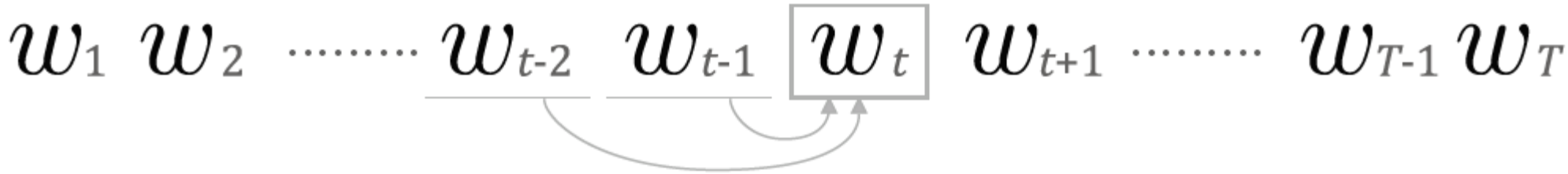
밑바닥부터
시작하는 딥러닝2

5장. 순환신경망(RNN)

송선영

CONTENTS

- CBOW 복습



- 사후확률 $P(w_t \mid w_{t-2},\ w_{t-1})$

- 손실함수 $L = -\log P(w_t \mid w_{t-2},\ w_{t-1})$

- 동시확률 : 사후확률의 총곱

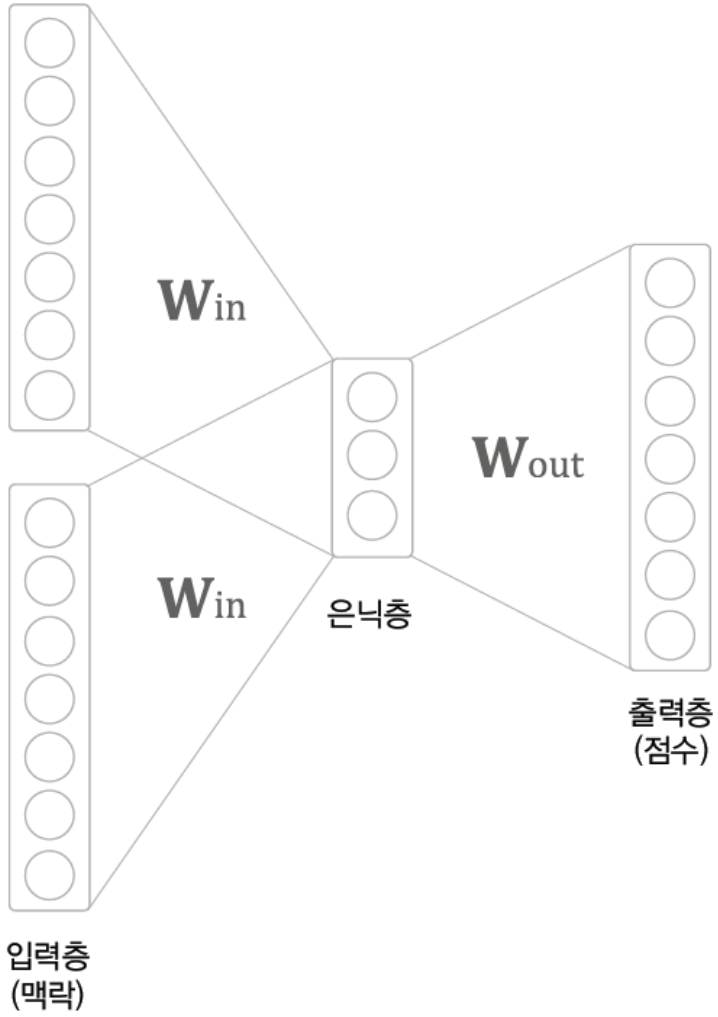
사후확률

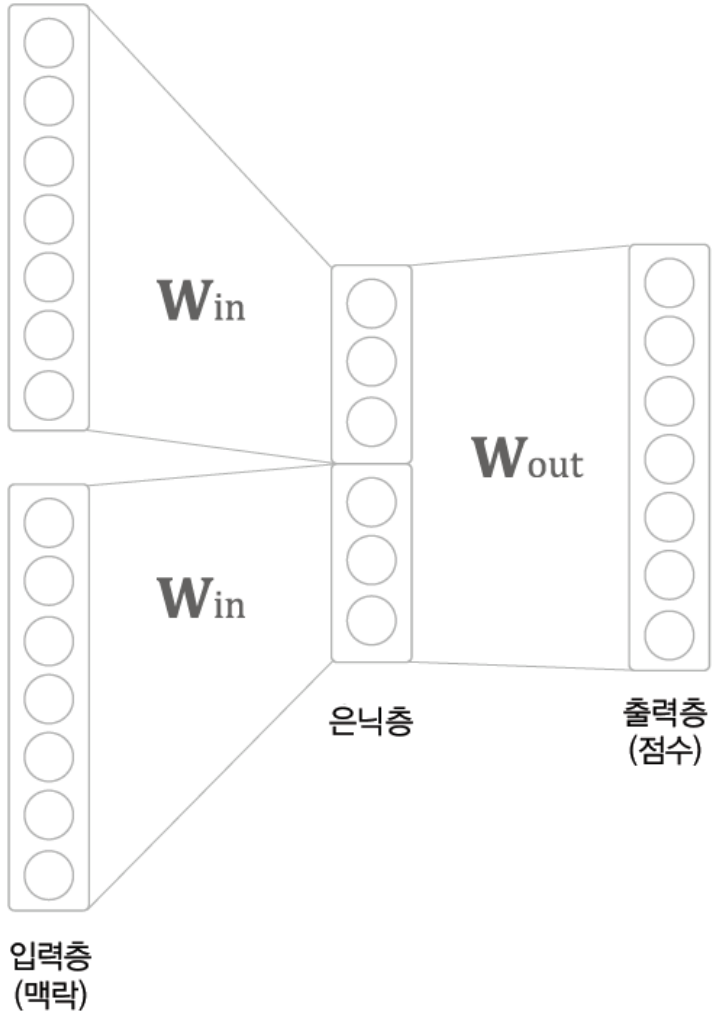
$$P(w_1,\cdots,w_m) = P(w_m \mid w_1,\cdots,w_{m-1})P(w_{m-1} \mid w_1,\cdots,w_{m-2})$$
$$\cdots P(w_3 \mid w_1,w_2)P(w_2 \mid w_1)P(w_1)$$
$$= \prod_{t=1}^{m} P(w_t \mid w_1,\cdots,w_{t-1})$$

