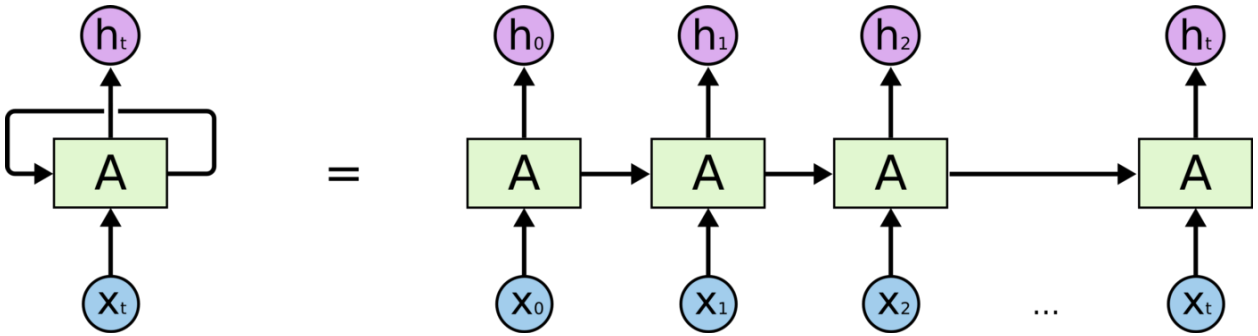
밑바닥부터 시작하는 딥러닝2

20213093 정현우

CONTENTS

1. RNN

RNN, BPTT, Time RNN, RNNLM



RNN

RNN은 시계열 데이터를 처리하기 유리함.

그림과 같이 순환하는 신경망을 의미함.

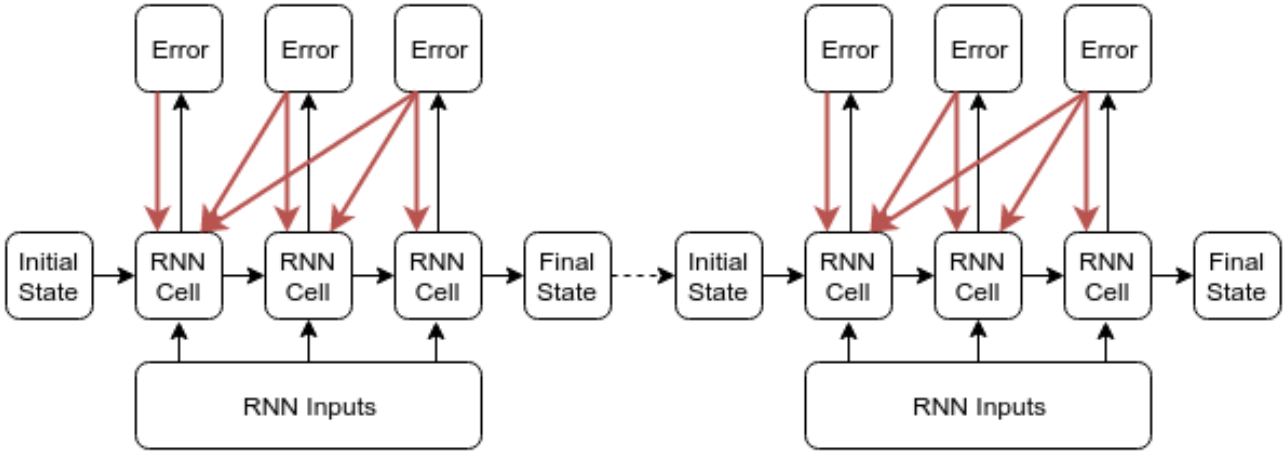
때문에 과거의 정보를 기억하면서 데이터 갱신

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

위 식으로 정보를 갱신 함.

1. RNN

RNN, BPTT, Time RNN, RNNLM



BPTT

Backpropagation Through Time

시간 방향으로 펼친 신경망의 오차 역전파법

문제점 : 시간의 크기가 커지면 소비하는 컴퓨팅 자원도 그만큼 커짐.

