word2vec

1. 추론 기반 기법과 신경망

- 문맥이 주어졌을 때, 올 단어를 추론하는 기법

통계 기반 기법의 문제점

- 동시발생행렬을 만들고 그 행렬에 SVD를 적용하여 밀집벡터(단어의 분산 표현)를 얻음
- 하지만, 말뭉치의 어휘 수는 어마어마하고 SVD를 거기에 적용하는 것은 많은 계산시간이 소요됨($O(n^3)$)

추론 기반 기법과 통계 기반 기법의 비교

- 통계 기반 기법 : 학습데이터를 한꺼번에 처리(배치 학습)
- 추론 기반 기법 : 학습데이터를 일부를 사용하여 순차적을 학습(미니배치 학습)
  -> 크기가 큰 데이터/계산량이 큰 작업 또한 학습이 가능, 병렬 계산또한 가능

신경망에서의 단어 처리

- 데이터 준비
  - 단어를 원핫 벡터를 사용하여 '고정 길이의 벡터'로 변환하여 뉴런의 수를 고정

In [5]:

```
# 편향이 생략된 신경망
import numpy as np
c = np.array([[1,0,0,0,0,0,0]]) # 입력(하나의 벡터)
W = np.random.randn(7,3) # 가중치
h = np.matmul(c,W) # 중간 노드
print(h)
```

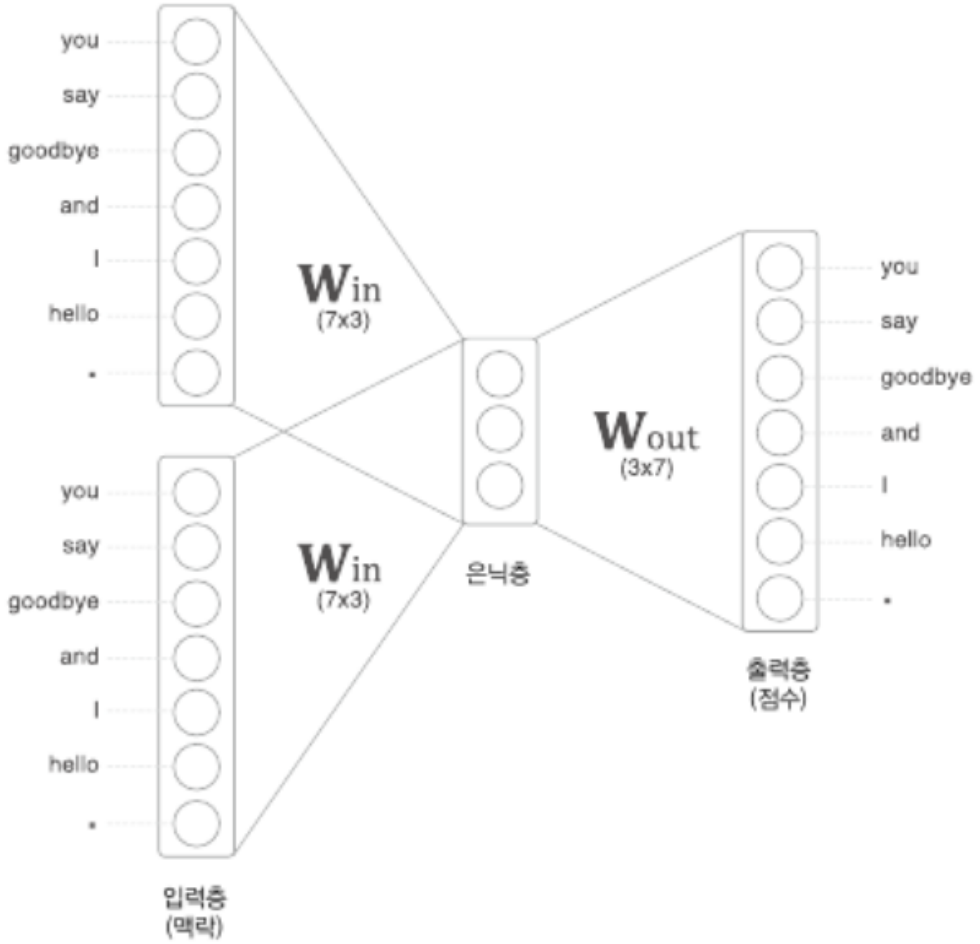
```
[[-0.90370696 -0.13447339  0.09305026]]
```

