6장 게이트가 추가된 RNN

윤 예 준

INDEX

01

RNN의 문제점

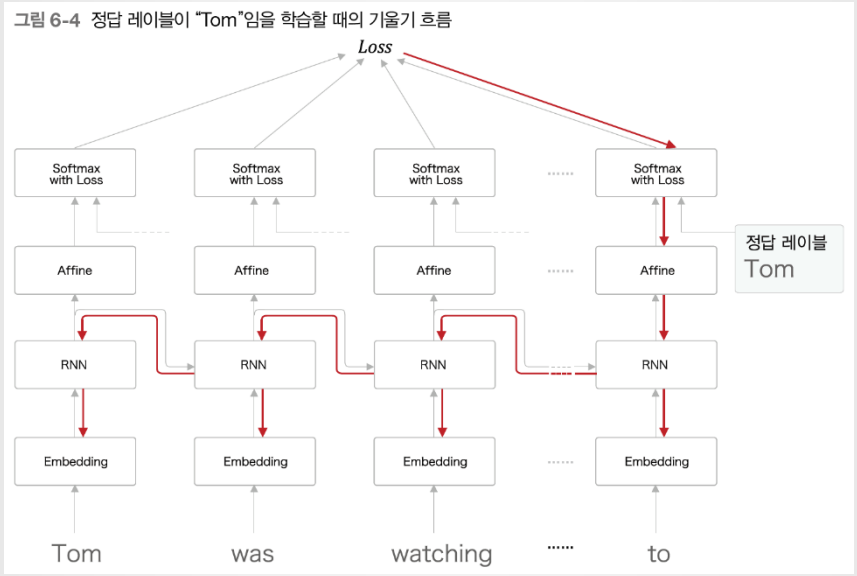
기울기 소실 : 역전파의 기울기 값이 점점 작아지다가 사라지는 현상

기울기 폭발 : 역전파의 기울기 값이 점점 커지다가 매우 커지는 현상

그림 6-3 "?"에 들어갈 단어는?: (어느 정도의) 장기 기억이 필요한 문제의 예

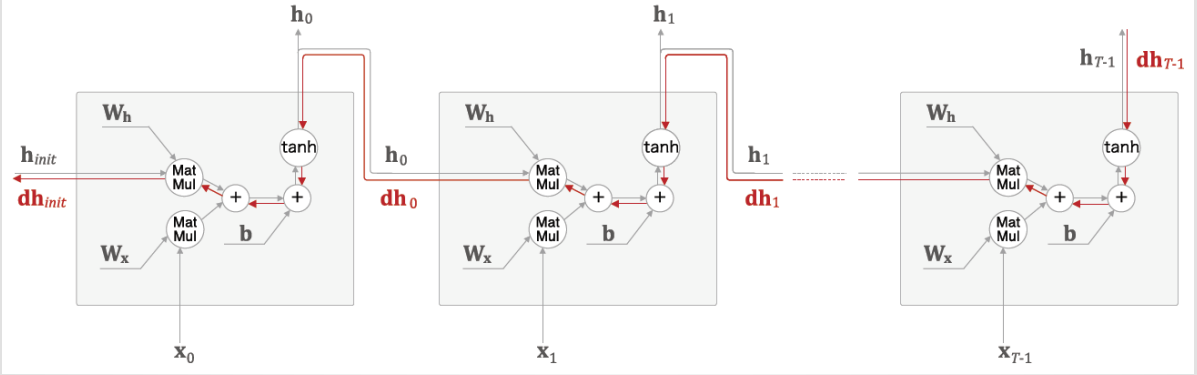
Tom was watching TV in his room. Mary came into the room. Mary said hi to ?

그림 6-4 정답 레이블이 "Tom"임을 학습할 때의 기울기 흐름



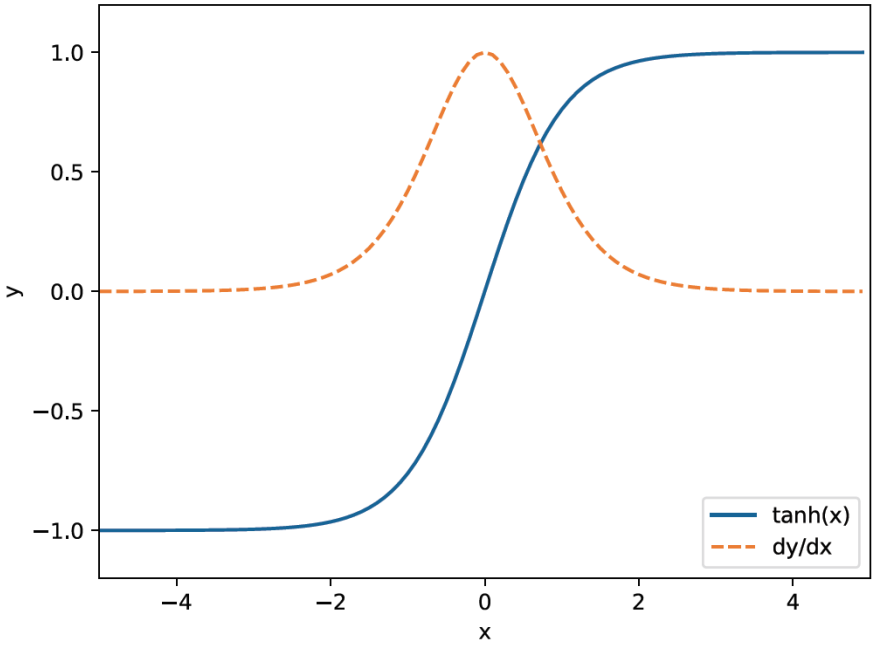
기울기 소실과 기울기 폭발 원인

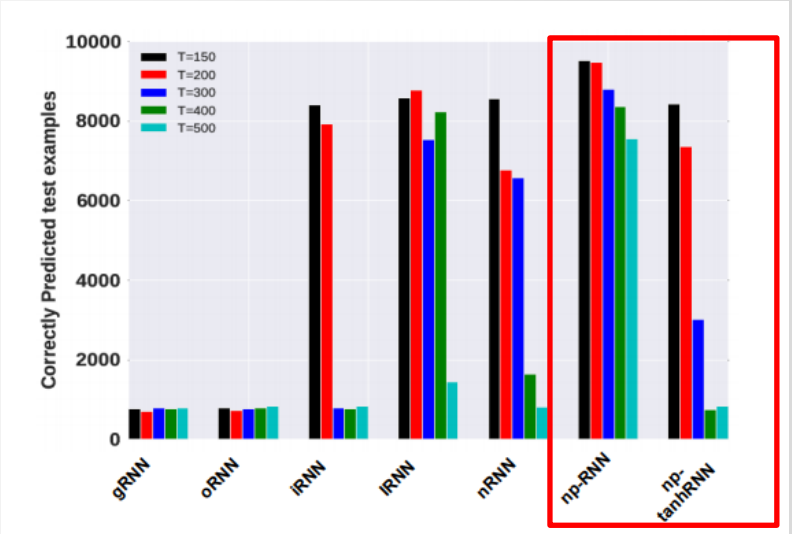
그림 6-5 RNN 계층에서 시간 방향으로의 기울기 전파



기울기 소실과 기울기 폭발 원인

그림 6-6 $y = \tanh(x)$의 그래프(점선은 미분)