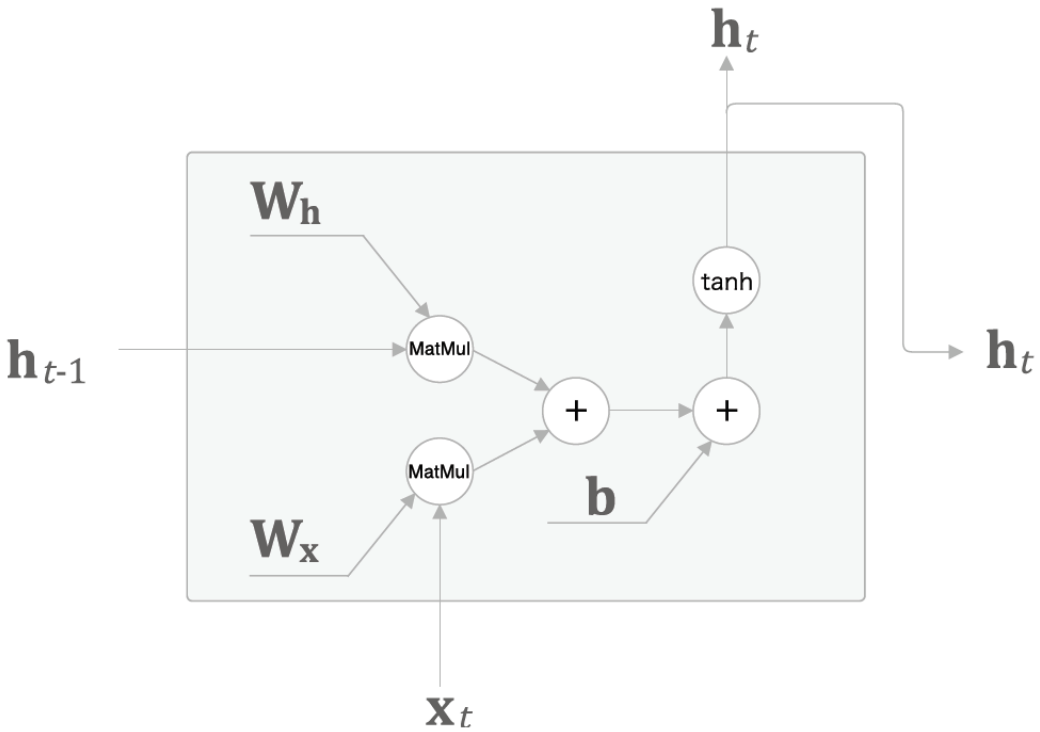
해결법 : Truncated BPTT

역전파의 신경망을 적당히 자른다. 잘라낸 신경망 단위로 학습을 진행한다.

1. RNN

RNN, BPTT, Time RNN, RNNLM

그림 6-2 RNN 계층의 계산 그래프(MatMul 노드는 행렬 곱을 나타냄)



RNN 구현

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

위의 식을 보기 좋게 풀어 놓은 것이다.

가중치끼리 행렬 곱을 하고 편향을 더한 후에

tanh 함수를 지나게 되면 된다.

1. RNN

RNN, BPTT, Time RNN, RNNLM

```python
class RNN:
  def __init__(self, Wx, Wh, b):
    self.params = [Wx, Wh, b]
    self.grads = [np.zeros_like(Wx), np.zeros_like(Wh), np.zeros_like(b)]
    self.cache= None
  def forward(self, x, h_prev):
    Wx, Wh, b = self.params
    t = np.matmul(h_prev, Wh) + np.matmul(x, Wx) + b
    h_next = np.tanh(t)

    self.cache = (x, h_prev, h_next)
    return h_next
```

0.3s

RNN 구현

이를 코드로 옮기면 다음과 같다.

