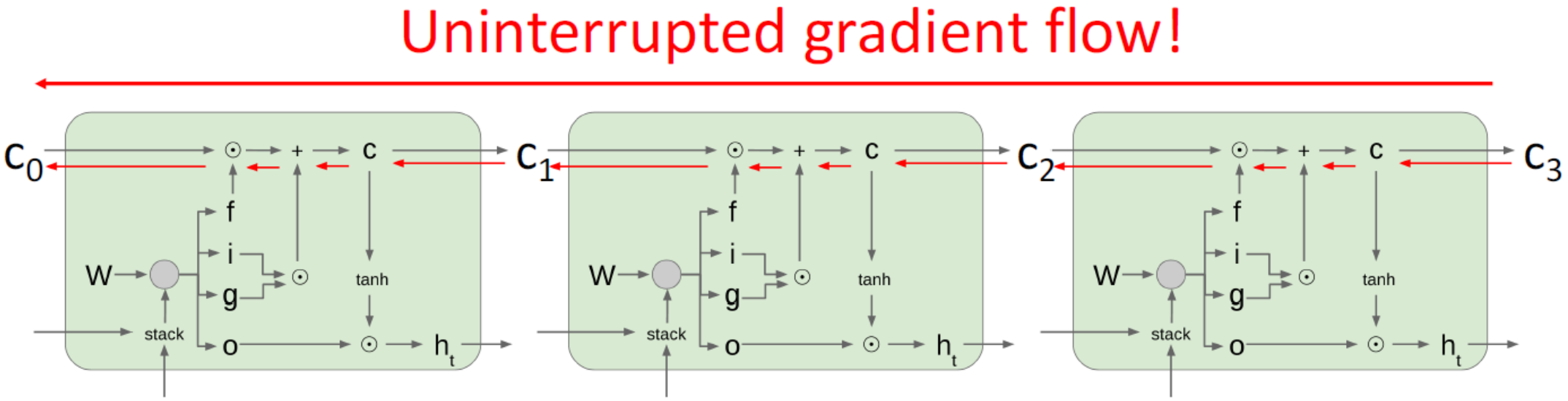
Copyright © Gilbut, Inc. All rights reserved.

- ✓ 현재의 정보를 어느정도 기억할지 결정
- ✓ 시그모이드 : 0이면 의미있는 결과로 출력, 1이면 출력 x
- ✓ 하이퍼볼릭탄젠트: t-1시점의 값을 '얼마나' 저장할지
- ✓ 이게 새로운 단기 메모리가 됨!

#05 LSTM

역전파

- ✓ 셀을 통해서 수행하기 때문에 중단 없는 기울기(uninterrupted gradient flow)
- ✓ 오차도 전파될 수 있음
- ✓ 심지어 오차가 입력으로 전파될 수 있음
- ✓ F값 (시그모이드 결과값)이 0에 가까워지면, C_t에서 C_t-1로 역전파시,
  정보의 손실을 넘어 정보의 파괴가 일어날 잠재적인 위험이 있다.



#05 LSTM

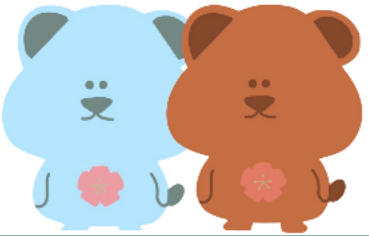
🚀 코드 맛보기

✓ 이렇게 셀게이트가 너무 복잡해서 고안된 것이 GRU

```
def forward(self, x, hidden):
    hx, cx = hidden
    x = x.view(-1, x.size(1))

    gates = self.x2h(x) + self.h2h(hx) ------ ① ″
    gates = gates.squeeze() ------ ③
    ingate, forgetgate, cellgate, outgate = gates.chunk(4, 1) ------ ① ‴

    ingate = F.sigmoid(ingate) ------ 입력 게이트에 시그모이드 활성화 함수 적용
    forgetgate = F.sigmoid(forgetgate) ------ 망각 게이트에 시그모이드 활성화 함수 적용
    cellgate = F.tanh(cellgate) ------ 셀 게이트에 탄젠트 활성화 함수 적용
    outgate = F.sigmoid(outgate) ------ 출력 게이트에 시그모이드 활성화 함수 적용

    cy = torch.mul(cx, forgetgate) + torch.mul(ingate, cellgate) ------ ④
    hy = torch.mul(outgate, F.tanh(cy)) ------ ④ ′
    return(hy, cy)
```