Improving performance of recurrent neural network with relu nonlinearity

기울기 소실과 기울기 폭발 원인

그림 6-7 RNN 계층의 행렬 곱에만 주목했을 때의 역전파의 기울기

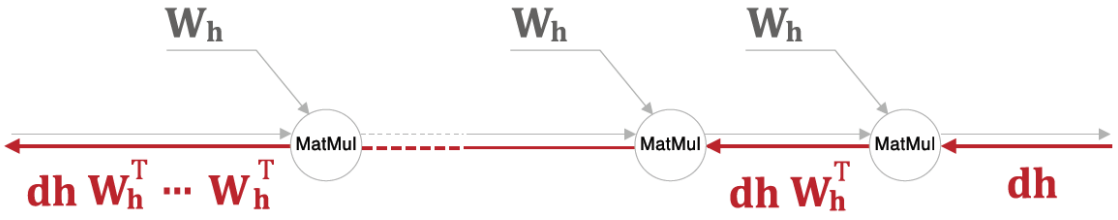


그림 6-8 기울기 dh는 시간 크기에 비례하여 지수적으로 증가한다.

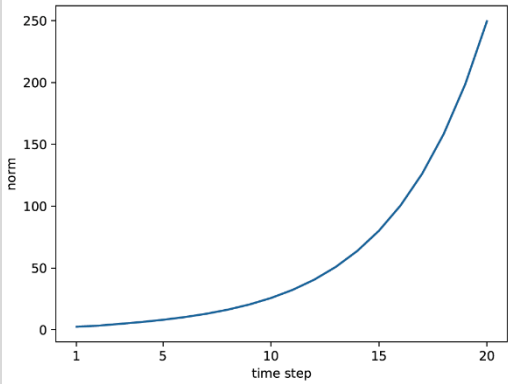
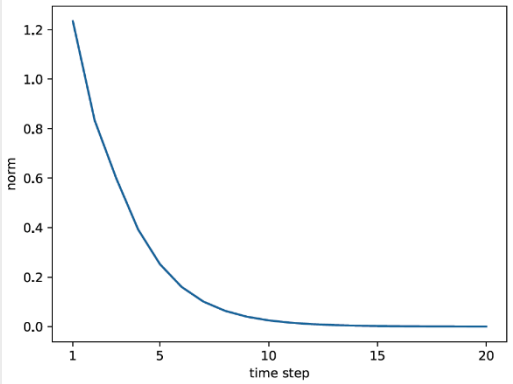


그림 6-9 기울기 dh는 시간 크기에 비례하여 지수적으로 감소한다.




```python
# coding: utf-8
import numpy as np
import matplotlib.pyplot as plt


N = 2   # 미니배치 크기
H = 3   # 은닉 상태 벡터의 차원 수
T = 20  # 시계열 데이터의 길이

dh = np.ones((N, H))

np.random.seed(3) # 재현할 수 있도록 난수의 시드 고정

Wh = np.random.randn(H, H)
#Wh = np.random.randn(H, H) * 0.5

norm_list = []
for t in range(T):
    dh = np.dot(dh, Wh.T)
    norm = np.sqrt(np.sum(dh**2)) / N
    norm_list.append(norm)

print(norm_list)

# 그래프 그리기
plt.plot(np.arange(len(norm_list)), norm_list)
plt.xticks([0, 4, 9, 14, 19], [1, 5, 10, 15, 20])
plt.xlabel('시간 크기(time step)')
plt.ylabel('노름(norm)')
plt.show()
```

기울기 소실과 기울기 폭발 원인

그림 6-7 RNN 계층의 행렬 곱에만 주목했을 때의 역전파의 기울기

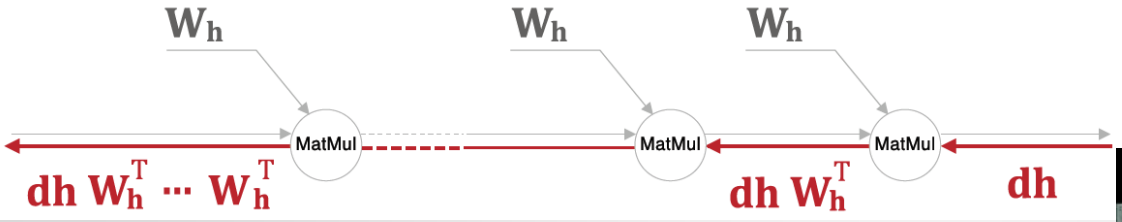


그림 6-8 기울기 dh는 시간 크기에 비례하여 지수적으로 증가한다.

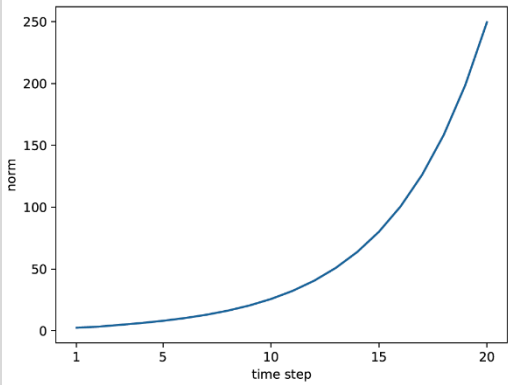
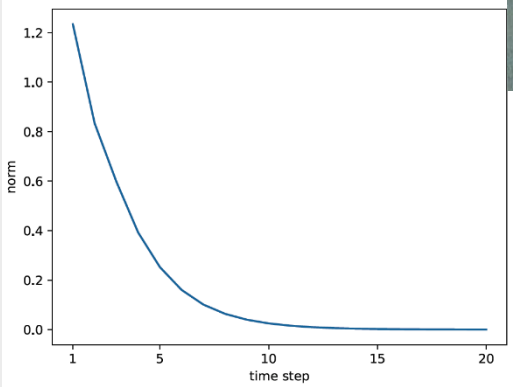


그림 6-9 기울기 dh는 시간 크기에 비례하여 지수적으로 감소한다.




```python
# coding: utf-8
import numpy as np
import matplotlib.pyplot as plt


N = 2  # 미니배치 크기
H = 3  # 은닉 상태 벡터의 차원 수
T = 20  # 시계열 데이터의 길이
```

정의 1

A가 $m \times n$ 행렬이고, $\lambda_1, \lambda_2, \cdots, \lambda_n$이 A^TA의 고유값이라면

$$\sigma_1 = \sqrt{\lambda_1}, \quad \sigma_2 = \sqrt{\lambda_2}, \ldots, \quad \sigma_n = \sqrt{\lambda_n}$$

을 A의 ***특이값***(singular values)이라고 한다.

```python
        dh = np.dot(dh, Wh.T)
        norm = np.sqrt(np.sum(dh**2)) / N
        norm_list.append(norm)

print(norm_list)

# 그래프 그리기
plt.plot(np.arange(len(norm_list)), norm_list)
plt.xticks([0, 4, 9, 14, 19], [1, 5, 10, 15, 20])
plt.xlabel('시간 크기(time step)')
plt.ylabel('노름(norm)')
plt.show()
```

기울기 폭발 해결 방법

Gradients clipping

$\hat{g}$: 모든 매개변수에 대한 기울기를 하나로 모은 것

$$\text{if } \left\|\hat{\mathbf{g}}\right\| \geq threshold:$$

$$\hat{\mathbf{g}} = \frac{threshold}{\left\|\hat{\mathbf{g}}\right\|}\hat{\mathbf{g}}$$