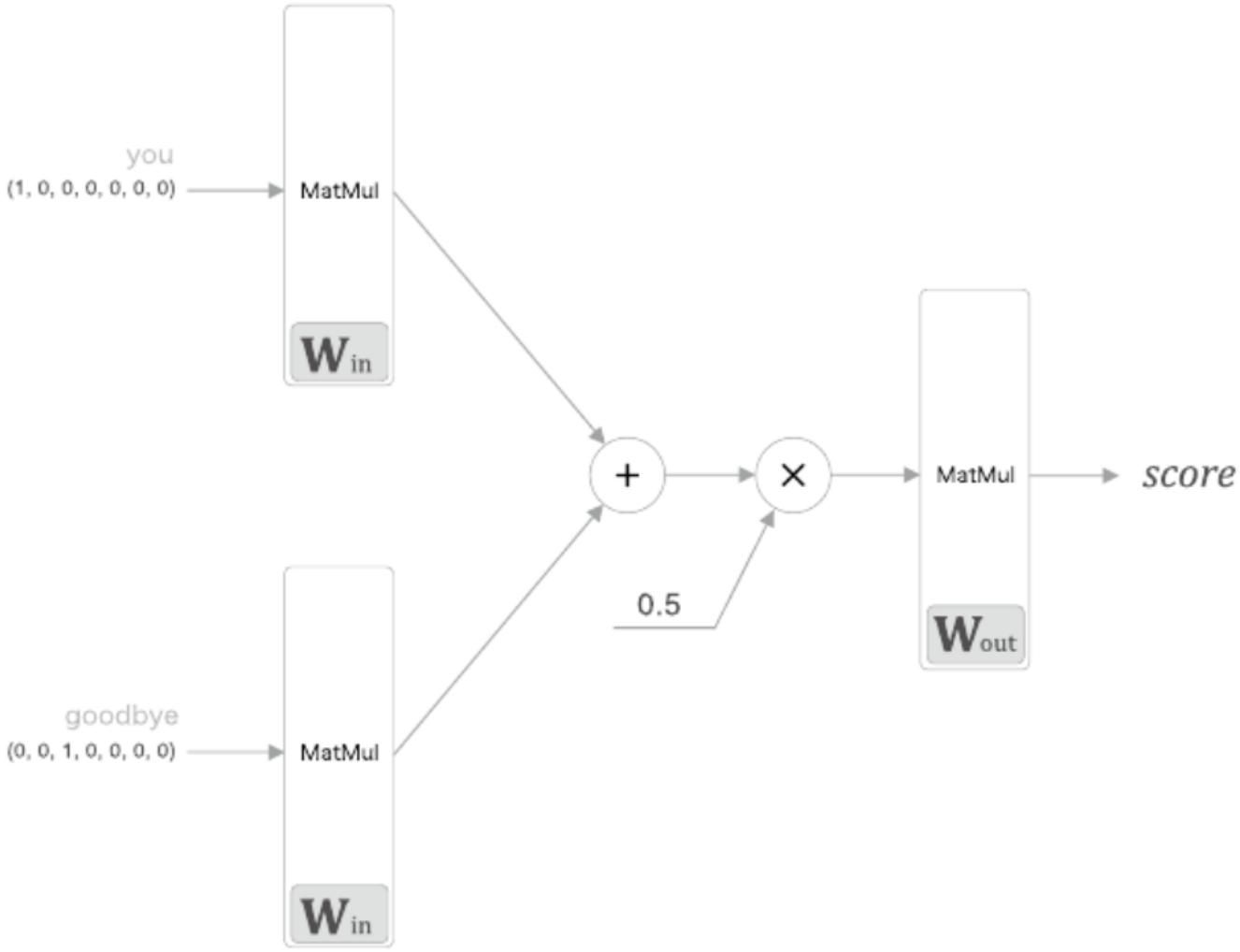
-> 편향이 생략되어 있고, one-hot vector이기 때문에 행렬곱 계산 결과가 가중치 행렬의 행 하나를 뽑은 것과 같음