EWHA EURON

#06 GRU

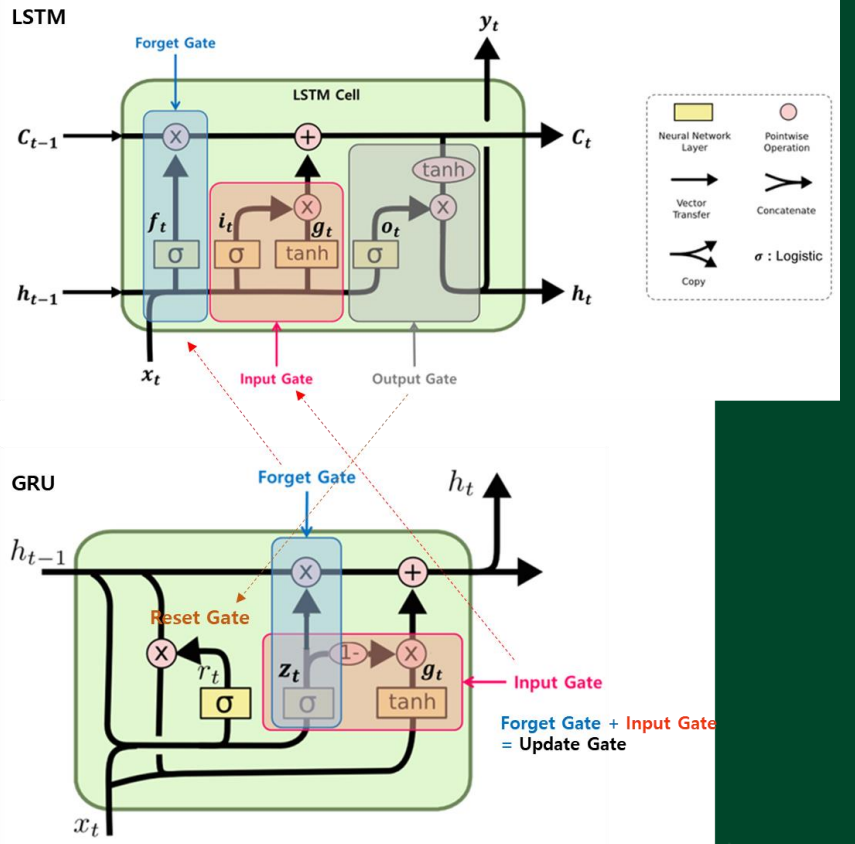


EWHA EURON

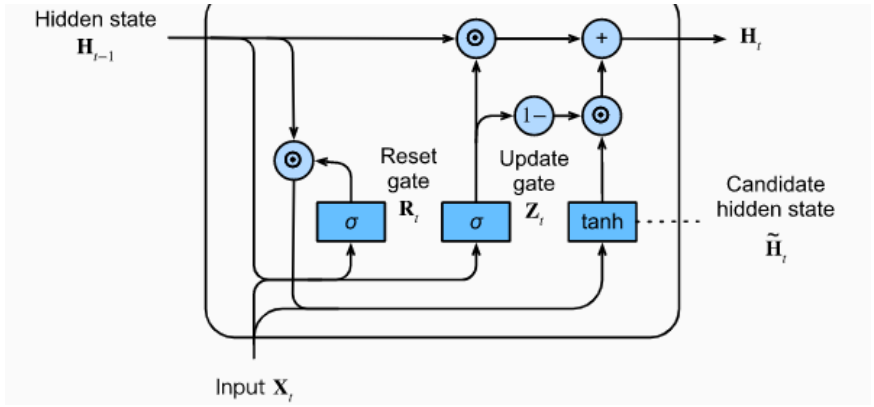
1. GRU(Gated Recurrent Unit)

GRU: 게이트 메커니즘이 적용된 RNN 프레임워크의 한 종류로, 기존 LSTM의 구조를 조금 더 간단하게 개선한 모델

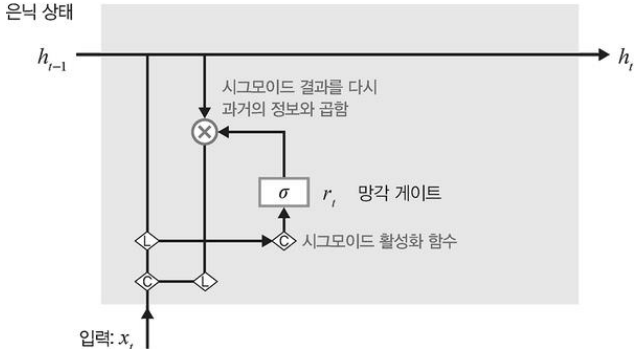
LSTM	GRU
Gate 3개: Forget gate, Input gate, Output gate	Gate 2개: 망각 게이트(Reset gate), Update gate
Cell state, hidden state 있음	cell state, hidden state가 합쳐져 하나의 hidden state로 표현 (즉, hidden state가 cell state의 역할까지 같이 함)
마지막 출력값에 활성화 함수 적용	마지막 출력값에 활성화함수를 적용하지 않음
	성능 면에서는 LSTM과 비교해서 우월하다고 할 수 없지만, 학습할 파라미터가 더 적은 것이 장점 (**데이터 양이 적을 때는, 매개 변수의 양이 적은 GRU가 조금 더 낫고, 데이터 양이 더 많으면 LSTM이 더 낫다고 알려져 있음)