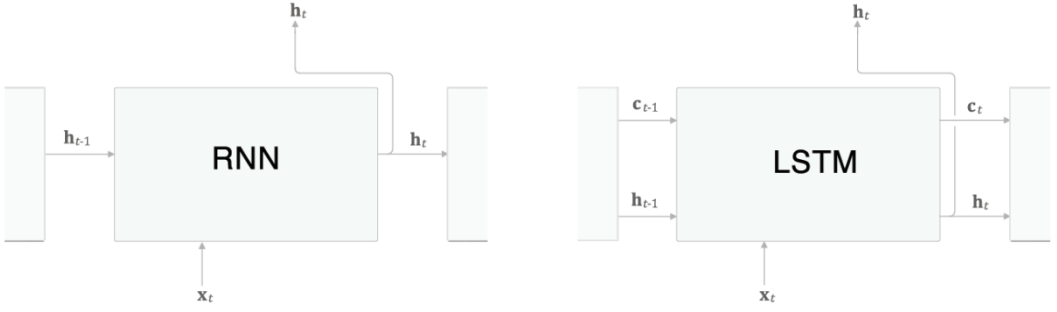
```python
# coding: utf-8
import numpy as np


dW1 = np.random.rand(3, 3) * 10
dW2 = np.random.rand(3, 3) * 10
grads = [dW1, dW2]
max_norm = 5.0


def clip_grads(grads, max_norm):
    total_norm = 0
    for grad in grads:
        total_norm += np.sum(grad ** 2)
    total_norm = np.sqrt(total_norm)

    rate = max_norm / (total_norm + 1e-6)
    if rate < 1:
        for grad in grads:
            grad *= rate


print('before:', dW1.flatten())
clip_grads(grads, max_norm)
print('after:', dW1.flatten())

```

02

기울기 소실과 LSTM

LSTM 인터페이스

그림 6-11 RNN 계층과 LSTM 계층 비교



C (memory cell) : LSTM 전용의 기억 메커니즘

LSTM 계층 조립

그림 6-12 기억 셀 $\mathbf{c}_t$를 바탕으로 은닉 상태 $\mathbf{h}_t$를 계산하는 LSTM 계층

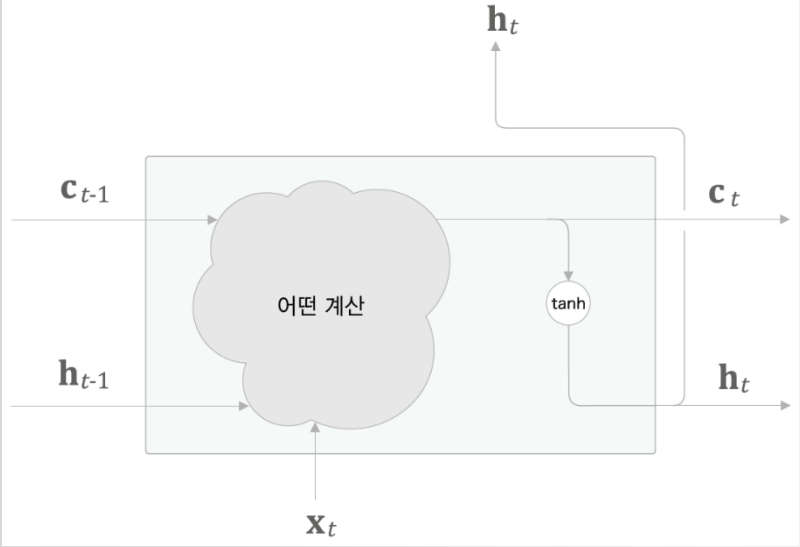
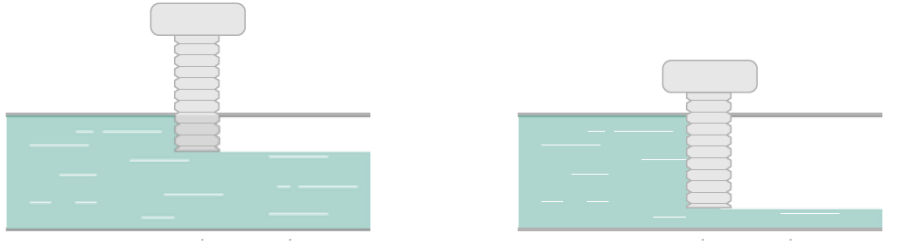


그림 6-14 물이 흐르는 양을 0.0~1.0 범위에서 제어한다.

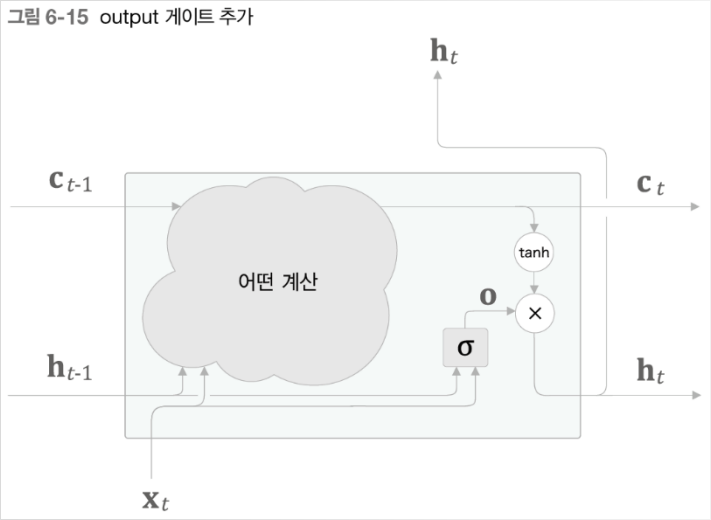


게이트

전용 가중치 매개변수 이용
가중치 매개변수는 학습 데이터로부터 갱신
게이트 열림 상태를 구할 때는 시그모이드 함수 사용
※ '게이트를 얼마나 열까' 라는 것도 데이터로부터 (자동으로) 학습

Output 게이트

그림 6-15 output 게이트 추가



$$\mathbf{o} = \sigma(\mathbf{x}_t \mathbf{W}_{\mathbf{x}}^{(\mathbf{o})} + \mathbf{h}_{t-1} \mathbf{W}_{\mathbf{h}}^{(\mathbf{o})} + \mathbf{b}^{(\mathbf{o})})$$

Ouput 게이트의 열림상태
입력 x_t와 이전 상태 h_{t-1}로 구함

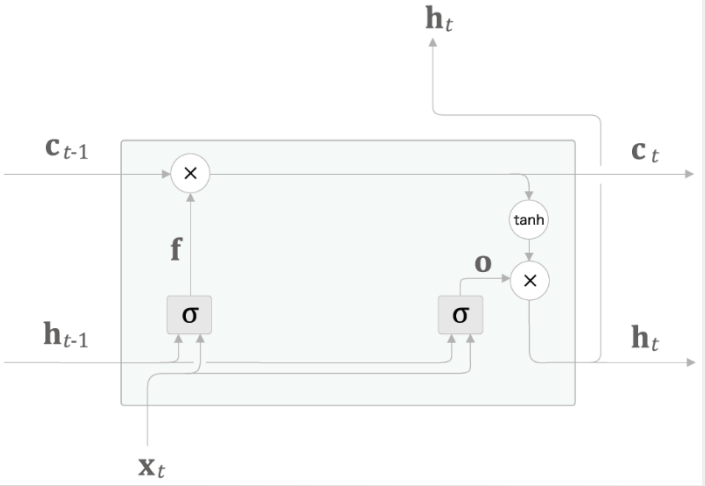
Ouput 게이트의 출력
위 식으로부터 구함

$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

h_t는 아다마르 곱으로 계산됨.