2. 단순한 word2vec

CBOW 모델의 추론 처리

- 맥락으로부터 target을 추측하는 용도의 신경망
- **input layer**
  - input -> 각 단어의 one-hot vector
  - output -> input과 동일
  - input layer의 개수는 맥락에 포함시킬 단어 수에 따라 달라짐
- **input layer to hidden layer**
  - MatMul
    - W_{in} : 입력층에서 은닉층으로의 변환을 시키는 가중치 행렬
    - 모든 input layer은 동일한 W_{in} 을 공유
- **hidden layer**
  - input -> 각 input layer의 matmul 결과값
  - output -> 각 input layer의 matmul 결과값의 평균
- **output**
  - input -> hidden layer의 output





In [14]:

```
import os
function_path = os.path.join(".","function")
import sys
sys.path.append(function_path)
from base_layer import MatMul

# 샘플 맥락 데이터
c0 = np.array([[1,0,0,0,0,0,0]])
c1 = np.array([[0,0,1,0,0,0,0]])

# 가중치 초기화
W_in = np.random.randn(7,3)
W_out = np.random.randn(3,7)

# 계층 생성
in_layer0 = MatMul(W_in)
in_layer1 = MatMul(W_in)
out_layer = MatMul(W_out)

# 순전파
h0 = in_layer0.forward(c0)
h1 = in_layer1.forward(c1)
h = 0.5*(h0+h1)
s = out_layer.forward(h)
```

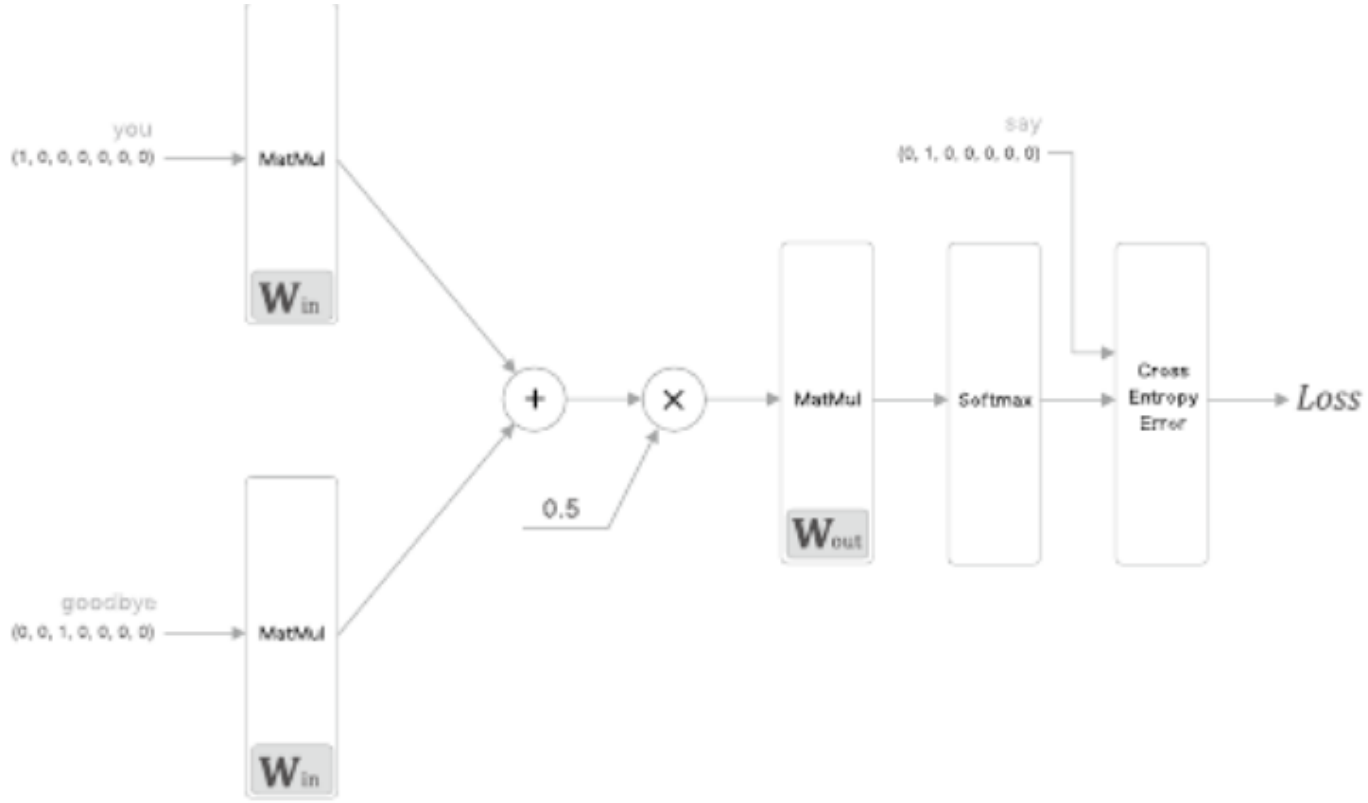
In [15]:

```
print(s)
```

```
[[ 0.44546982  2.11184992  1.76553713  0.40405985 -1.210514    0.32171621
  -0.53326582]]
```