2. GRU(Gated Recurrent Unit) 구조



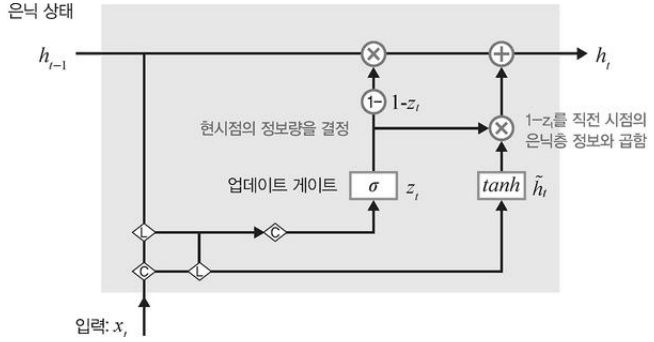
<망각 게이트>



- 이전 정보에서 얼마만큼을 선택해서 내보낼지 결정
- **과거 정보를 적당히 초기화**시키려는 목적으로,
  시그모이드 함수를 출력으로 이용하여 (0,1) 값을 이전 은닉층에 곱함
- 식: 이전 시점의 은닉층 값에 현시점의 정보에 대한 가중치를 곱한 것

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

<업데이트 게이트>



- **과거와 현재 정보의 최신화 비율 결정**
  (과거 정보와 현재 새로운 정보를 얼마만큼 가져갈지 정해줌)
- LSTM의 망각 게이트와 인풋 게이트의 역할 모두 담당
- Zt: 현재 정보의 비율 결정
- 1-Zt: 이전 정보의 비율 결정

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

<후보군>

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

- 현시점의 정보에 대한 후보군 계산
- 과거 은닉층의 정보를 그대로 이용하지 않고
  망각 게이트의 결과를 이용하여 후보군 계산
- Tanh layer를 통해 -1과 1 사이의 값으로 바꿔줌

<은닉층 계산>

$$h_t = (1 - z_t) * h_{t-1} + z_t \times \tilde{h}_t$$

- 마지막으로 업데이트 게이트 결과와 후보군 결과를 결합하여
  현시점의 은닉층 계산
- 시그모이드 함수의 결과(Zt)는 현시점에서 결과에 대한 정보량 결정,
  1-시그모이드 함수의 결과(1-Zt)는 과거의 정보량을 결정함

3. GRU 셀 구현

- MNIST 데이터셋 사용, 전처리 과정 생략(LSTM의 코드와 같음)

```
# GRU 셀에 대한 네트워크 (LSTM과 크게 다르지 않음)
class GRUCell(nn.Module):
    def __init__(self, input_size, hidden_size, bias=True):
        super(GRUCell, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.bias = bias
        self.x2h = nn.Linear(input_size, 3 * hidden_size, bias=bias)
        self.h2h = nn.Linear(hidden_size, 3 * hidden_size, bias=bias)
        self.reset_parameters()

    def reset_parameters(self):
        std = 1.0 / math.sqrt(self.hidden_size)
        for w in self.parameters():
            w.data.uniform_(-std, std)

    def forward(self, x, hidden):
        x = x.view(-1, x.size(1))

        gate_x = self.x2h(x)
        gate_h = self.h2h(hidden)

        gate_x = gate_x.squeeze()
        gate_h = gate_h.squeeze()

        i_r, i_i, i_n = gate_x.chunk(3, 1)
        h_r, h_i, h_n = gate_h.chunk(3, 1)

        resetgate = F.sigmoid(i_r + h_r)
        inputgate = F.sigmoid(i_i + h_i)
        newgate = F.tanh(i_n + (resetgate * h_n))

        hy = newgate + inputgate * (hidden - newgate)
        return hy
```

- LSTM 셀에서는 4를 곱했음
- GRU 셀에서는 3개의 게이트 사용 → 3 곱함
  - 망각, 입력 게이트 + newgate(탄젠트 활성화 함수 적용되는 부분)

파라미터 초기화

- LSTM 셀에서는 gates를 x2h+h2h로 정의
- GRU 셀에서는 개별적인 상태 유지함

총 3개의 게이트를 위해 세 개로 쪼개짐

'새로운 게이트 ' 는 탄젠트 활성화 함수가 적용된 게이트

3. GRU 셀 구현

```python
# GRU 전반적인 네트워크 구조
class GRUModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, layer_dim, output_dim, bias=True):
        super(GRUModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.layer_dim = layer_dim
        self.gru_cell = GRUCell(input_dim, hidden_dim, layer_dim) 앞에서 정의한 GRUCell 함수 불러옴
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        if torch.cuda.is_available():
            h0 = Variable(torch.zeros(self.layer_dim, x.size(0), self.hidden_dim).cuda())
        else:
            h0 = Variable(torch.zeros(self.layer_dim, x.size(0), self.hidden_dim))

        outs = []
        hn = h0[0,:,:]

        for seq in range(x.size(1)):
            hn = self.gru_cell(x[:,seq,:], hn)
            outs.append(hn)

        out = outs[-1].squeeze()
        out = self.fc(out)
        return out
```

```python
# 모델에 적용될 옵티마이저와 손실 함수 설정
input_dim = 28
hidden_dim = 128
layer_dim = 1
output_dim = 10

model = GRUModel(input_dim, hidden_dim, layer_dim, output_dim)

if torch.cuda.is_available():
    model.cuda()

criterion = nn.CrossEntropyLoss()
learning_rate = 0.1
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