Forget 게이트

그림 6-16 forget 게이트 추가

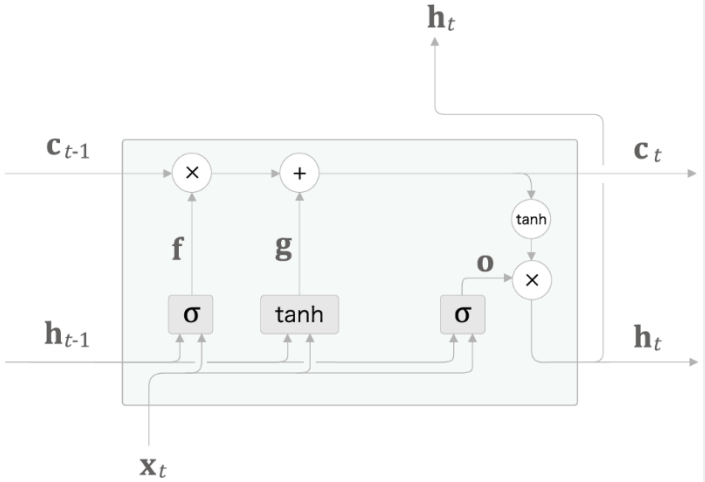


$$\mathbf{f} = \sigma(\mathbf{x}_t \mathbf{W}_x^{(\mathbf{f})} + \mathbf{h}_{t-1} \mathbf{W}_\mathbf{h}^{(\mathbf{f})} + \mathbf{b}^{(\mathbf{f})})$$

$$c_t = f \odot c_{t-1}$$

새로운 기억 셀

그림 6-17 새로운 기억 셀에 필요한 정보를 추가

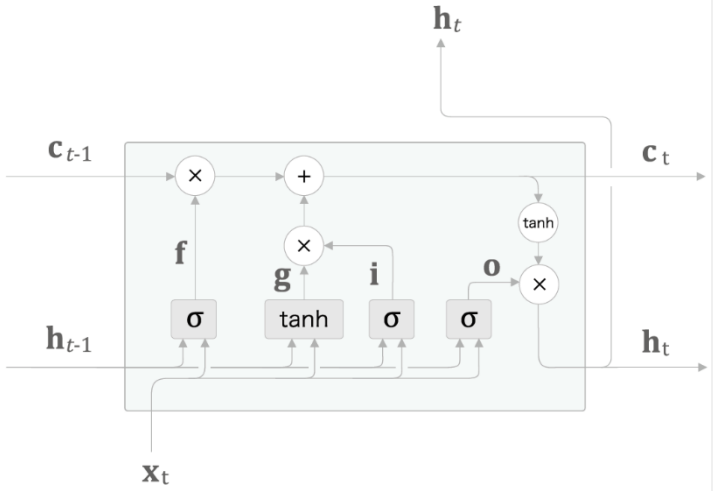


새로 기억해야 할 정보를 기억셀에 추가

$$\mathbf{g} = \tanh(\mathbf{x}_t \mathbf{W}_x^{(\mathbf{g})} + \mathbf{h}_{t-1} \mathbf{W}_\mathbf{h}^{(\mathbf{g})} + \mathbf{b}^{(\mathbf{g})})$$

Input 게이트

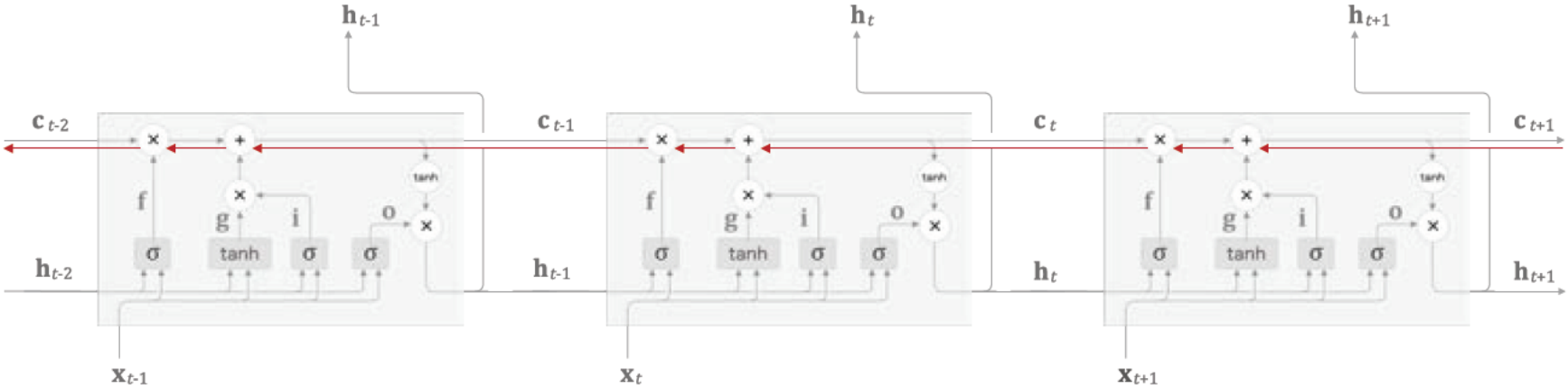
그림 6-18 input 게이트 추가



$$\mathbf{i} = \sigma(\mathbf{x}_t \mathbf{W}_x^{(\mathbf{i})} + \mathbf{h}_{t-1} \mathbf{W}_\mathbf{h}^{(\mathbf{i})} + \mathbf{b}^{(\mathbf{i})})$$

LSTM의 기울기 흐름

그림 6-19 기억 셀의 역전파



03

LSTM 구현

LSTM 구현

$$\mathbf{f} = \sigma(\mathbf{x}_t \mathbf{W}_{\mathbf{x}}^{(\mathbf{f})} + \mathbf{h}_{t-1} \mathbf{W}_{\mathbf{h}}^{(\mathbf{f})} + \mathbf{b}^{(\mathbf{f})})$$