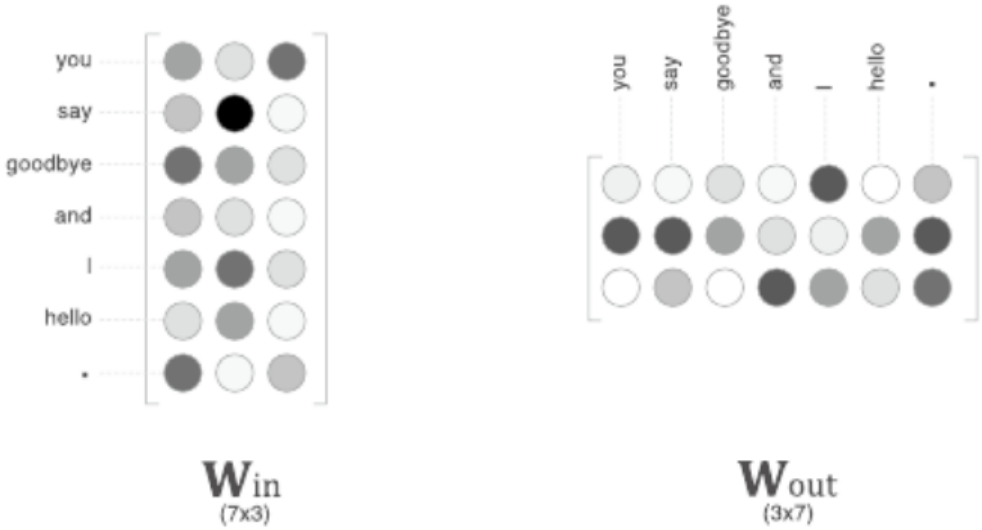
CBOW 모델의 학습

- output layer의 출력에 softmax를 적용하여 확률벡터를 얻은 후, cross-entropy-loss를 계산하여 Loss를 얻음
- 최종적으로 얻은 Loss에 대하여 gradient descent 적용