파라미터를 입력 받아 저장하고

위의 식을 그대로 연산해주면 된다.

Cache는 역전파 연산을 위해 필요한 것이다.

1. RNN

RNN, BPTT, Time RNN, RNNLM

```python
def backward(self, dh_next):
    Wx, Wh, b = self.params
    x, h_prev, h_next = self.cache

    dt = dh_next * (1 - h_next ** 2)
    db = np.sum(dt, axis=0)
    dWh = np.matmul(h_prev.T, dt)
    dh_prev = np.matmul(dt, Wh.T)
    dWx = np.matmul(x.T, dt)
    dx = np.matmul(dt, Wx.T)

    self.grads[0][...] = dWx
    self.grads[1][...] = dWh
    self.grads[2][...] = db

    return dx, dh_prev
```

RNN 구현

Tanh 함수의 경우 (1 – x ** 2) 를 곱해서 보낸다.

나머지는 앞에서 배웠던

Matmul 연산의 역전파이다.

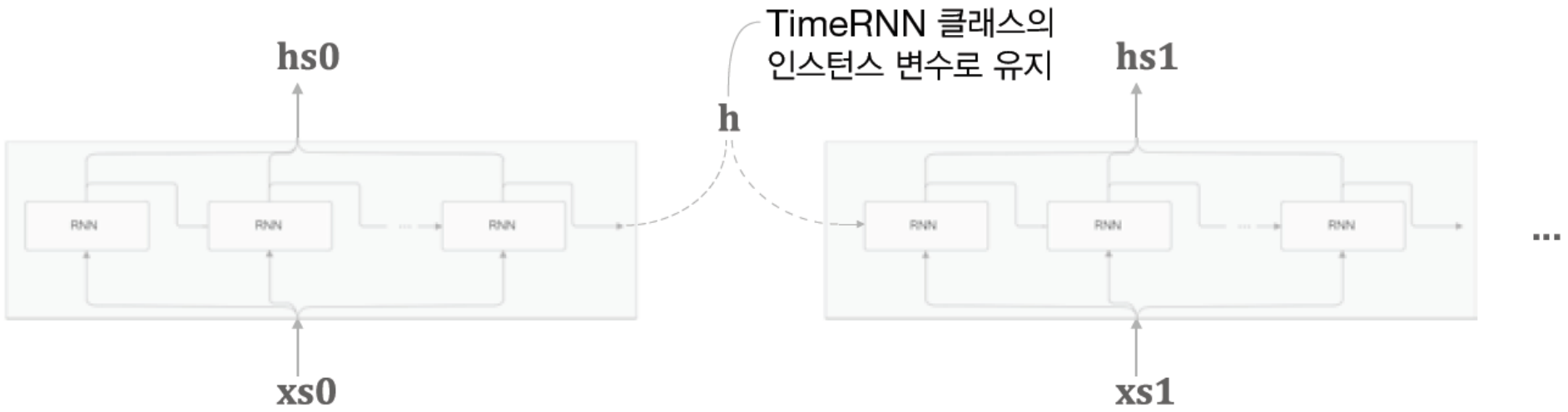
이를 grads 리스트에 저장해준다.

1. RNN

RNN, BPTT, Time RNN, RNNLM

Time RNN

그림 5-22 Time RNN 계층은 은닉 상태를 인스턴스 변수 h로 보관한다. 그러면 은닉 상태를 다음 블록에 인계할 수 있다.



1. RNN

RNN, BPTT, Time RNN, RNNLM

Time RNN

```
class TimeRNN:
  def __init__(self, Wx, Wh, b, stateful=False):
    self.params = [Wx, Wh, b]
    self.grads = [np.zeros_like(Wx), np.zeros_like(Wh), np.zeros_like(b)]

    self.h, self.dh = None, None
    self.stateful = stateful

  def set_state(self, h):
    self.h = h
  def reset_state(self):
    self.h = None
```

Stateful 에는 bool 형 데이터를 저장.