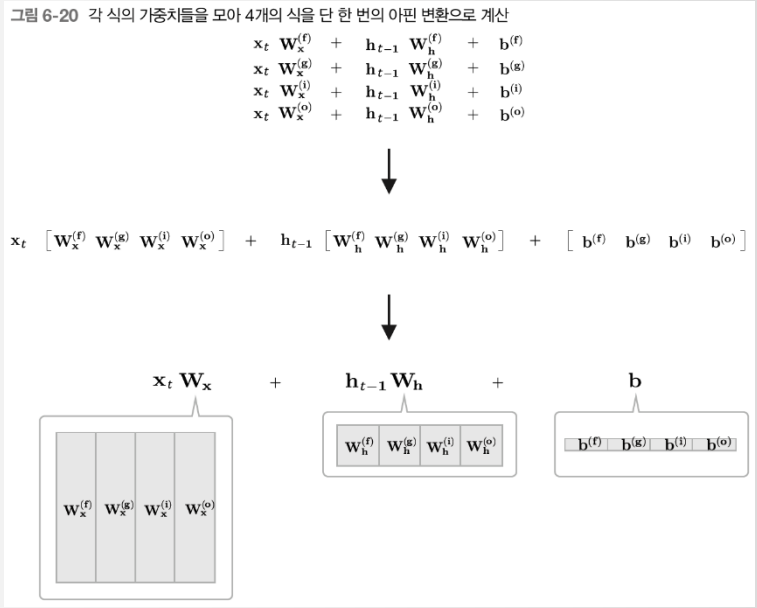
$$\mathbf{g} = \tanh(\mathbf{x}_t \mathbf{W}_{\mathbf{x}}^{(\mathbf{g})} + \mathbf{h}_{t-1} \mathbf{W}_{\mathbf{h}}^{(\mathbf{g})} + \mathbf{b}^{(\mathbf{g})})$$

$$\mathbf{i} = \sigma(\mathbf{x}_t \mathbf{W}_{\mathbf{x}}^{(\mathbf{i})} + \mathbf{h}_{t-1} \mathbf{W}_{\mathbf{h}}^{(\mathbf{i})} + \mathbf{b}^{(\mathbf{i})})$$

$$\mathbf{o} = \sigma(\mathbf{x}_t \mathbf{W}_{\mathbf{x}}^{(\mathbf{o})} + \mathbf{h}_{t-1} \mathbf{W}_{\mathbf{h}}^{(\mathbf{o})} + \mathbf{b}^{(\mathbf{o})})$$

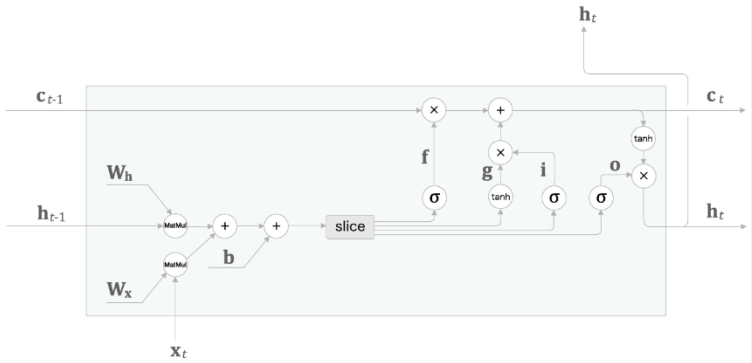
$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{g} \odot \mathbf{i}$$

$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$



LSTM 구현

그림 6-21 4개분의 가중치를 모아 아핀 변환을 수행하는 LSTM의 계산 그래프




```python
class LSTM:
    def __init__(self, Wx, Wh, b):
        '''

        Parameters
        ----------
        Wx: 입력 x에 대한 가중치 매개변수(4개분의 가중치가 담겨 있음)
        Wh: 은닉 상태 h에 대한 가장추 매개변수(4개분의 가중치가 담겨 있음)
        b: 편향 (4개분의 편향이 담겨 있음)
        '''
        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, c_prev):
        Wx, Wh, b = self.params
        N, H = h_prev.shape

        A = np.dot(x, Wx) + np.dot(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 = np.tanh(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, c_prev, i, f, g, o, c_next)
        return h_next, c_next
```

LSTM 구현

그림 6-22 아핀 변환 시의 형상 추이(편향은 생략)

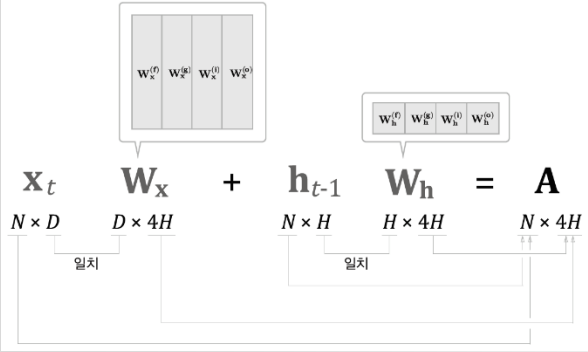
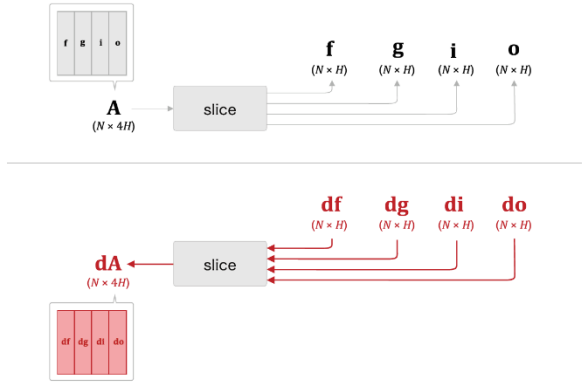


그림 6-23 slice 노드의 순전파(위)와 역전파(아래)




```python
def backward(self, dh_next, dc_next):
    Wx, Wh, b = self.params
    x, h_prev, c_prev, i, f, g, o, c_next = self.cache

    tanh_c_next = np.tanh(c_next)

    ds = dc_next + (dh_next * o) * (1 - tanh_c_next ** 2)

    dc_prev = ds * f

    di = ds * g
    df = ds * c_prev
    do = dh_next * tanh_c_next
    dg = ds * i

    di *= i * (1 - i)
    df *= f * (1 - f)
    do *= o * (1 - o)
    dg *= (1 - g ** 2)

    dA = np.hstack((df, dg, di, do))

    dWh = np.dot(h_prev.T, dA)
    dWx = np.dot(x.T, dA)
    db = dA.sum(axis=0)

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

    dx = np.dot(dA, Wx.T)
    dh_prev = np.dot(dA, Wh.T)

    return dx, dh_prev, dc_prev
```

Time LSTM 구현

그림 6-24 Time LSTM의 입출력

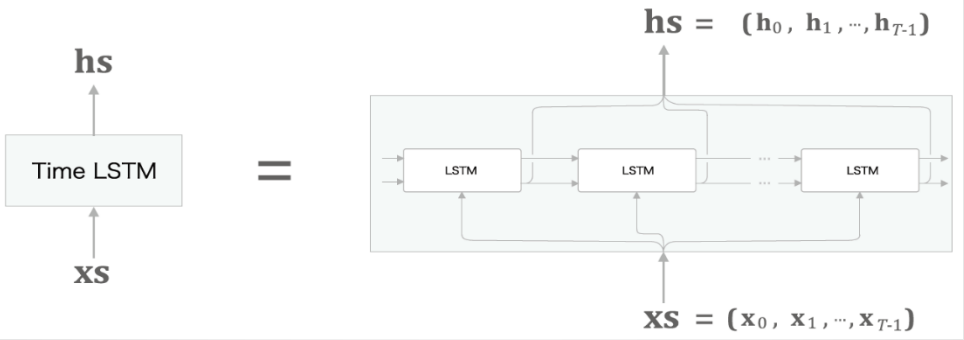
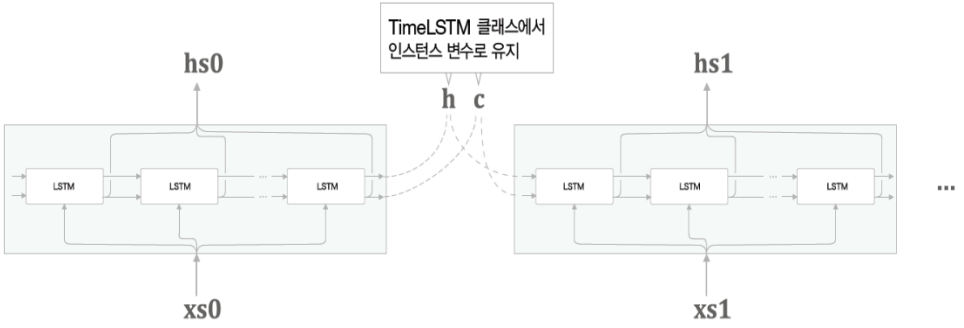