동시확률

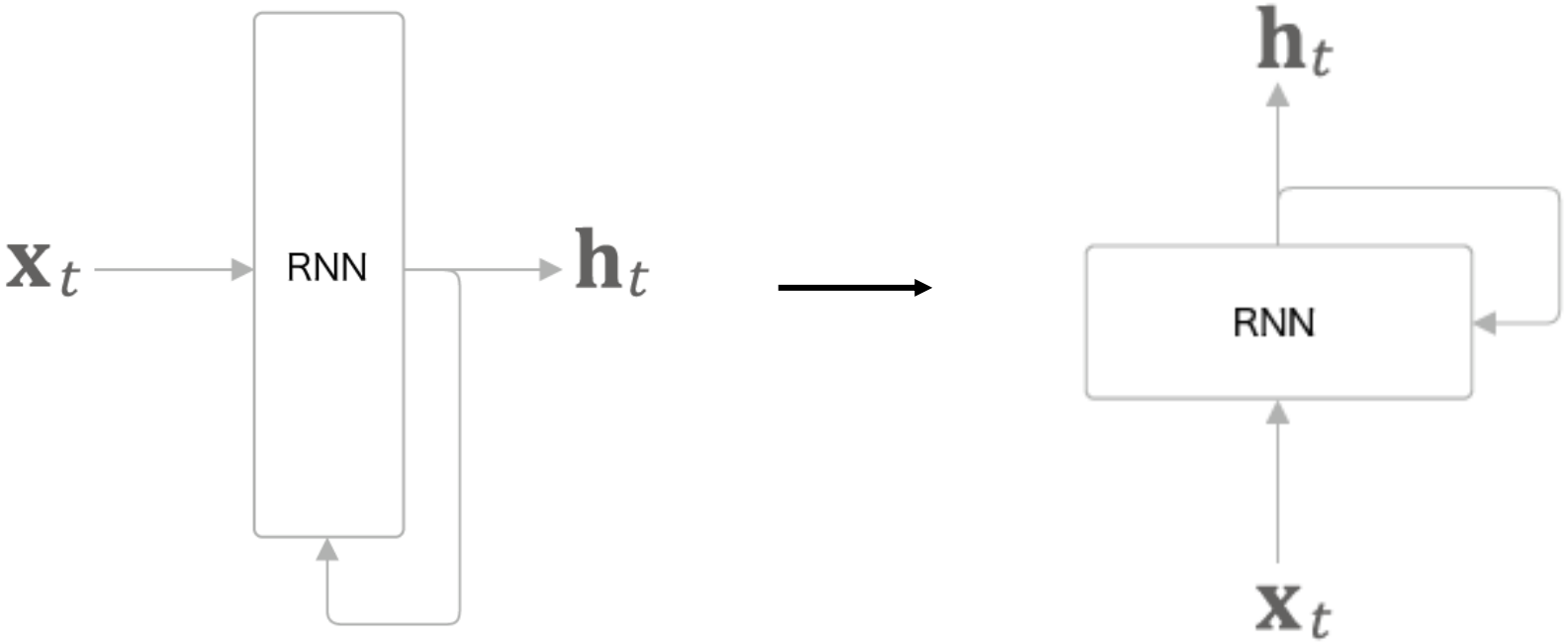
- CBOW 모델을 언어 모델로 사용하기

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

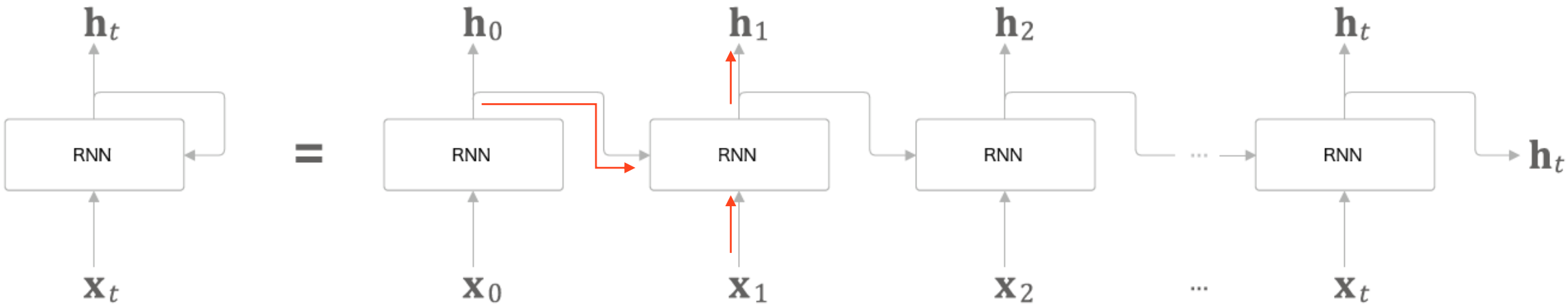

Win
Win
Wout
은닉층
출력층
(점수)
입력층
(맥락)


Win
Win
Wout
은닉층
출력층
(점수)
입력층
(맥락)

- RNN (Recurrent Neural Network) : 순환신경망
  - **닫힌 경로** 혹은 **순환하는 경로**를 따라 데이터가 끊임없이 순환한다.
  - 이를 통해 과거의 정보를 기억하는 동시에 최신 데이터로 갱신된다.



입력 : $X_0, X_1, \ldots, X_t, \ldots$
출력 : $h_0, h_1, \ldots, h_t, \ldots$



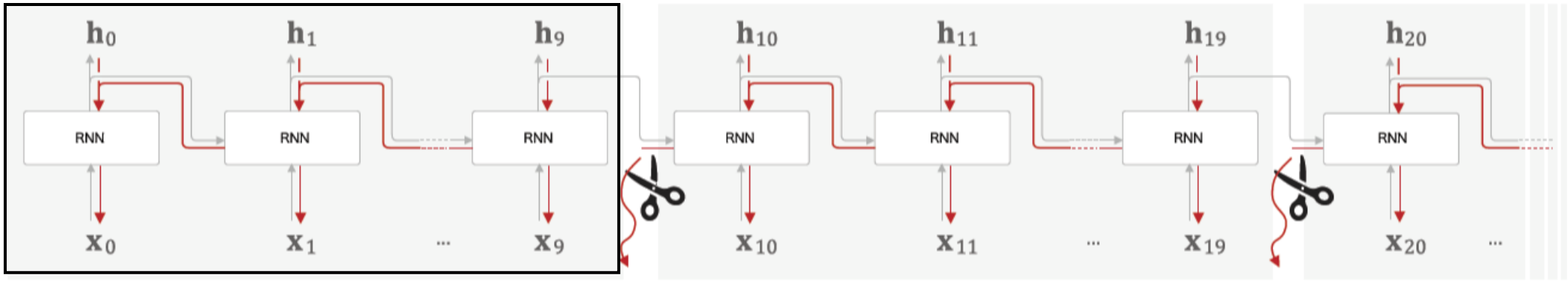
$$\mathbf{h}_t = \tanh(\mathbf{h}_{t-1}\mathbf{W}_\mathbf{h} + \mathbf{x}_t\mathbf{W}_\mathbf{x} + \mathbf{b})$$