True이면 은닉 상태를 유지한다는 뜻임.

False일 때는 영행렬로 초기화함.

1. RNN

RNN, BPTT, Time RNN, RNNLM

```
def forward(self, xs):
    Wx, Wh, b = self.params
    N, T, D = xs.shape
    D, H = Wx.shape

    self.layers = []
    hs = np.empty((N, T, H), dtype = 'f')

    if not self.stateful or self.h is None:
        self.h = np.zeros((N, H), dtype='f')

    for t in range(T):
        layer = RNN(*self.params)
        self.h = layer.forward(xs[:, t, :], self.h)
        hs[:, t, :] = self.h
        self.layers.append(layer)

    return hs
```

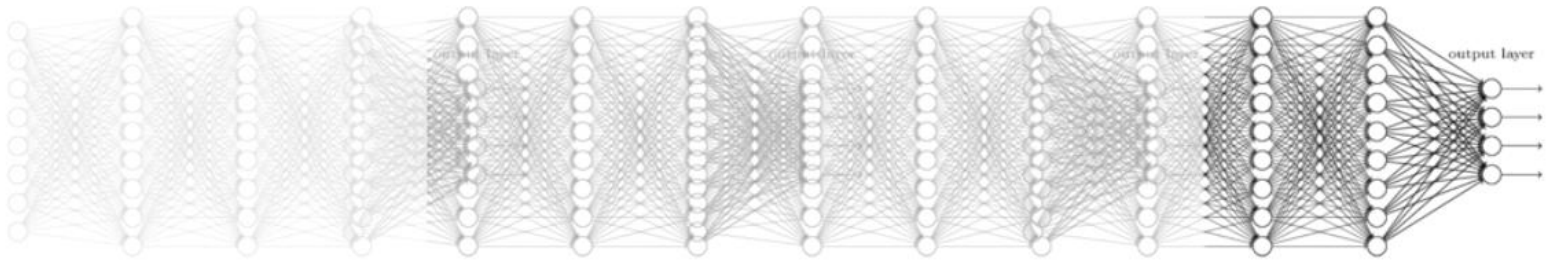
Time RNN

2. 게이트 RNN

기울기 소실/폭발, LSTM, 게이트, LSTM 구현

기울기 소실/ 폭발

Vanishing gradient (NN winter2: 1986-2006)