```python
# 모델 학습, 검증 데이터셋으로 모델 성능 측정 => LSTM과 동일
seq_dim = 28
loss_list = []
iter = 0
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        if torch.cuda.is_available():
            images = Variable(images.view(-1, seq_dim, input_dim).cuda())
            labels = Variable(labels.cuda())
        else:
            images = Variable(images.view(-1, seq_dim, input_dim))
            labels = Variable(labels)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)

        if torch.cuda.is_available():
            loss.cuda()

        loss.backward()
        optimizer.step()

        loss_list.append(loss.item())
        iter += 1

        if iter % 500 == 0:
            correct = 0
            total = 0
            for images, labels in valid_loader:
                if torch.cuda.is_available():
                    images = Variable(images.view(-1, seq_dim, input_dim).cuda())
                else:
                    images = Variable(images.view(-1 , seq_dim, input_dim))

                outputs = model(images)
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)

                if torch.cuda.is_available():
                    correct += (predicted.cpu() == labels.cpu()).sum()
                else:
                    correct += (predicted == labels).sum()

            accuracy = 100 * correct / total
            print('Iteration: {}. Loss: {}. Accuracy: {}'.format(iter, loss.item(), accuracy))
```

3. GRU 셀 구현

```
# 테스트 데이터셋을 이용한 모델 예측
def evaluate(model, val_iter):
    corrects, total, total_loss = 0, 0, 0
    model.eval()
    for images, labels in val_iter:
        if torch.cuda.is_available():
            images = Variable(images.view(-1, seq_dim, input_dim).cuda())
        else:
            images = Variable(images.view(-1 , seq_dim, input_dim)).to(device)
        labels = labels.cuda()
        logit = model(images).cuda()
        loss = F.cross_entropy(logit, labels, reduction = "sum")
        _, predicted = torch.max(logit.data, 1)
        total += labels.size(0)
        total_loss += loss.item()
        corrects += (predicted == labels).sum()

    avg_loss = total_loss / len(val_iter.dataset)
    avg_accuracy = corrects / total
    return avg_loss, avg_accuracy
```

```
test_loss, test_acc = evaluate(model,test_loader)
print("Test Loss: %5.2f | Test Accuracy: %5.2f" % (test_loss, test_acc))
```

```
Test Loss:  0.07 | Test Accuracy:  0.98
```

- LSTM 셀을 사용했을 때와 성능 비슷함
- LSTM과 GRU 둘 중 어느 것이 더 좋다고 말할 수 없음
- 즉, 주어진 데이터셋을 다양한 모델에 적용하여 최적의 모델 찾는 것이 중요함

EWHA EURON

4. GRU 계층 구현

- 스타벅스 주가 데이터셋 사용
- 전처리
  - ‘Date’ 칼럼 인덱스 지정, ‘Volume’ 칼럼 실수로 변경
  - X, y 분리 및 정규화
  - LSTM 네트워크에 적용하기 위해 데이터셋 형태 변경

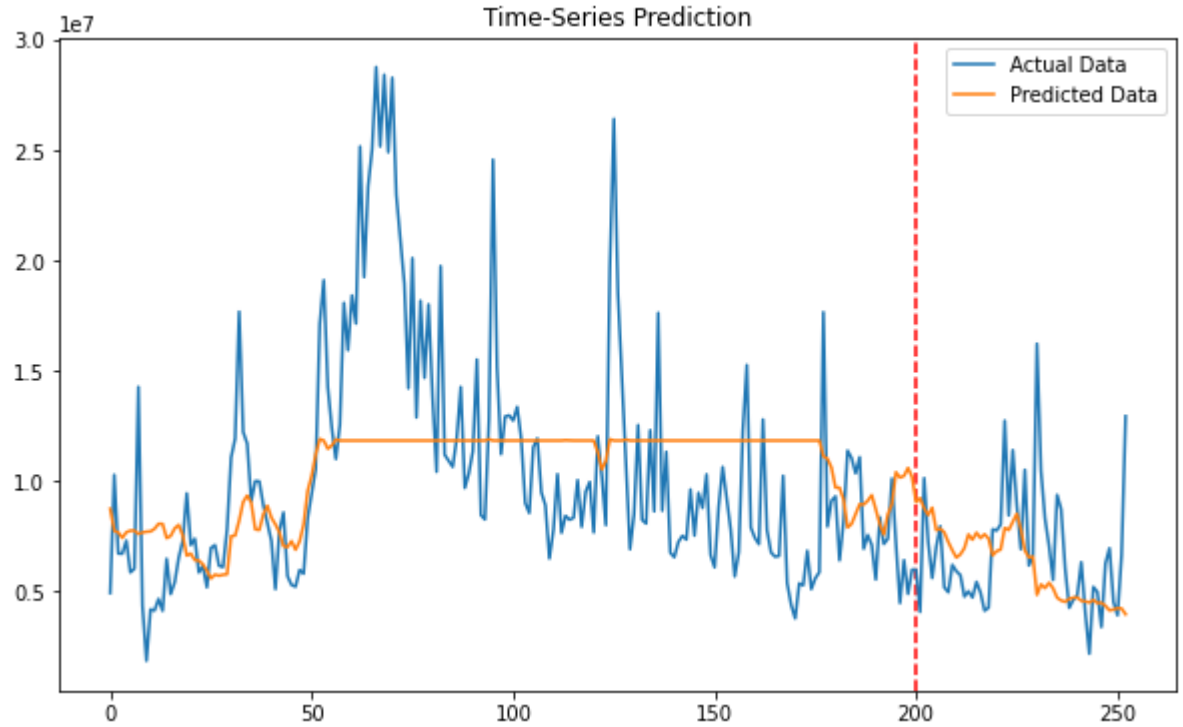
```
# GRU 모델의 네트워크
class GRU(nn.Module):
    def __init__(self, num_classes, input_size, hidden_size, num_layers, seq_length):
        super(GRU, self).__init__()
        self.num_classes = num_classes
        self.num_layers = num_layers
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.seq_length = seq_length