h_t: 시각 t의 출력 (은닉상태 벡터)
h_{t-1} : 시각 t-1의 출력
b : 편향

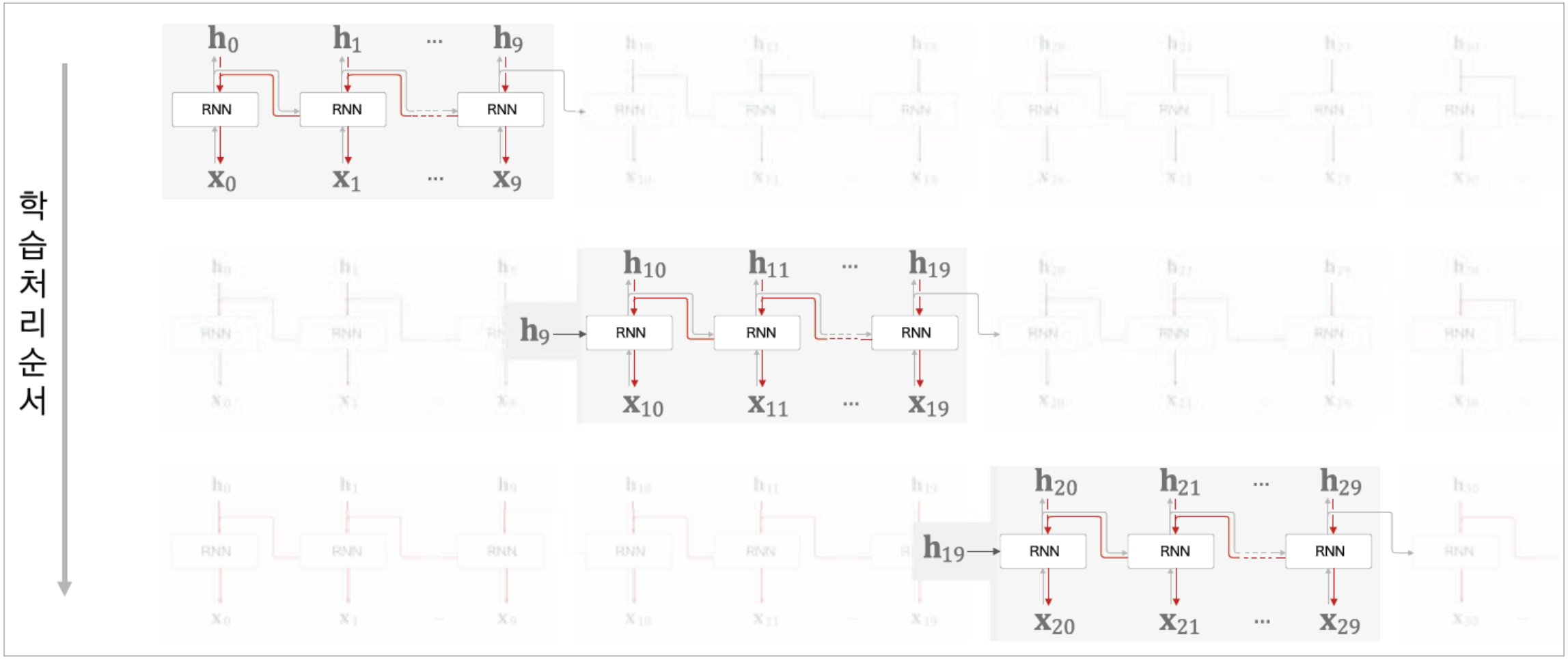
W_h : 1개 전의 RNN 출력을 다음 시각의 출력으로 변환하기 위한 가중치
W_x : 입력 x를 출력 h로 변환하기 위한 가중치

- BPTT (Backpropagation Through Time)
  - 시간 방향으로 펼친 신경망의 오차역전파법
  - 시계열 데이터의 시간 크기가 커질수록 메모리 사용량이 증가하고 역전파 시의 기울기가 불안정해진다.

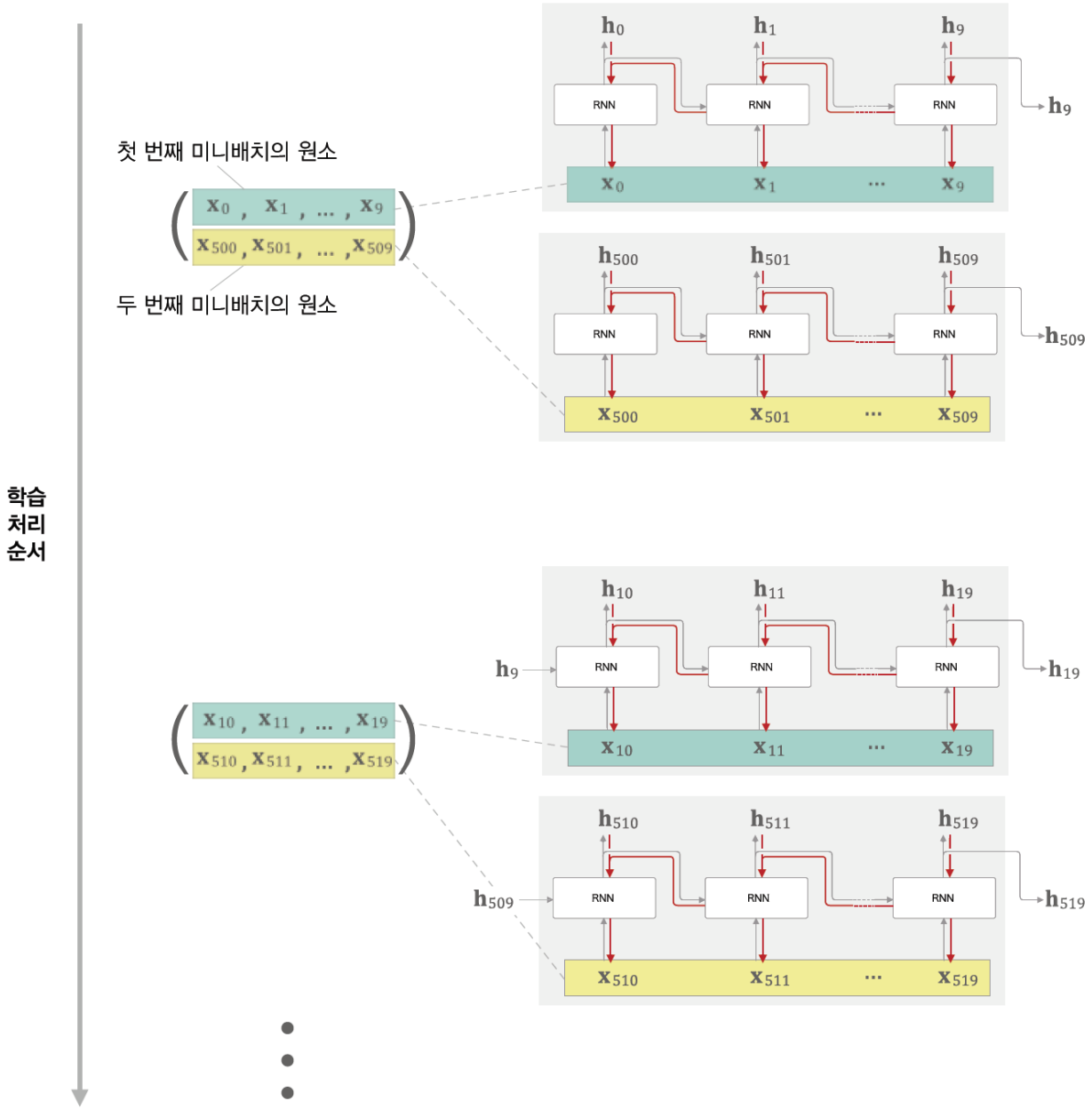
- Truncated BPTT
  - 너무 길어진 신경망을 적당한 지점에서 잘라내어 작은 신경망 여러 개로 만든다.
  - 순전파의 연결은 유지하고, 역전파의 연결만 끊는다.