Tanh의 미분 값은

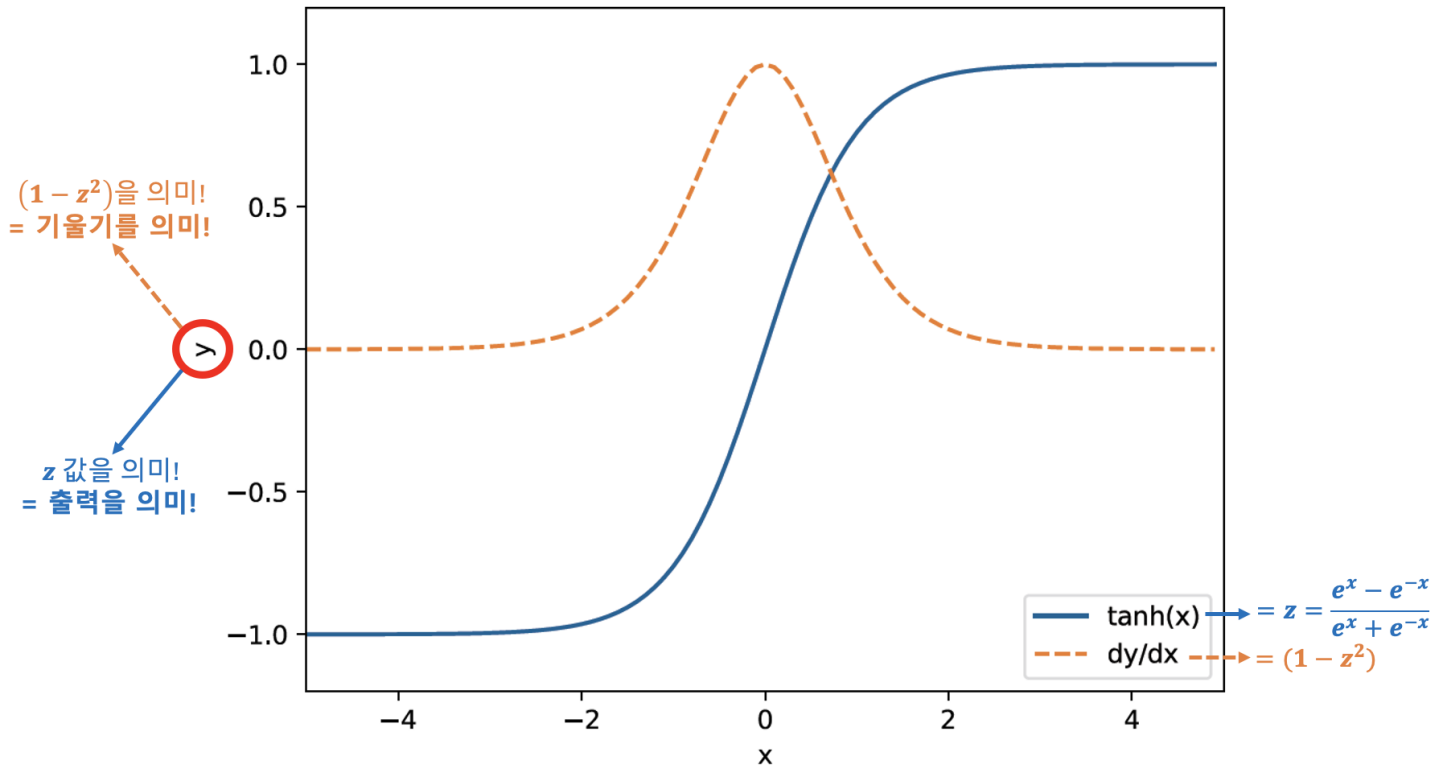
(1 – y ** 2) 이기 때문에

층을 지날수록 기울기가 줄어들게 된다.