그림 6-25 Time LSTM 역전파의 입출력



Time LSTM 구현

```python
class TimeLSTM:
    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.layers = None

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

    def forward(self, xs):
        Wx, Wh, b = self.params
        N, T, D = xs.shape
        H = Wh.shape[0]

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

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

            self.layers.append(layer)

        return hs
```

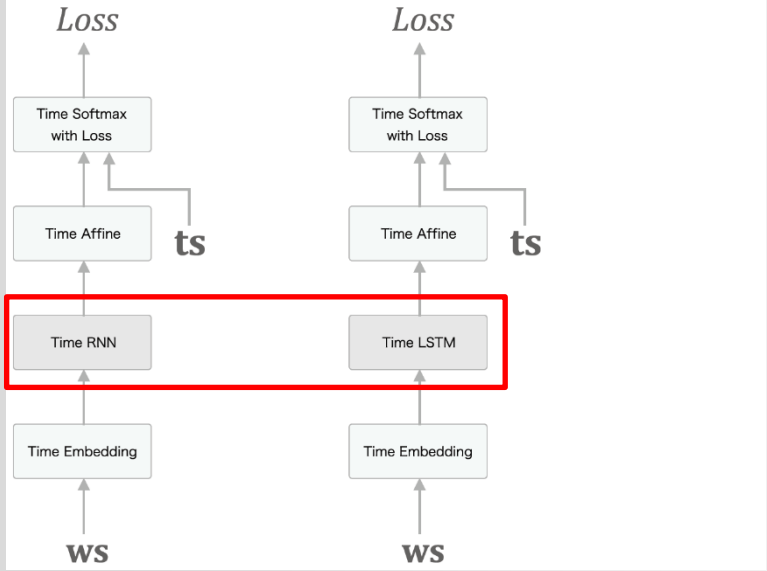
```python
def backward(self, dhs):
    Wx, Wh, b = self.params
    N, T, H = dhs.shape
    D = Wx.shape[0]

    dxs = np.empty((N, T, D), dtype='f')
    dh, dc = 0, 0

    grads = [0, 0, 0]
    for t in reversed(range(T)):
        layer = self.layers[t]
        dx, dh, dc = layer.backward(dhs[:, t, :] + dh, dc)
        dxs[:, t, :] = dx
        for i, grad in enumerate(layer.grads):
            grads[i] += grad

    for i, grad in enumerate(grads):
        self.grads[i][...] = grad
    self.dh = dh
    return dxs

def set_state(self, h, c=None):
    self.h, self.c = h, c

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

04

LSTM을 사용한 언어 모델

LSTM을 사용한 언어 모델

그림 6-26 언어 모델의 신경망 구성(왼쪽은 앞 장에서 작성한 Time RNN을 이용한 모델, 오른쪽은 이번 장에서 작성하는 Time LSTM을 이용한 모델)




```python
# 기울기 클리핑을 적용하여 학습
trainer.fit(xs, ts, max_epoch, batch_size, time_size, max_grad,
            eval_interval=20)
trainer.plot(ylim=(0, 500))

# 테스트 데이터로 평가
model.reset_state()
ppl_test = eval_perplexity(model, corpus_test)
print('테스트 퍼플렉서티: ', ppl_test)

# 매개변수 저장
model.save_params()
```

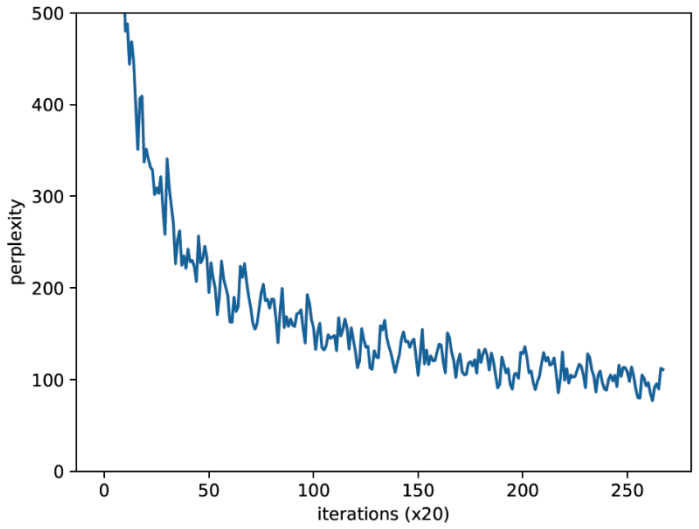
1. 기울기 클리핑 수행
2. 데이터 크기 때문에 모든 에폭에서 평가하지 않고, 20번 반복 될 때마다 평가
3. 학습이 끝난 후 테스트 데이터를 사용해 퍼플렉서티를 평가 (LSTM의 은닉 상태와 기억셀을 재설정하여 평가 수행)
4. 학습이 완료된 매개변수들을 파일로 저장

LSTM을 사용한 언어 모델

그림 6-27 터미널 출력 결과