        self.gru = nn.GRU(input_size=input_size, hidden_size=hidden_size,
                          num_layers=num_layers, batch_first=True)
        self.fc_1 =  nn.Linear(hidden_size, 128)
        self.fc = nn.Linear(128, num_classes)
        self.relu = nn.ReLU()

    def forward(self,x):
        h_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size))
        output, (hn) = self.gru(x, (h_0))
        hn = hn.view(-1, self.hidden_size)
        out = self.relu(hn)
        out = self.fc_1(out)
        out = self.relu(out)
        out = self.fc(out)
        return out
```

- 은닉 상태에 대해 0으로 초기화하는 부분
- LSTM의 계층은 셀 상태가 있었지만 GRU는 셀 상태 정의하지 않음

4. GRU 계층 구현

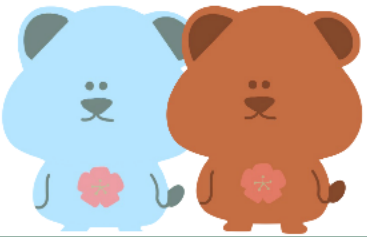
```python
# 모델 학습
for epoch in range(num_epochs):
    outputs = model.forward(X_train_tensors_f)
    optimizer.zero_grad()
    loss = criterion(outputs, y_train_tensors)
    loss.backward()

    optimizer.step()
    if epoch % 100 == 0:
        print("Epoch: %d, loss: %1.5f" % (epoch, loss.item()))
```