2. 게이트 RNN

기울기 소실/폭발, LSTM, 게이트, LSTM 구현

기울기 소실/ 폭발



2. 게이트 RNN

기울기 소실/폭발, LSTM, 게이트, LSTM 구현

$$\text{if } \|g\| > \ threshold$$

$$g \leftarrow \frac{threshold \times g}{\|g\|}$$

$$where: \ g \ is \ the \ gradient \ and$$

$$\|g\| \ is \ the \ norm \ of \ the \ gradient$$

기울기 폭발 대책

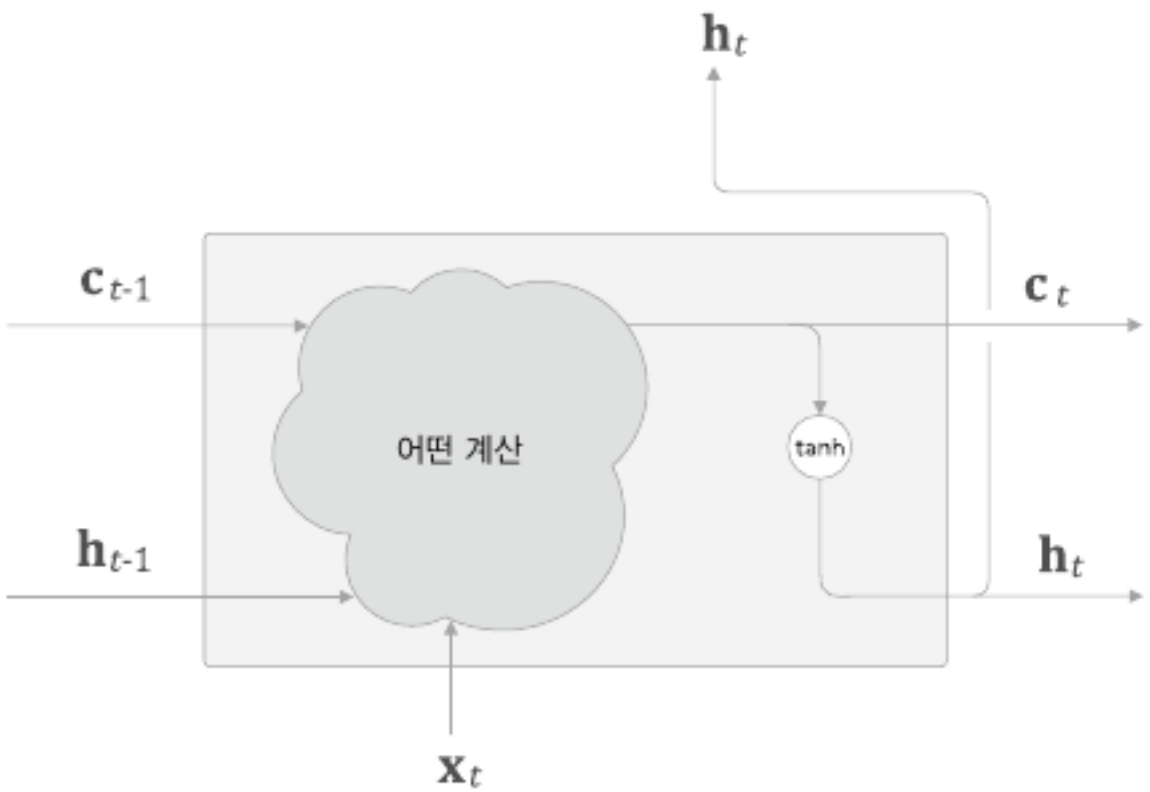
기울기 클리핑을 사용한다.

기울기의 L2 노름이 문턱값을 초과하면 기울기를 줄여준다.

단순하지만 잘 작동한다고 한다.

2. 게이트 RNN

기울기 소실/폭발, LSTM, 게이트, LSTM 구현



기울기 소실과 LSTM

LSTM의 게이트는 수도꼭지와 같은 역할을 한다.