```
$python train_rnnlm.py
| epoch 1 |  iter 1 / 1327 | time 0[s] | perplexity 10000.84
| epoch 1 |  iter 21 / 1327 | time 4[s] | perplexity 3065.17
| epoch 1 |  iter 41 / 1327 | time 9[s] | perplexity 1255.96
| epoch 1 |  iter 61 / 1327 | time 14[s] | perplexity 956.13
| epoch 1 |  iter 81 / 1327 | time 18[s] | perplexity 806.56
| epoch 1 |  iter 101 / 1327 | time 23[s] | perplexity 658.86
| epoch 1 |  iter 121 / 1327 | time 27[s] | perplexity 636.88
| epoch 1 |  iter 141 / 1327 | time 31[s] | perplexity 601.75
| epoch 1 |  iter 161 / 1327 | time 35[s] | perplexity 575.78
| epoch 1 |  iter 181 / 1327 | time 40[s] | perplexity 590.01
| epoch 1 |  iter 201 / 1327 | time 44[s] | perplexity 479.95
| epoch 1 |  iter 221 / 1327 | time 48[s] | perplexity 488.23
| epoch 1 |  iter 241 / 1327 | time 53[s] | perplexity 443.62
| epoch 1 |  iter 261 / 1327 | time 57[s] | perplexity 468.75
| epoch 1 |  iter 281 / 1327 | time 61[s] | perplexity 447.81
| epoch 1 |  iter 301 / 1327 | time 66[s] | perplexity 398.51
| epoch 1 |  iter 321 / 1327 | time 70[s] | perplexity 350.89
| epoch 1 |  iter 341 / 1327 | time 74[s] | perplexity 406.82
| epoch 1 |  iter 361 / 1327 | time 79[s] | perplexity 409.33
| epoch 1 |  iter 381 / 1327 | time 83[s] | perplexity 337.04
```

그림 6-28 퍼플렉서티 추이(훈련 데이터를 대상으로 20번째 반복마다 평가한 결과)



LSTM을 사용한 언어 모델

Rank	Model	Test perplexity	Validation perplexity	Params	Extra Training Data	Paper	Code	Result	Year	Tags
1	**GPT-3** (Zero-Shot)	20.5		175000M	✓	Language Models are Few-Shot Learners			2020	
2	**BERT-Large-CAS**	31.3	36.1	395M	✓	Language Models with Transformers			2019	
3	**GPT-2**	35.76		1542M	✓	Language Models are Unsupervised Multitask Learners			2019	
4	**Mogrifier LSTM + dynamic eval**	44.9	44.8	24M	×	Mogrifier LSTM			2019	LSTM
5	**adversarial + AWD-LSTM-MoS + dynamic eval**	46.01	46.63	22M	×	Improving Neural Language Modeling via Adversarial Training			2019	LSTM
6	**GL-LWGC + AWD-MoS-LSTM + dynamic eval**	46.34	46.64	26M	×	Gradual Learning of Recurrent Neural Networks			2018	LSTM
7	**FRAGE + AWD-LSTM-MoS + dynamic eval**	46.54	47.38	22M	×	FRAGE: Frequency-Agnostic Word Representation			2018	LSTM
8	**AWD-LSTM-DOC x5**	47.17	48.63	185M	×	Direct Output Connection for a High-Rank Language Model			2018	LSTM
9	**Past Decode Reg. + AWD-LSTM-MoS + dyn. eval.**	47.3	48.0	22M	×	Improved Language Modeling by Decoding the Past			2018	LSTM

출처 : https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word

05

RNNLM 추가 개선

RNNLM 추가 개선

1. LSTM 계층 다층화 : 층이 증가할 수록 모델 표현력이 좋아짐
2. 드롭아웃 : 과적합 억제
3. 가중치 공유 : 두 계층이 가중치를 공유함으로써 학습하는 매개변수 수가 크게 줄어 및 정확도 향상

그림 6-29 LSTM 계층을 2층으로 쌓은 RNNLM

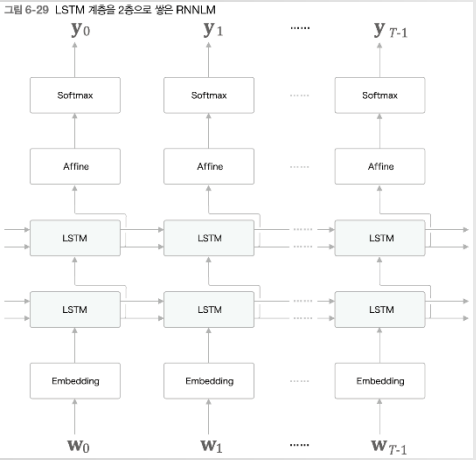


그림 6-33 좋은 예: 드롭아웃 계층을 깊이 방향(상하 방향)으로 삽입

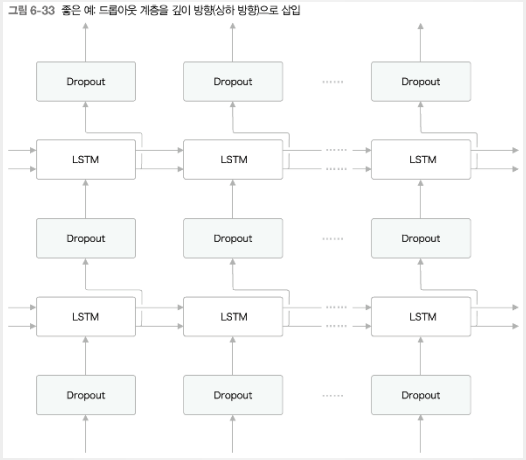
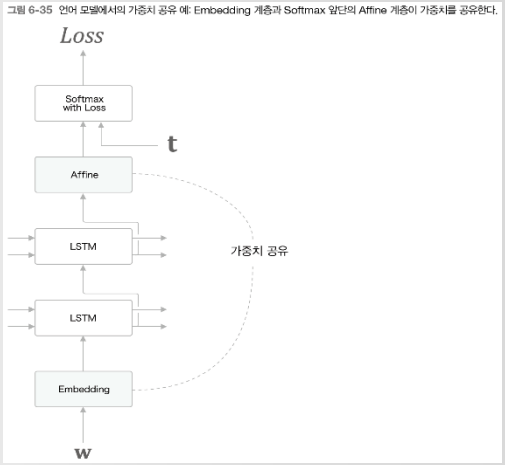


그림 6-35 언어 모델에서의 가중치 공유 예: Embedding 계층과 Softmax 앞단의 Affine 계층이 가중치를 공유한다.



RNNLM 추가 개선
드롭아웃

그림 6-32 나쁜 예: 드롭아웃 계층을 시계열 방향으로 삽입

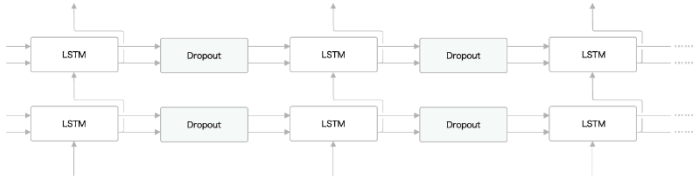


그림 6-33 좋은 예: 드롭아웃 계층을 깊이 방향(상하 방향)으로 삽입

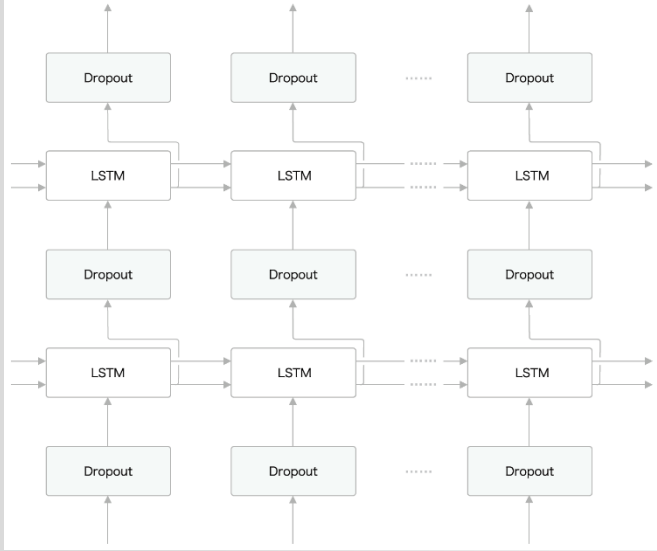
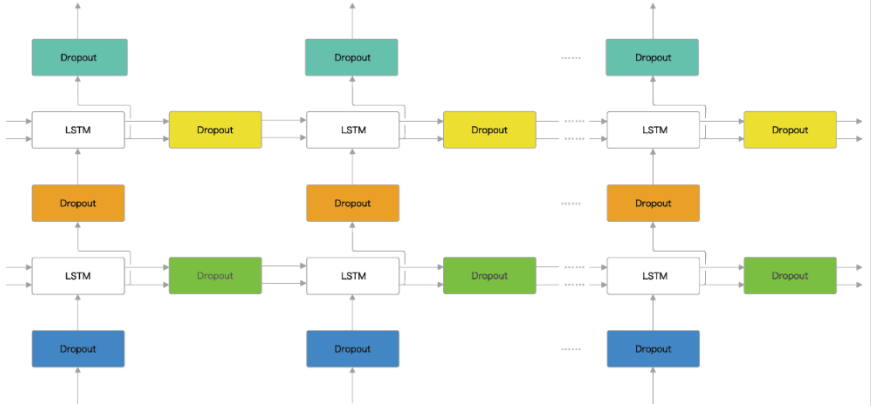


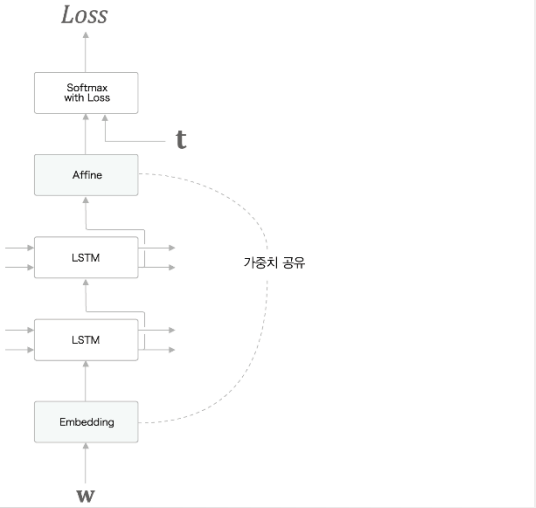
그림 6-34 변형 드롭아웃의 예: 색이 같은 드롭아웃끼리는 같은 마스크를 이용한다. 이처럼 같은 계층에 적용되는 드롭아웃끼리는 공통의 마스크를 이용함으로써 시간 방향 드롭아웃도 효과적으로 작동할 수 있다.

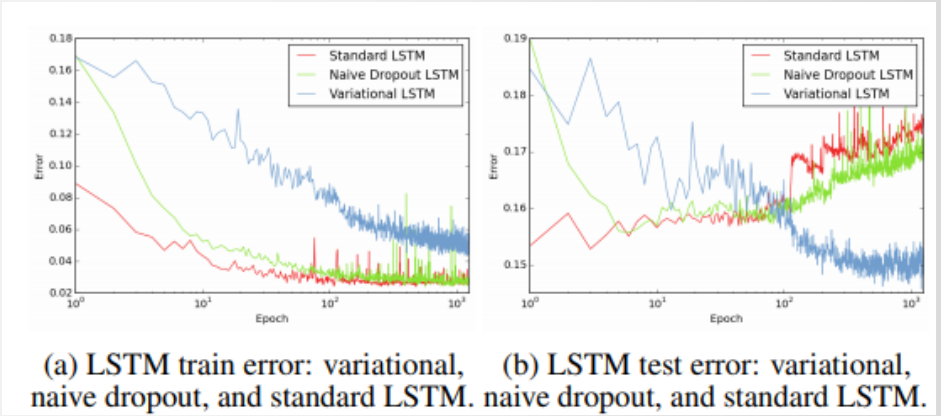


참고: A theoretically grounded application of dropout in recurrent neural networks

RNNLM 추가 개선
가중치 공유

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





(a) LSTM train error: variational, naive dropout, and standard LSTM.

(b) LSTM test error: variational, naive dropout, and standard LSTM.

참고: A theoretically grounded application of dropout in recurrent neural networks

개선된 RNNLM 구현