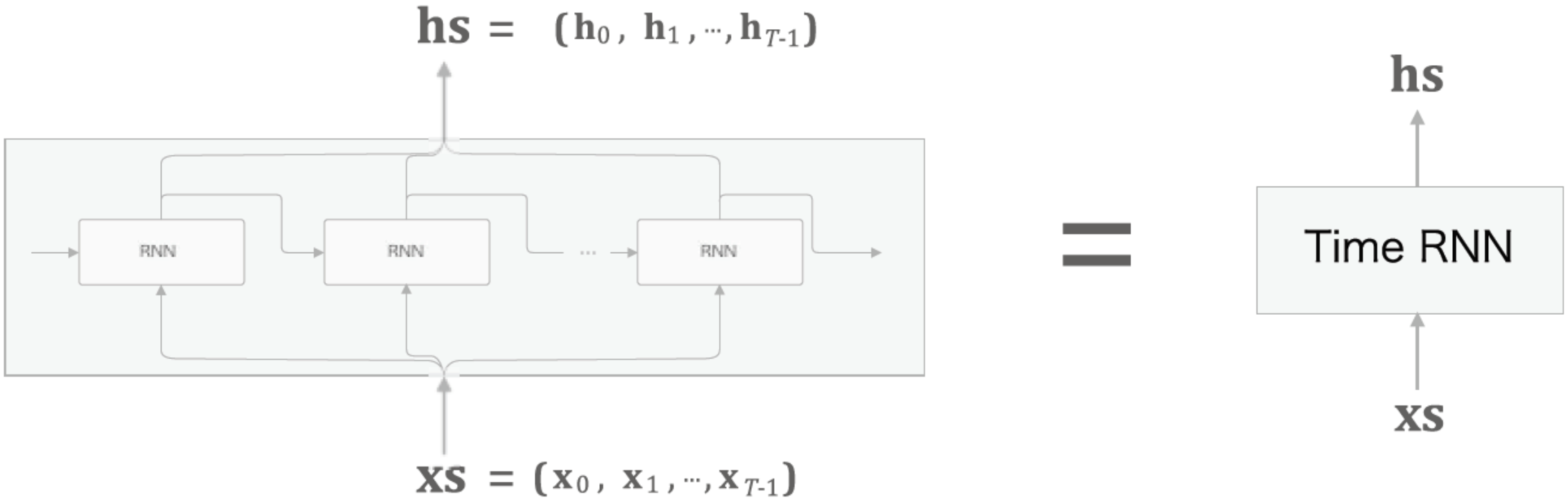
- Truncated BPTT 의 데이터 처리 과정



- Truncated BPTT 의 미니배치 학습



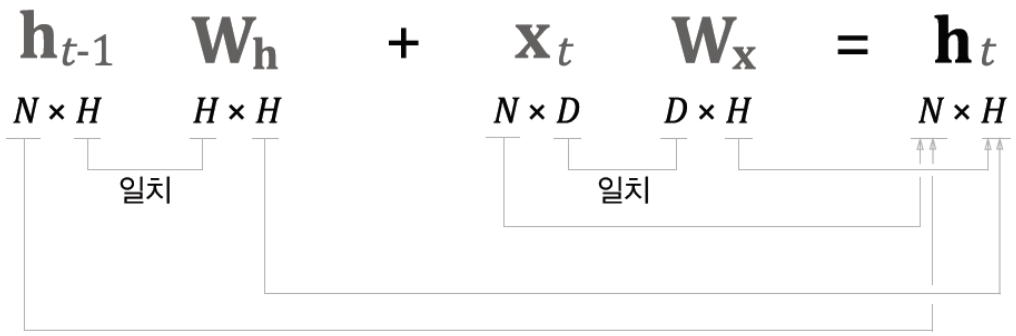
- Time RNN 계층



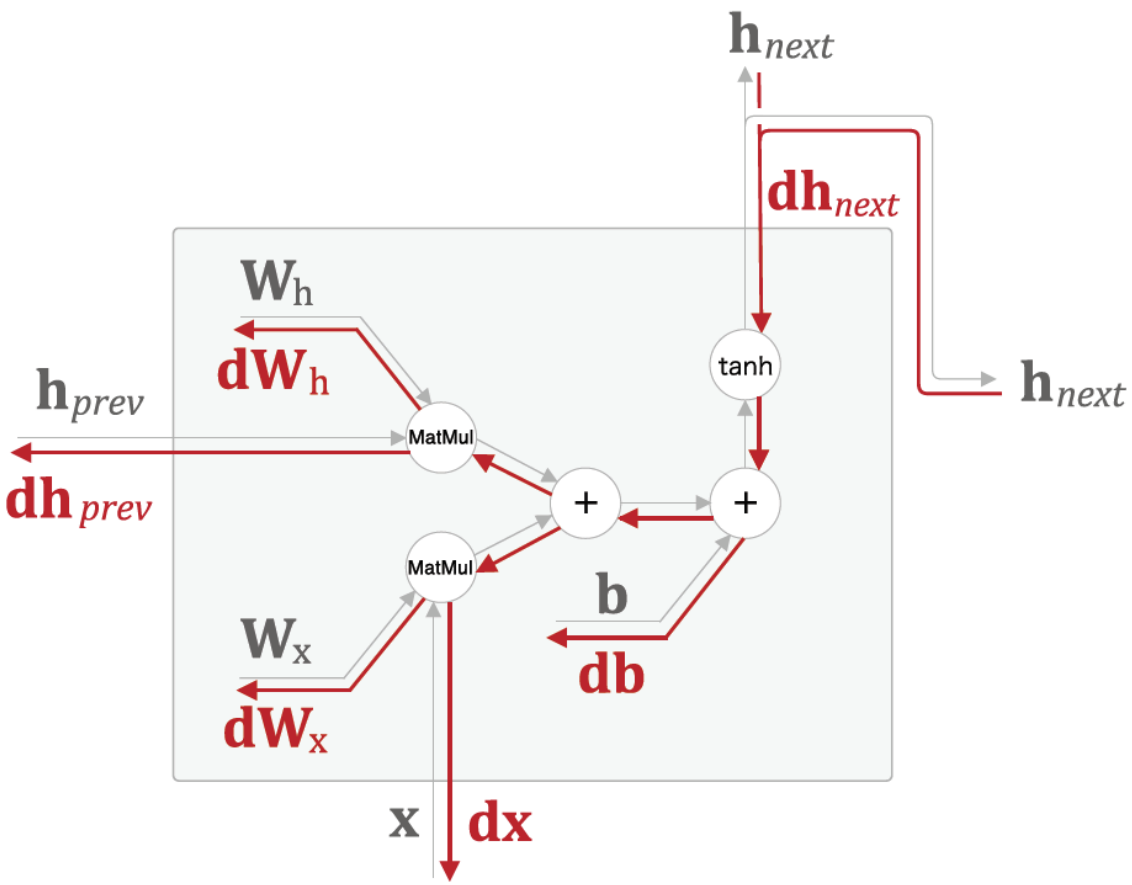
- RNN 계층 구현

```
class RNN:
  def __init__(self, Wx, Wh, b):
    self.params = [Wx, Wh, b]
    #가중치 2개, 편향 1개
    self.grads = [np.zeros_like(Wx), np.zeros_like(Wh), np.zeros_like(b)]
    #각 매개변수에 대응하는 형태로 기울기를 초기화한 후 grads에 저장
    self.cache = None
    #역전파 계산 시 사용하는 중간 데이터를 담은 cache를 None으로 초기화

  def forward(self, x, h_prev):
    #아래로부터의 입력 x와 왼쪽으로부터의 입력 h_prev
    Wx, Wh, b = self.params
    t = np.matmul(h_prev, Wh) + np.matmul(x, Wx) + b
    #matmul: 행렬의 곱
    h_next = np.tanh(t)

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

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



N : 미니배치 크기
D : 입력 벡터의 차원 수
H : 은닉 상태 벡터의 차원 수

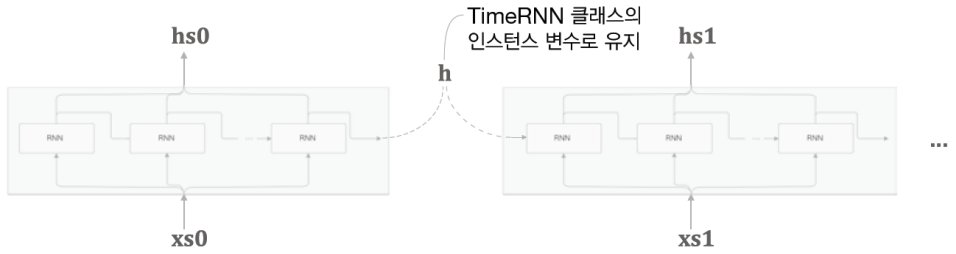