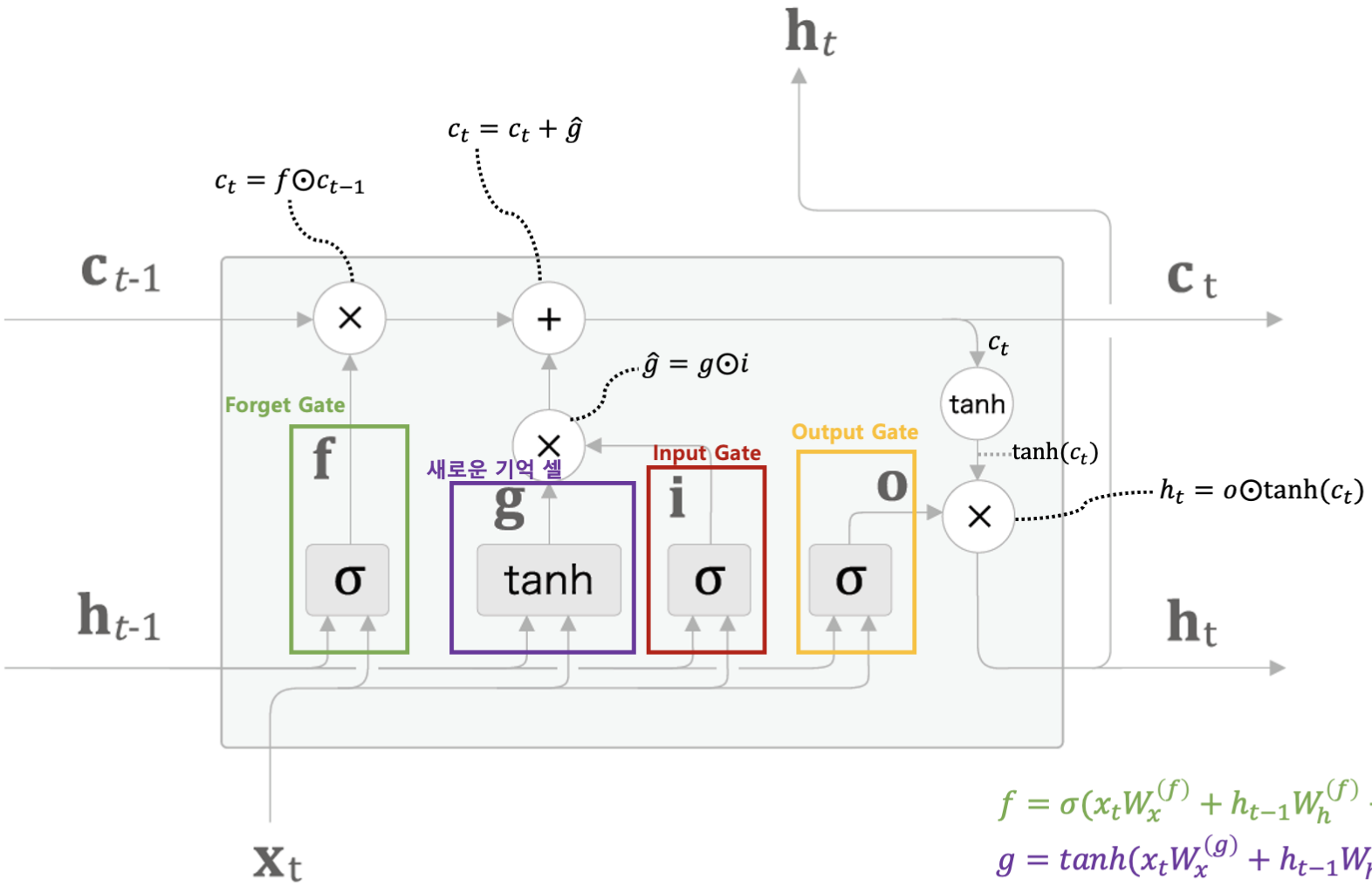
게이트는

Out gate, forget gate, input gate

새로운 기억 셀 이 있다.

2. 게이트 RNN

기울기 소실/폭발, LSTM, 게이트, LSTM 구현



$$f = \sigma(x_t W_x^{(f)} + h_{t-1} W_h^{(f)} + b^{(f)})$$
$$g = tanh(x_t W_x^{(g)} + h_{t-1} W_h^{(g)} + b^{(g)})$$
$$i = \sigma(x_t W_x^{(i)} + h_{t-1} W_h^{(i)} + b^{(i)})$$
$$o = \sigma(x_t W_x^{(o)} + h_{t-1} W_h^{(o)} + b^{(o)})$$