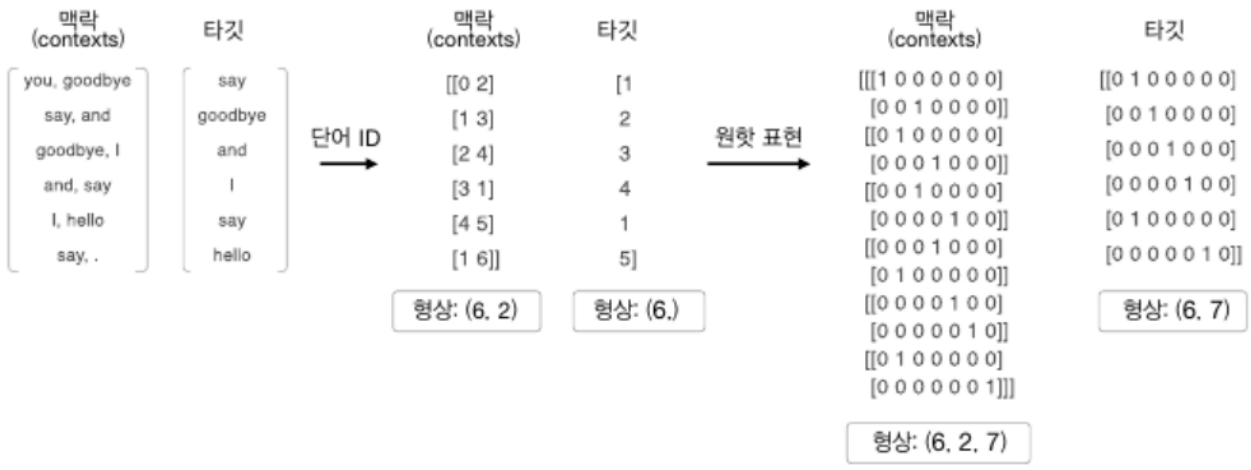
단어의 분산 표현

- W_{in}, W_{out}의 각 행에는 해당 단어의 분산 표현이 담겨 있게 됨
  *(word2vec*기법에서는 W_{in}만 사용하는 것이 가장 좋은 결과를 줌*)*
  - 은닉층의 뉴런수를 입력층의 뉴런 수 보다 적게 하는 것이 핵심. 그렇게 해야 밀집벡터 표현을 얻을 수 있음
  - 어떤 말뭉치를 넣느냐에 따라 같은 단어라도 분산표현이 달라짐
- 학습을 진행할 수록 해당 단어를 잘 표현하는 방향으로 분산 표현들이 갱신될 것임
  - 이 같은 표현은 시소러스, 통계학 기반처럼 추론 방식대로 단어를 표현한 것임



학습데이터 준비

1. 말뭉치 텍스트를 단어ID로 변환
2. 단어 ID에서 맥락(X)과 타겟(Y)을 만듦
   - 맥락에 포함될 단어 수가 정해지면 그 중앙에 위치할 수 있는 단어만 타겟값이 됨. 즉, 해당 단어에 맥락의 단어 수가 정해진 해당 수가 되면 타겟값이 될 수 있음
3. 맥락과 타겟을 one-hot-vector로 변환



1. 말뭉치 텍스트를 단어ID로 변환

In [19]:

```
from preprocess import preprocess
text = "You say goodbye and I say hello."
corpus, word_to_id, id_to_word = preprocess(text)
print(corpus)

print(id_to_word)
```

```
[0 1 2 3 4 1 5 6]
{0: 'you', 1: 'say', 2: 'goodbye', 3: 'and', 4: 'i', 5: 'hello', 6: '.'}
```

2. 맥락(X)와 타깃(Y)을 반환

In [20]:

```
def create_contexts_target(corpus, window_size = 1):
    target = corpus[window_size:-window_size]
    contexts = []

    for idx in range(window_size, len(corpus)-window_size):
        cs = [] # 해당 target의 context

        # 자기자신을 포함한 범위를 루프
        for t in range(-window_size, window_size + 1):
            if t==0: # 자기자신
                continue
            cs.append(corpus[idx+t])

        contexts.append(cs)

    return np.array(contexts), np.array(target)
```

In [21]:

```
contexts, target = create_contexts_target(corpus, window_size = 1)
```