```python
#RNN계층의 역전파 메서드 구현
def backward(self, dh_next):
    Wx, Wh, b = self.params
    x, h_prev, h_next = self.cache

    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
```

- Time RNN 계층 구현

```
class TimeRNN:
  def __init__(self, Wx, Wh, b, stateful=false):
    #초기화 메서드는 가중치, 편향, stateful이라는 boolean값을 인수로 받음
    self.params = [Wx, Wh, b]
    self.grads = [np.zeors_like(Wx), np.zeros_like(Wh), np.zeros_like(b)]
    self.layers = None
    #layers : 다수의 RNN계층을 리스트로 저장하는 용도

    self.h, self.dh = None, None
    #h: forward() 메서드를 불렀을 때 마지막 RNN 계층의 은닉 상태를 저장
    #dh: backward()를 불렀을 때 하나 앞 블록의 은닉 상태의 기울기를 저장
    self.stateful = stateful

  def set_state(self, h):
    #Time RNN계층의 은닉 상태를 설정하는 메서드
    self.h = h

  def reset_state(self):
    #은닉 상태를 초기화하는 메서드
    self.h = None
```

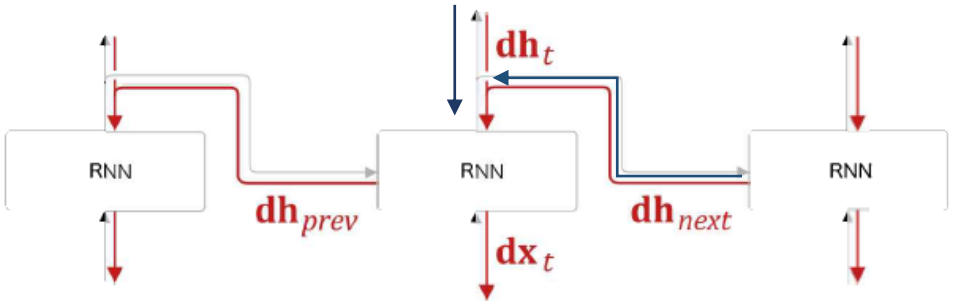
```
#순전파 구현
def forward(self, xs):
  #아래로부터 입력 xs(T개 분량의 시계열 데이터를 하나로 모은 것)를 받는다.
  Wx, Wh, b = self.params
  N, T, D = xs.shape #N: 미니배치 크기, D: 입력 벡터의 차원 수
  D, H = Wx.shape

  self.layers = []
  hs = np.empty((N, T, H), dtype='f')
  #출력값을 담을 그릇 hs를 준비한다.

  if not self.stateful or self.h is None:  #첫 호출시
    self.h = np.zeros((N, H), dtype='f')
    #h: RNN 계층의 은닉 상태.
    #self.h=None: 처음 호출 시에는 원소가 모두 0인 영행렬로 초기화됨.
    #stateful=False: 항상 영행렬로 초기화

  for t in range(T):
    layer = RNN(*self.params)
    # *: 리스트의 원소들을 추출하여 메서드의 인수로 전달
    #self.params에 들어 있는 Wx, Wh, b를 추출하여 RNN 클래스의 __init__()메서드에 전달
    #RNN계층을 생성하여 인스턴스 변수 layers에 추가한다.
    self.h = layer.forward(xs[:, t, :], self.h)
    hs[:, t, :] = self.h
    self.layers.append(layer)

  return hs
```



stateful=True : Time RNN계층이 은닉 상태를 유지한다.
-> 아무리 긴 시계열 데이터라도 Time RNN계층의 순전파를 끊지 않고 전파한다.
stateful=False: Time RNN 계층은 은닉 상태를 '영행렬'로 초기화한다.상태가 없다.

- Time RNN 계층 구현

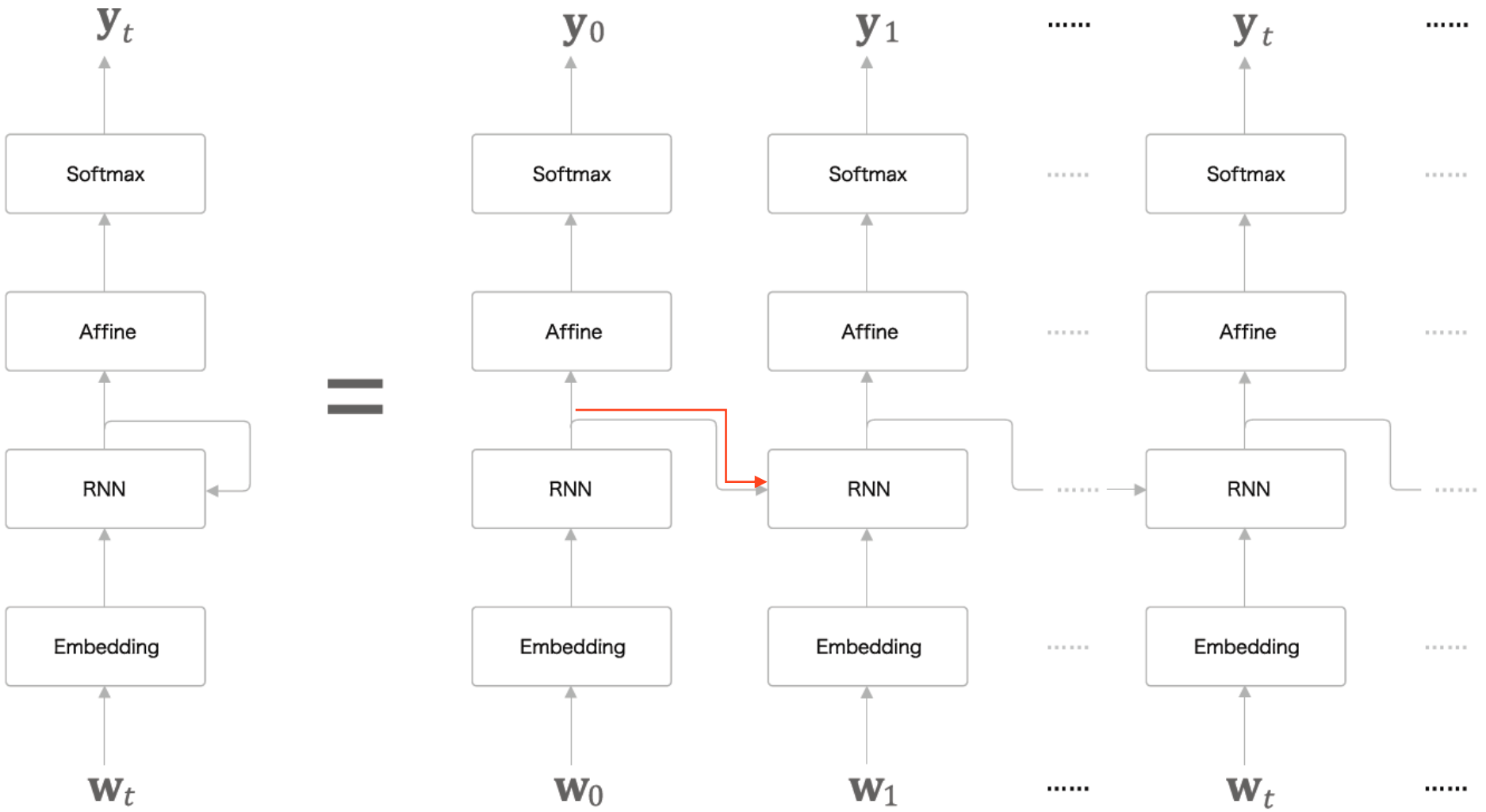
```
#역전파 구현
def backward(self, dhs):
  Wx, Wh, b = self.params
  N, T, H = dhs.shape
  D, H = Wx.shape