```
Epoch: 0, loss: 0.27521
Epoch: 100, loss: 0.11170
Epoch: 200, loss: 0.05401
Epoch: 300, loss: 0.04031
Epoch: 400, loss: 0.03741
Epoch: 500, loss: 0.03610
Epoch: 600, loss: 0.03494
Epoch: 700, loss: 0.03398
Epoch: 800, loss: 0.03340
Epoch: 900, loss: 0.03313
```



- 모델의 예측 결과와 레이블 비교하여 출력
- 그래프상으로는 GRU의 예측력이 LSTM보다 더 좋아보임
- 하지만 수치상으로는 모델의 예측 정확도 유사함

#07 양방향 RNN



EWHA EURON

#07 양방향 RNN

양방향 RNN

- 이전 시점의 데이터 뿐만 아니라, 이후 시점의 데이터도 함께 활용해 출력값 예측할 수 있다는 아이디어에서 기반

운동을 열심히 하는 것은 [ ]을 늘리는데 효과적이다.

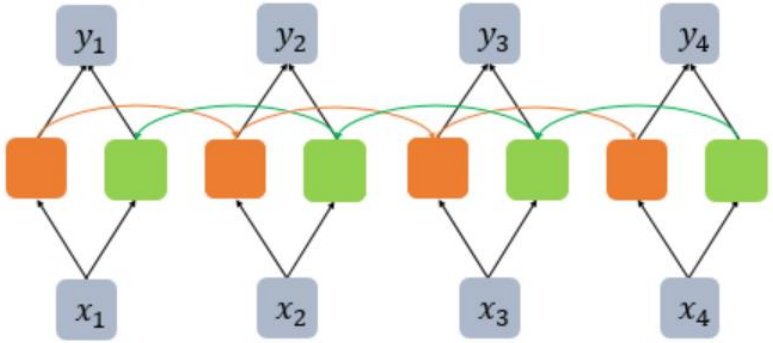
1) 근육
2) 지방
3) 스트레스

이전에 나온 단어들로만 빈칸을 채우는 것보다
뒤의 단어들까지 알고 있는 상태가 정답을 고르는데 도움이 됨

텍스트 데이터는 정방향 추론 못지 않게 역방향 추론도 유의미한 결과를 낼 수 있음

양방향 RNN 구조

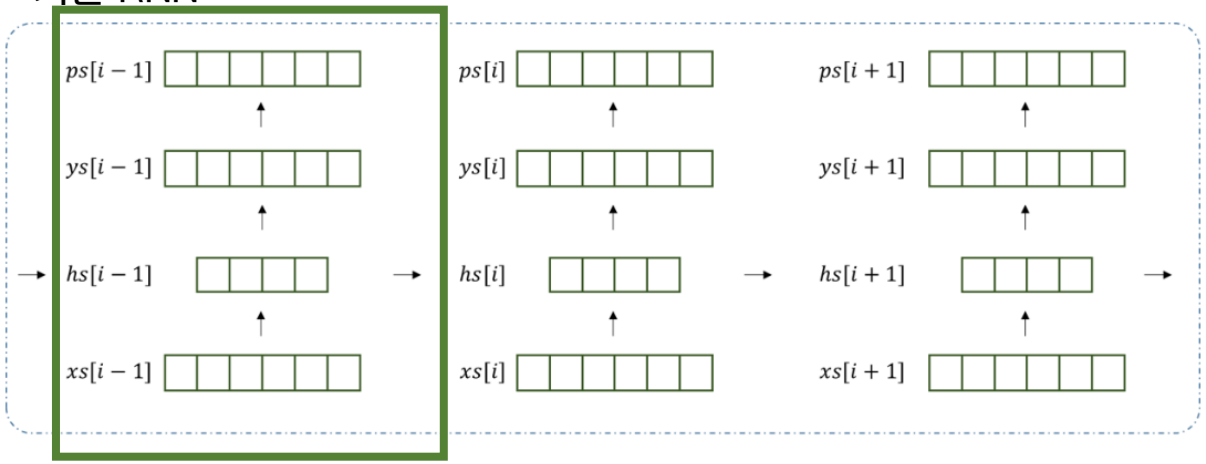
- 메모리 셀 2개를 이용해 하나의 출력값 예측
- 첫 번째 메모리 셀 : 이전 시점의 은닉 상태 이용 (주황색)
- 두 번째 메모리 셀 : 다음 시점의 은닉 상태 이용 (초록색)



#07 양방향 RNN

📌 기본 RNN VS 양방향 RNN

기본 RNN



$$ps[i] = softmax(ys[i])$$
$V \times 1 \quad V \times 1$

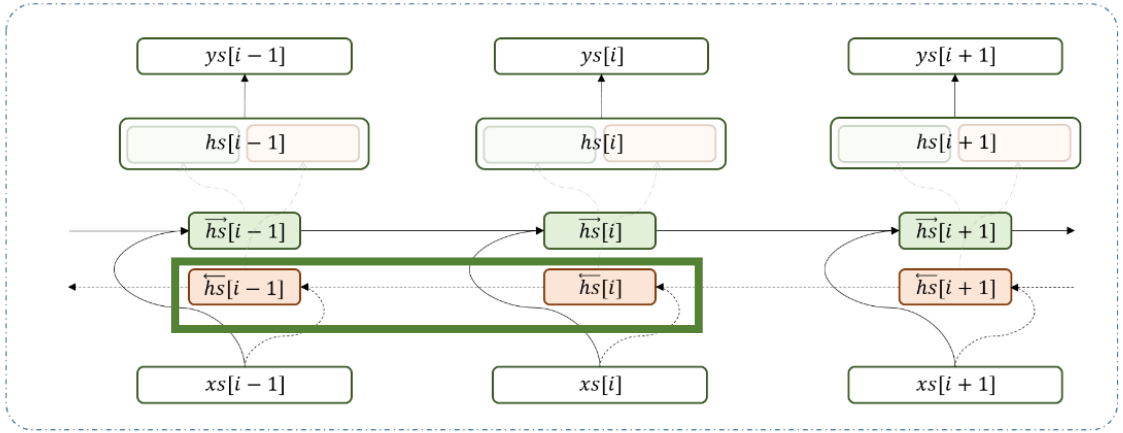
$$ys[i] = W_{hy}\, hs[i] + by$$
$V \times 1 \quad V \times H \quad H \times 1 \quad V \times 1$

$$hs[i] = \tanh(W_{xh}\, xs[i] + W_{hh}\, h[i-1] + bh)$$
$H \times 1 \quad H \times V \quad V \times 1 \quad H \times H \quad H \times 1 \quad H \times 1$

Hidden Size = H

Vocabulary size = V

양방향 RNN



$$ys[i] = W_{hy}\, hs[i] + by$$
$Y \times 1 \quad Y \times H \quad H \times 1 \quad Y \times 1$

$$\overleftarrow{hs}[i] = \tanh(\overleftarrow{W_{xh}}\, xs[i] + \overleftarrow{W_{hh}}\, \overleftarrow{hs}[i+1] + \overleftarrow{bh})$$