In [22]:

```
print(contexts)
print(target)
```

```
[[0 2]
 [1 3]
 [2 4]
 [3 1]
 [4 5]
 [1 6]]
[1 2 3 4 1 5]
```

one-hot-vector로 변환

- X : 6x2 -> 6x2x7
- Y : 1x6 -> 6x7

In [23]:

```
def convert_one_hot(corpus, vocab_size):
    """
    원핫 표현으로 변환

    : param corpus : 단어 ID목록(1차원 또는 2차원 넘파이 배열)
    : param vocab_size : 어휘 수
    : return : 원핫표현(2차원 or 3차원 numpy 배열)
    """

    N = corpus.shape[0]

    # 주로 target
    if corpus.ndim == 1:
        one_hot = np.zeros((N, vocab_size), dtype = np.int32)
        for idx, word_id in enumerate(corpus):
            one_hot[idx, word_id] = 1

    # 주로 context
    elif corpus.ndim == 2:
        C = corpus.shape[1]
        one_hot = np.zeros((N, C, vocab_size), dtype = np.int32)
        for idx_0, word_ids in enumerate(corpus):
            for idx_1, word_id in enumerate(word_ids):
                one_hot[idx_0, idx_1, word_id] = 1

    return one_hot
```

In [24]:

```
vocab_size = len(word_to_id)
target = convert_one_hot(target, vocab_size)
contexts = convert_one_hot(contexts, vocab_size)
```

In [25]:

```
target
```

Out[25]:

```
array([[0, 1, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0],
       [0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0]])
```

In [26]:

```
contexts
```

Out[26]:

```
array([[[1, 0, 0, 0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0, 0]],

       [[0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 0]],

       [[0, 0, 1, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0]],

       [[0, 0, 0, 1, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0]],

       [[0, 0, 0, 0, 1, 0, 0],
        [0, 0, 0, 0, 0, 1, 0]],

       [[0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 1]]])
```

3. CBOW 모델 구현

In [1]:

```
import sys
import os
function_path = os.path.join("..","function")
sys.path.append(function_path)
import numpy as np
from base_layer import MatMul,SoftmaxWithLoss

class SimpleCBOW:
    def __init__(self, vocab_size, hidden_size):
        V, H = vocab_size, hidden_size

        # 가중치 초기화
        W_in = 0.01 * np.random.randn(V,H).astype("f")
        W_out = 0.01 * np.random.randn(H,V).astype("f") # 32비트 부동소수점 수

        # 계층 생성
        # input layers
        self.in_layer0 = MatMul(W_in)
        self.in_layer1 = MatMul(W_in)
        # output layer
        self.out_layer = MatMul(W_out)
        # loss layer
        self.loss_layer = SoftmaxWithLoss()

        # 모든 가중치와 기울기를 리스트에 모은다.
        layers = [self.in_layer0, self.in_layer1, self.out_layer]
        self.params, self.grads = [], []
        for layer in layers:
            self.params += layer.params
            self.grads += layer.grads

        # 인스턴스 변수에 단어에 분산 표현을 저장
        self.words_vecs = W_in

    def forward(self, contexts, target):
        h0 = self.in_layer0.forward(contexts[:,0])
        h1 = self.in_layer1.forward(contexts[:,1])
        h = (h0+h1)*0.5
        score = self.out_layer.forward(h)
        loss = self.loss_layer.forward(target, score)
        return loss

    # grads 갱신
    def backward(self, dout=1):
        ds = self.loss_layer.backward(dout)
        da = self.out_layer.backward(ds)
        da *= 0.5
        self.in_layer1.backward(da)
        self.in_layer0.backward(da)
        return None
```

- 같은 가중치가 여러 개 존재하기 때문에 SGD를 쓰면 문제가 안되겠지만, momentum이나 adagrad, adam과 같은 optimizer를 사용하게 되면 같은 가중치이지만 각각 다른 grad를 가지기 때문에 본래의 동작과 달라짐 (다른 가중치로 취급되기 때문) -> 이를 해결하기 위해 새로운 함수 필요