```
self.layers = [
    TimeEmbedding(embed_W),
    TimeDropout(dropout_ratio),
    TimeLSTM(lstm_Wx1, lstm_Wh1, lstm_b1, stateful=True),
    TimeDropout(dropout_ratio),
    TimeLSTM(lstm_Wx2, lstm_Wh2, lstm_b2, stateful=True),
    TimeDropout(dropout_ratio),
    TimeAffine(embed_W.T, affine_b)  # weight tying!!
]
```

04

정 리

GRU

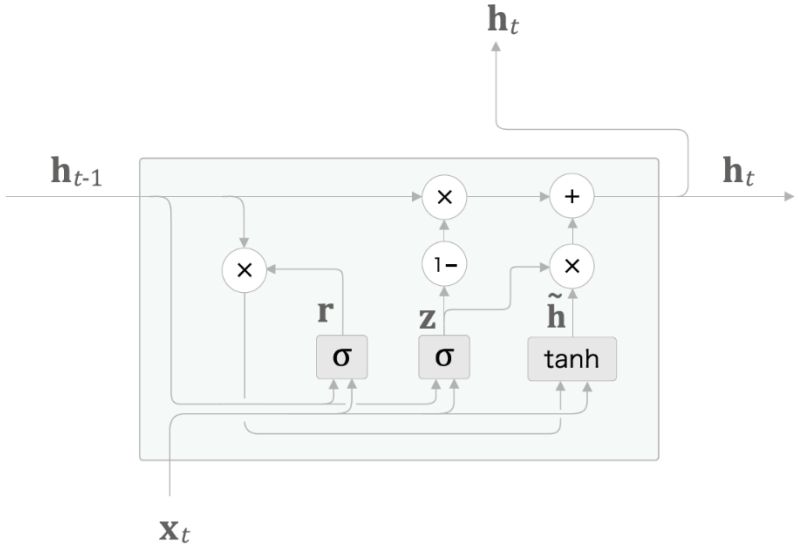
$$\mathbf{z} = \sigma(\mathbf{x}_t \mathbf{W}_{\mathbf{x}}^{(\mathbf{z})} + \mathbf{h}_{t-1} \mathbf{W}_{\mathbf{h}}^{(\mathbf{z})} + \mathbf{b}^{(\mathbf{z})})$$

$$\mathbf{r} = \sigma(\mathbf{x}_t \mathbf{W}_{\mathbf{x}}^{(\mathbf{r})} + \mathbf{h}_{t-1} \mathbf{W}_{\mathbf{h}}^{(\mathbf{r})} + \mathbf{b}^{(\mathbf{r})})$$

$$\tilde{\mathbf{h}} = \tanh(\mathbf{x}_t \mathbf{W}_{\mathbf{x}} + (\mathbf{r} \odot \mathbf{h}_{t-1}) \mathbf{W}_{\mathbf{h}} + \mathbf{b})$$

$$\mathbf{h}_t = (1 - \mathbf{z}) \odot \mathbf{h}_{t-1} + \mathbf{z} \odot \tilde{\mathbf{h}}$$

그림 C-2 GRU의 계산 그래프: 'σ' 노드와 'tanh' 노드에는 전용 가중치가 있고, 노드 내부에서 아핀 변환을 수행한다 ('1−' 노드는 x를 입력하면 $1-x$를 출력한다).



RNN 학습 문제점 : 기울기 소실, 폭발이 있음

LSTM은 output, forget, input 3개의 게이트로 이루어짐

게이트에는 전용 가중치가 있으며, 시그모이드 함수를 사용한다.

언어 모델 개선에는 LSTM 다층화, 드롭아웃, 가중치 공유 등의 기법이 효과적

THE END

감 사 합 니 다