  dxs = np.empty((N, T, D), dtype='f')
  dh = 0
  grads = [0, 0, 0]
  for t in reversed(range(T)):
    layer = self.layers[t]
    dx, dh = layer.backward(dhs[:, t, :] + dh) #합산된 기울기
    #RNN계층의 순전파에서는 출력이 2개로 분기되는데 역전파에서는
    #각 기울기가 합산되어 전해짐
    dxs[:, t, :] = dx

    for i, grad in enumerate(layer.grads):
      grads[i] +=grad          # 가중치 기울기를 모두 합산

  for i, grad in enumerate(grads):    # 모두 합산한 가중치 기울기를
    self.grads[i][...] = grad         # self.grads에 덮어씀
  self.dh = dh

  return dxs
```



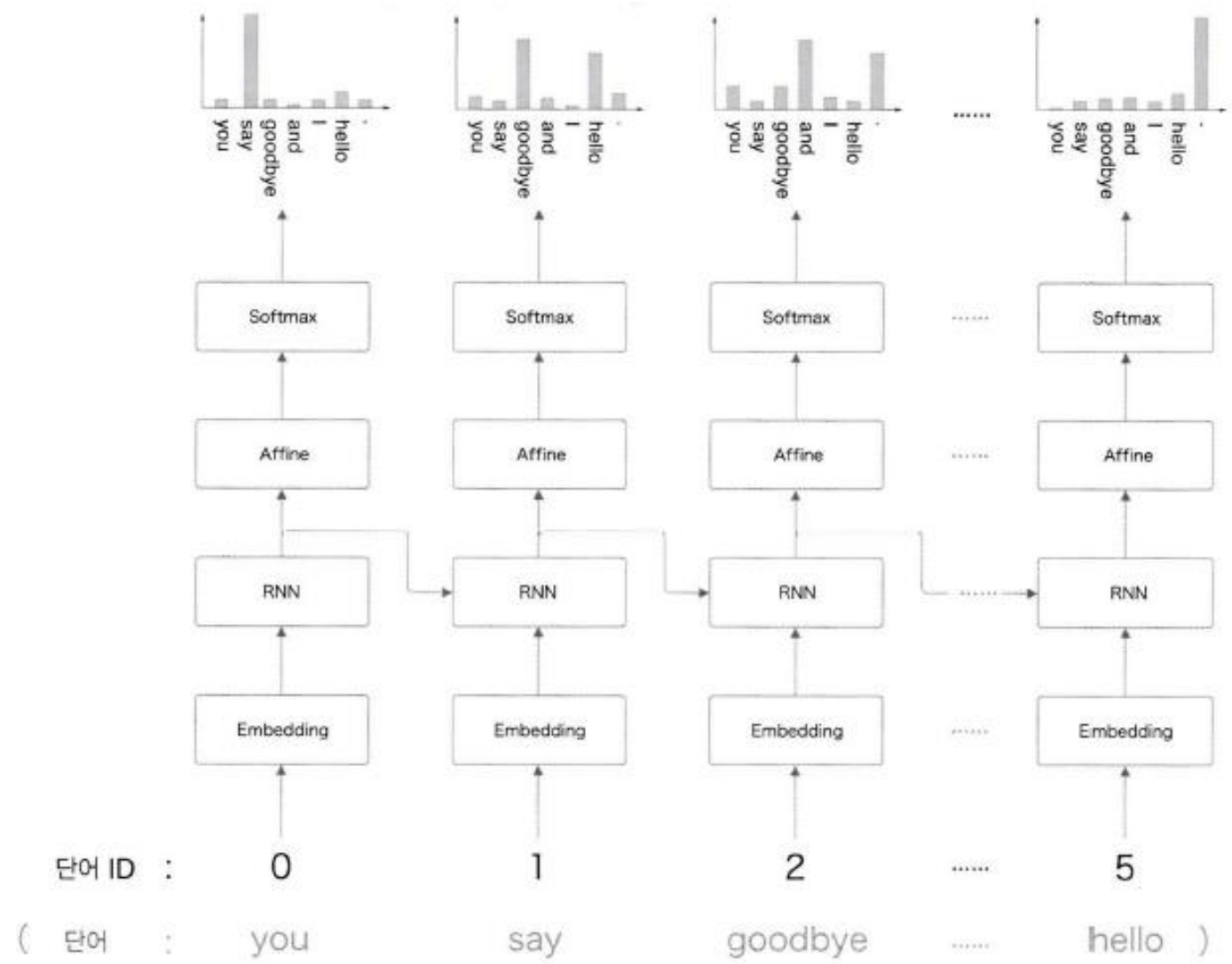
dh_t : t번째 RNN 계층에서 위로부터의 기울기
dh_{next} : 다음 계층으로부터의 기울기

dhs : 상류(출력 쪽 층)에서부터 전해지는 기울기
dxs : 하류(입력 쪽 층)로 내보내는 기울기

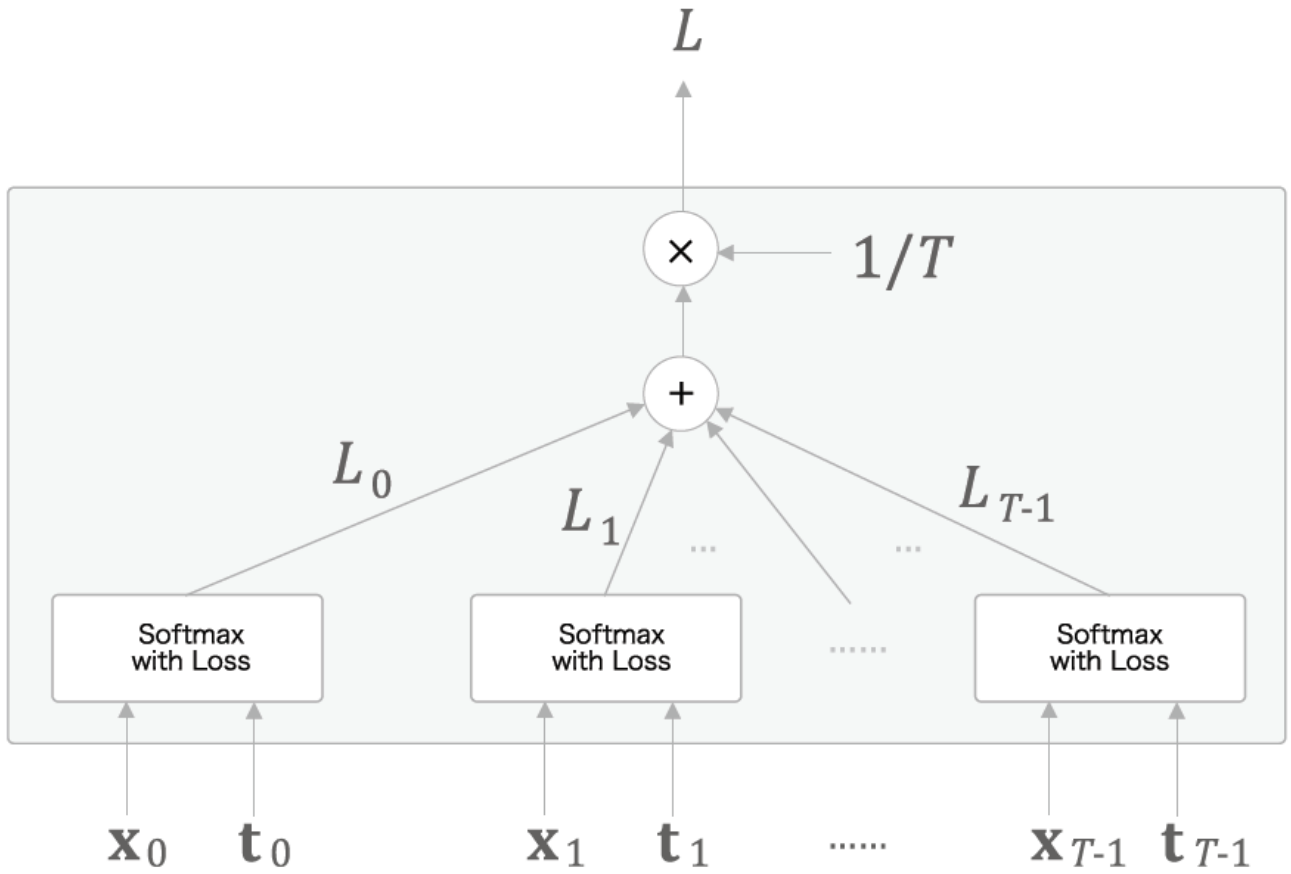
- RNNLM (RNN Language Model) : RNN을 사용한 언어 모델



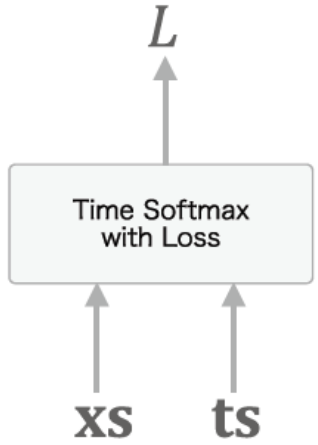
- RNNLM (RNN Language Model) : RNN을 사용한 언어 모델



$X_0, X_1, \ldots$: 아래층에서부터 전해지는 점수(확률로 정규화되기 전의 값)
$t_0, t_1, \ldots$: 정답 레이블



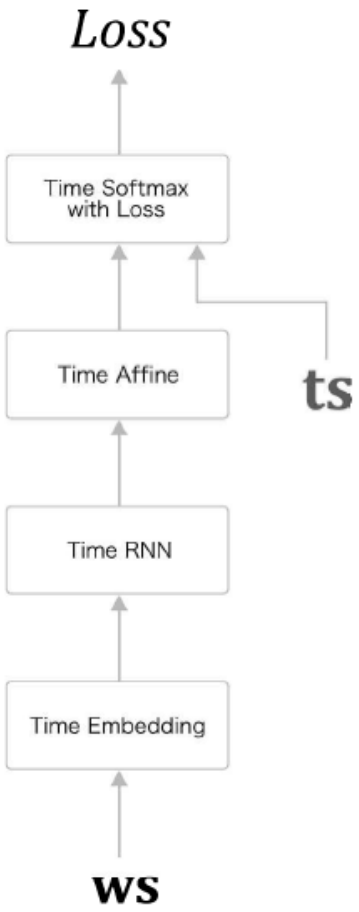
=



$$L = \frac{1}{T}(L_0 + L_1 + \cdots + L_{T-1})$$

05 RNNLM 학습과 평가

- SimpleRnnlm 클래스 구현



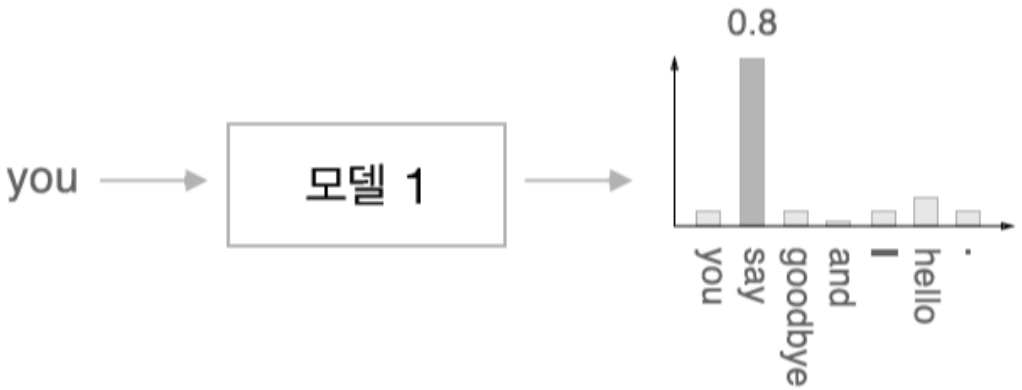