$\frac{H}{2} \times 1 \quad \frac{H}{2} \times V \quad V \times 1 \quad \frac{H}{2} \times \frac{H}{2} \quad \frac{H}{2} \times 1 \quad \frac{H}{2} \times 1$

$$\overrightarrow{hs}[i] = \tanh(\overrightarrow{W_{xh}}\, xs[i] + \overrightarrow{W_{hh}}\, \overrightarrow{hs}[i-1] + \overrightarrow{bh})$$
$\frac{H}{2} \times 1 \quad \frac{H}{2} \times V \quad V \times 1 \quad \frac{H}{2} \times \frac{H}{2} \quad \frac{H}{2} \times 1 \quad \frac{H}{2} \times 1$

stacked!

$$hs[i] = \begin{bmatrix} \overrightarrow{hs}[i] \\ \overleftarrow{hs}[i] \end{bmatrix}$$

Output Size = Y

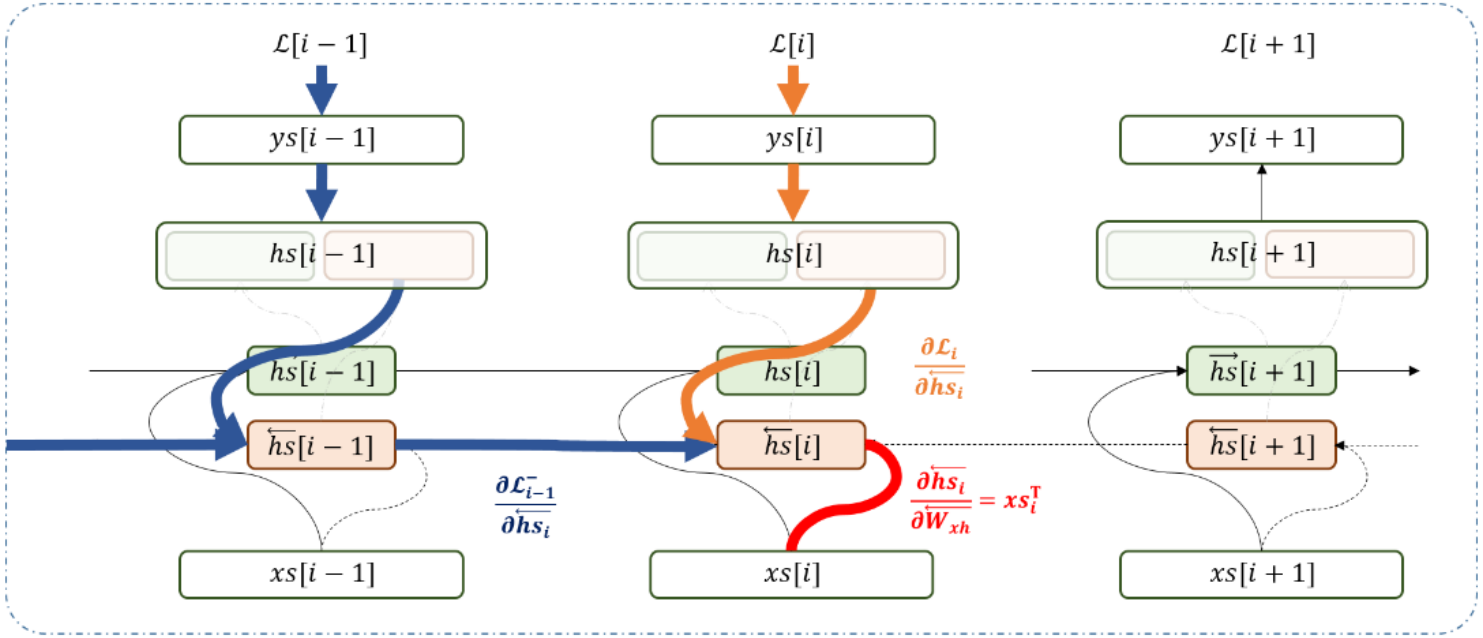
Hidden Size = H

Vocabulary size = V

정방향의 hidden layer와 역방향의 hidden layer을
concatenate해 완전한 [i]번째 hidden layer 만든다

#07 양방향 RNN

양방향 RNN의 backpropagation



$$let:\ \mathcal{L} = \Sigma_{i=0}^{T}\mathcal{L}_i \qquad let:\ \mathcal{L}_k^{-} = \Sigma_{i=0}^{k}\mathcal{L}_i$$

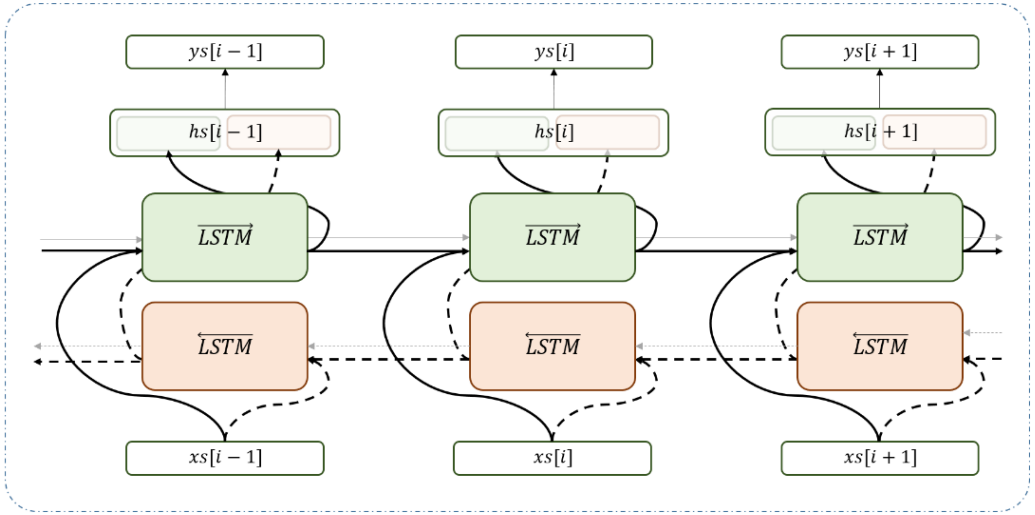
$$\frac{\partial \mathcal{L}}{\partial \overleftarrow{W}_{xh}} = \sum_{i=0}^{T}\frac{\partial L}{\partial \overleftarrow{hs}_i}\frac{\partial \overleftarrow{hs}_i}{\partial \overleftarrow{W}_{xh}} = \sum_{i=0}^{T}\left(\sum_{j}^{T}\frac{\partial L_j}{\partial \overleftarrow{hs}_i}\right)\frac{\partial \overleftarrow{hs}_i}{\partial \overleftarrow{W}_{xh}} = \sum_{i=0}^{T}\left(\frac{\partial \mathcal{L}_{i-1}^{-}}{\partial \overleftarrow{hs}_i} + \frac{\partial \mathcal{L}_i}{\partial \overleftarrow{hs}_i}\right)\frac{\partial \overleftarrow{hs}_i}{\partial \overleftarrow{W}_{xh}} = \sum_{i=0}^{T}\left(\left[\frac{\partial \mathcal{L}_{i-1}^{-}}{\partial \overleftarrow{hs}_i}\right]_{\frac{H}{2}:H} + \left[\frac{\partial \mathcal{L}_i}{\partial \overleftarrow{hs}_i}\right]_{\frac{H}{2}:0}\right)\frac{\partial \overleftarrow{hs}_i}{\partial \overleftarrow{W}_{xh}}$$