2. 게이트 RNN

기울기 소실/폭발, LSTM, 게이트, LSTM 구현

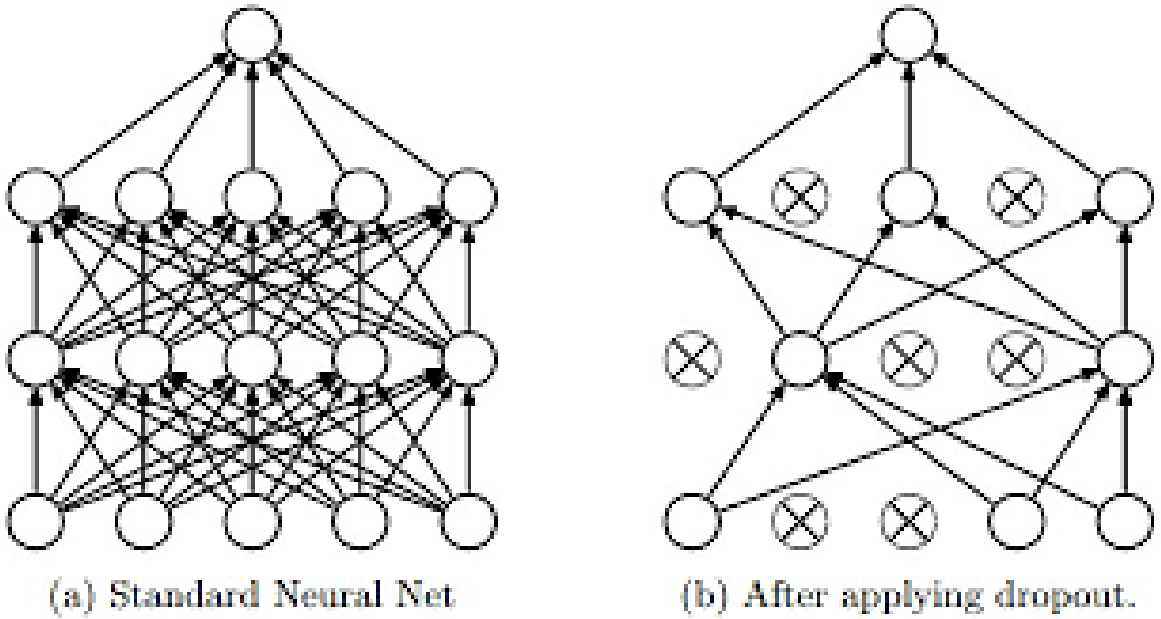
```
def forward(self, x, h_prev, c_prev):

  A = np.matmul(x, Wx) + np.matmul(h_prev, Wh) + b

  f = A[:, :H]
  g = A[:, H:2*H]
  i = A[:, 2*H:3*H]
  o = A[:, 3*H:]

  f = sigmoid(f)
  g = sigmoid(g)
  i = sigmoid(i)
  o = sigmoid(o)

  c_next = f * c_prev + g * i
  h_next = o * np.tanh(c_next)

  self.cache = (x,  h_prev, i, f, g, o, c_next)
  return h_next, c_next
```

LSTM 순전파 구현

최종적으로 다음과 같은 연산을 한다.

2. 게이트 RNN

기울기 소실/폭발, LSTM, 게이트, LSTM 구현

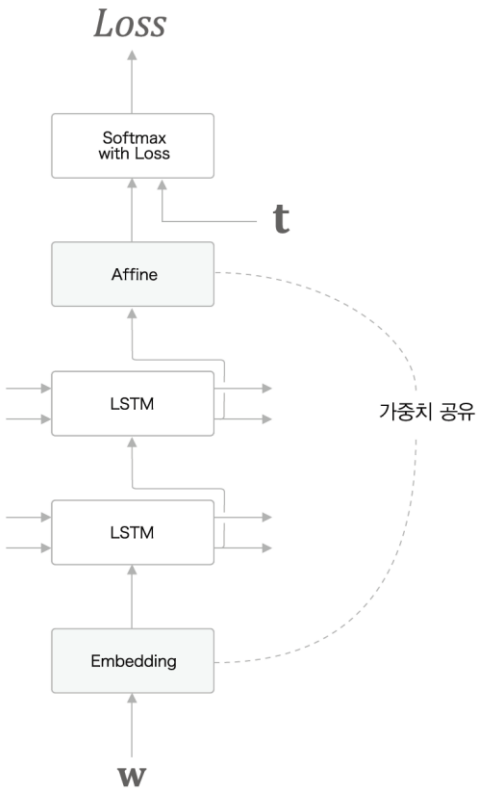


(a) Standard Neural Net

(b) After applying dropout.

추가 개선

그림 6-35 언어 모델에서의 가중치 공유 예: Embedding 계층과 Softmax 앞단의 Affine 계층이 가중치를 공유한다.



Thank you