```python
import sys
sys.path.append('..')
import numpy as np
from common.time_layers import *

class SimpleRnnlm:
    def __init__(self, vocab_sizse, wordvec_size, hidden_size):
      V, D, H = vocab_size, wordvec_size, hidden_size
      rn = np.random.randn

      #가중치 초기화
      embed_W = (rn(V, D) / 100).astype('f')
      rnn_Wx = (rn(D, H) / np.sqrt(D)).astype('f')
      rnn_Wh = (rn(H, H) / np.sqrt(H)).astype('f')
      rnn_b = np.zeros(H).astype('f')
      affine_W = (rn(H, V) / np.sqrt(H)), astype('f')
      affine_b = np.zeros(V).astype('f')


    #계층 생성
      self.layers = [
        TimeEmbedding(embed_W),
        TimeRNN(rnn_Wx, rnn_Wh, rnn_b, stateful=True),
        #Truncated BPTT로 학습한다고 가정하여 Time RNN계층의 stateful=True로 설정
        # -> TimeRNN계층은 이전 시간의 은닉 상태를 계승할 수 있음
        TimeAffine(affine_W, affine_b)
      ]
      self.loss_layer = TimeSoftmaxWithLoss()
      self.rnn_layer = self.layers[1]

      #모든 가중치와 기울기를 리스트에 모음
      self.params, self.grads = [],[]
      for layer in self.layers:
        self.params += layer.params
        self.grads += layer.grads
```

-> SimpleRnnlm클래스는 4개의 Time계층을 쌓은 신경망

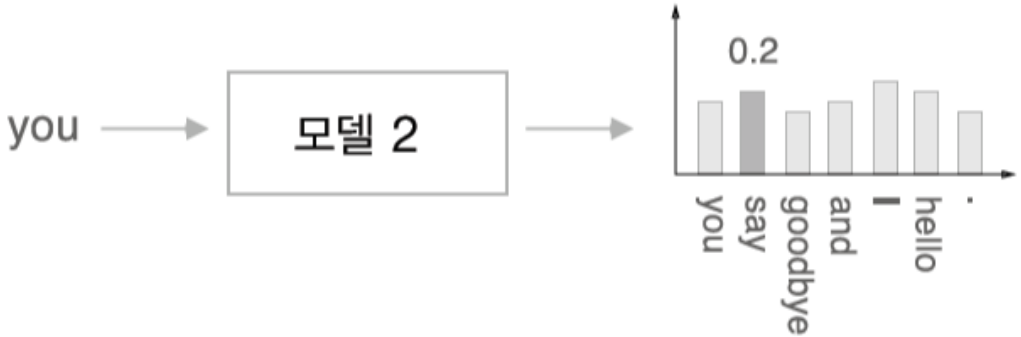
-> RNN 계층과 Affine계층에서 'Xabier초깃값'을 이용