$$= \sum_{i=0}^{T}\left(\left[\frac{\partial \mathcal{L}_{i-1}^{-}}{\partial \overleftarrow{hs}_i}\right]_{\frac{H}{2}:H} + \left[\frac{\partial \mathcal{L}_i}{\partial \overleftarrow{hs}_i}\right]_{\frac{H}{2}:0}\right) \odot \left(1 - \overleftarrow{hs}_i^{2}\right)xs_i^T$$

EWHA EURON

#07 양방향 RNN

양방향 LSTM



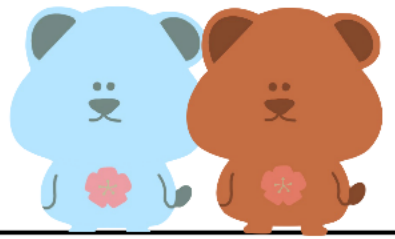

```
class biLSTM(nn.Module):
    def __init__(self, num_classes, input_size, hidden_size, num_layers, seq_length):
        super(biLSTM, self).__init__()
        self.num_classes = num_classes
        self.num_layers = num_layers
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.seq_length = seq_length

        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
                            num_layers=num_layers, bidirectional=True, batch_first=True)
        self.fc =  nn.Linear(hidden_size*2, num_classes)
        self.relu = nn.ReLU()

    def forward(self,x):
        h_0 = Variable(torch.zeros(self.num_layers*2, x.size(0), self.hidden_size))
        c_0 = Variable(torch.zeros(self.num_layers*2, x.size(0), self.hidden_size))
        out, _ = self.lstm(x, (h_0, c_0))
        out = self.fc(out[:, -1, :])
        out = self.relu(out)
        return out
```

양방향 옵션 = True

이전, 이후 시점의 hidden layer을 연결 > hidden size의 2배

한번 학습하는데 2개의 계층이 필요

THANK YOU



EWHA EURON