```python
def forward(self, xs, ts):
  for layer in self.layers:
    xs = layer.forward(xs)
  loss = self.loss_layer.forward(xs, ts)
  return loss

def backward(self, dout=1):
  dout = self.loss_layer.backward(dout)
  for layer in reversed(self.layers):
    dout = layer.backward(dout)
  return dout

def reset_state(self):
  self.rnn_layer.reset_state()
```

- 언어모델의 평가 방법



-> Perplexity = 1/(0.8) = 1.25



-> Perplexity = 1/(0.2) = 5

$$L = -\frac{1}{N}\sum_{n}\sum_{k} t_{nk}\log y_{nk}$$

$$\text{perplexity} = e^{L}$$

N : 데이터의 총 개수
t_n : 원핫 벡터로 나타낸 정답 레이블
t_{nk} : n개 째 데이터의 k번째 값
y_{nk} : 확률분포

* Perplexity가 작을수록 좋음

05 RNNLM 학습과 평가

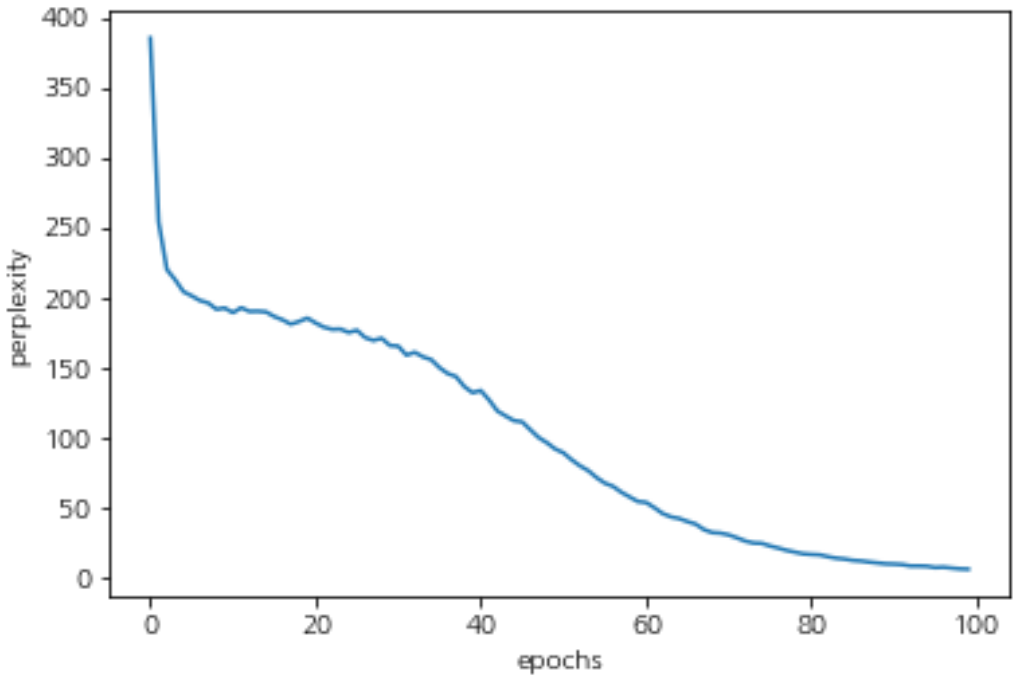
- RNNLM 학습 코드 구현

```
import sys
sys.path.append('..')
import matplotlib.pyplot as plt
import numpy as np
from common.optimizer import SGD
from dataset import ptb
from simple_rnnlm import SimpleRnnlm

# 하이퍼파라미터 설정
batch_size = 10
wordvec_size = 100
hidden_size = 100 # RNN의 은닉 상태 벡터의 원소 수
time_size = 5     # Truncated BPTT가 한 번에 펼치는 시간 크기
lr = 0.1
max_epoch = 100

# 학습 데이터 읽기(전체 중 1000개만)
corpus, word_to_id, id_to_word = ptb.load_data('train')
corpus_size = 1000
corpus = corpus[:corpus_size]
vocab_size = int(max(corpus) + 1)

xs = corpus[:-1]  # 입력
ts = corpus[1:]   # 출력(정답 레이블)
data_size = len(xs)
print('말뭉치 크기: %d, 어휘 수: %d' % (corpus_size, vocab_size))

# 학습 시 사용하는 변수
max_iters = data_size // (batch_size * time_size)
time_idx = 0
total_loss = 0
loss_count = 0
ppl_list = []

# 모델 생성
model = SimpleRnnlm(vocab_size, wordvec_size, hidden_size)
optimizer = SGD(lr)
```

```
# 미니배치의 각 샘플의 읽기 시작 위치를 계산
jump = (corpus_size - 1) // batch_size
offsets = [i * jump for i in range(batch_size)]
# 각 미니배치가 데이터를 읽기 시작하는 위치를 계산해 offsets에 저장
# offsets의 각 원소에 데이터를 읽는 시작 위치가 담김

for epoch in range(max_epoch):
    for iter in range(max_iters):
        # 미니배치 취득
        batch_x = np.empty((batch_size, time_size), dtype='i')
        batch_t = np.empty((batch_size, time_size), dtype='i')
        for t in range(time_size):
            for i, offset in enumerate(offsets):
                batch_x[i, t] = xs[(offset + time_idx) % data_size]
                batch_t[i, t] = ts[(offset + time_idx) % data_size]
                # 말뭉치를 읽는 위치가 말뭉치 크기를 넘어설 경우 말뭉치의 처음으로
                # 돌아오기 위해서 말뭉치의 크기로 나눈 나머지를 인덱스로 사용
            time_idx += 1
            # time_idx를 1씩 늘리면서 말뭉치에서 time_idx위치의 데이터를 얻음

        # 기울기를 구하여 매개변수 갱신
        loss = model.forward(batch_x, batch_t)
        model.backward()
        optimizer.update(model.params, model.grads)
        total_loss += loss
        loss_count += 1

    # 에폭마다 퍼플렉서티 평가
    ppl = np.exp(total_loss / loss_count)
    print('| 에폭 %d | 퍼플렉서티 %.2f'
          % (epoch+1, ppl))
    ppl_list.append(float(ppl))
    total_loss, loss_count = 0, 0
```

05 RNNLM 학습과 평가

```
# 그래프 그리기
x = np.arange(len(ppl_list))
plt.plot(x, ppl_list, label='train')
plt.xlabel('epochs')
plt.ylabel('perplexity')
plt.show()
```



- RnnlmTrainer 클래스 사용하여 학습하기

```
# 모델 생성
model = SimpleRnnlm(vocab_size, wordvec_size, hidden_size)
optimizer = SGD(lr)
trainer = RnnlmTrainer(model, optimizer)

# 학습 수행
trainer.fit(xs, ts, max_epoch, batch_size, time_size)
```

< 학습 과정 >
1 미니배치를 ' 순차적 ' 으로 만들어
2 모델의 순전파와 역전파를 호출하고
3 옵티마이저로 가중치를 갱신하고
4 퍼플렉시티를 구한다.

06 정리

1. RNN은 순환하는 경로가 있고, 이를 통해 내부에 '은닉 상태'를 기억할 수 있다.
2. RNN의 순환 경로를 펼침으로써 다수의 RNN 계층이 연결된 신경망으로 해석할 수 있으며, 보통의 오차역전법으로 학습할 수 있다. (=BPTT)
3. 긴 시계열 데이터를 학습할 때는 데이터를 적당한 길이씩 모으고(이를 '블록 ' 이라 한다), 블록 단위로 BPTT에 의한 학습을 수행한다 (=Truncated BPTT)
4. Truncated BPTT에서는 역전파의 연결만 끊는다.
5. Truncated BPTT에서는 순전파의 연결을 유지하기 위해 데이터를 '순차적 ' 으로 입력해야 한다.
6. 언어 모델은 단어 시퀀스를 확률로 해석한다.
7. RNN 계층을 이용한 조건부 언어 모델은 (이론적으로는) 그때까지 등장한 모든 단어의 정보를 기억할 수 있다.

끝