In [2]:

```
def remove_duplicate(params, grads):
    '''
    매개변수 배열 중 중복되는 가중치를 하나로 모아
    그 가중치에 대응하는 기울기를 더한다.
    '''
    # copy list(리스트 원소는 따라가지 않고, 넘파이 배열안의 객체들만 따라갈 수 있도록)
    params, grads = params[:], grads[:]

    while True:
        find_flg = False
        L = len(params)

        for i in range(0, L - 1):
            for j in range(i + 1, L):
                # 가중치 공유 시
                if params[i] is params[j]:
                    grads[i] += grads[j]  # 경사를 더함
                    find_flg = True # 반복문을 빠져나감
                    params.pop(j) # 해당 파라미터 삭제
                    grads.pop(j) # 해당 grads 삭제

                # 가중치를 전치행렬로 공유하는 경우(weight tying)
                elif params[i].ndim == 2 and params[j].ndim == 2 and ₩
                     params[i].T.shape == params[j].shape and np.all(params[i].T == params[j]):
                    grads[i] += grads[j].T
                    find_flg = True
                    params.pop(j)
                    grads.pop(j)

            # 중복된 것을 찾았을 시 빠져나감
                if find_flg: break
            if find_flg: break

        # 중복된 것을 찾지 못했더라도 for 문을 다 돌았으면 빠져나감
        if not find_flg: break

    return params, grads
```

In [35]:

```
class Adam:
    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.iter = 0
        self.m = None
        self.v = None

    def update(self, params, grads):
        if self.m is None:
            self.m, self.v = [], []
            for param in params:
                self.m.append(np.zeros_like(param))
                self.v.append(np.zeros_like(param))

        self.iter += 1
        lr_t = self.lr * np.sqrt(1.0 - self.beta2**self.iter) / (1.0 - self.beta1**self.iter)

        for i in range(len(params)):
            self.m[i] += (1 - self.beta1) * (grads[i] - self.m[i])
            self.v[i] += (1 - self.beta2) * (grads[i]**2 - self.v[i])

            params[i] -= lr_t * self.m[i] / (np.sqrt(self.v[i]) + 1e-7)
```

In [4]:

```
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
```

In [7]:

```
import time
import matplotlib.pyplot as plt
import numpy
import matplotlib.pyplot as plt
from clip_grads import clip_grads
import sys
sys.path.append('.')

class Trainer:
    def __init__(self, model, optimizer):
        self.model = model
        self.optimizer = optimizer
        self.loss_list = []
        self.eval_interval = None
        self.current_epoch = 0

    def fit(self, x, t, max_epoch=10, batch_size=32, max_grad=None, eval_interval=20):
        data_size = len(x)
        max_iters = data_size // batch_size
        self.eval_interval = eval_interval
        model, optimizer = self.model, self.optimizer
        total_loss = 0
        loss_count = 0

        start_time = time.time()
        for epoch in range(max_epoch):
            # 뒤섞기
            idx = numpy.random.permutation(numpy.arange(data_size))
            x = x[idx]
            t = t[idx]

            for iters in range(max_iters):
                batch_x = x[iters*batch_size:(iters+1)*batch_size]
                batch_t = t[iters*batch_size:(iters+1)*batch_size]

                # 기울기 구해 매개변수 갱신
                loss = model.forward(batch_x, batch_t)
                model.backward()
                # 공유된 가중치를 하나로 모음
                params, grads = remove_duplicate(model.params, model.grads)
                if max_grad is not None:
                    clip_grads(grads, max_grad)
                optimizer.update(params, grads)
                total_loss += loss
                loss_count += 1

                # 평가
                if (eval_interval is not None) and (iters % eval_interval) == 0:
                    avg_loss = total_loss / loss_count
                    elapsed_time = time.time() - start_time
                    print('| 에폭 %d |  반복 %d / %d | 시간 %d[s] | 손실 %.2f'
                          % (self.current_epoch + 1, iters + 1, max_iters, elapsed_time, avg_loss))
                    self.loss_list.append(float(avg_loss))
                    total_loss, loss_count = 0, 0

            self.current_epoch += 1

    def plot(self, ylim=None):
        x = numpy.arange(len(self.loss_list))
```

```
        if ylim is not None:
            plt.ylim(*ylim)
        plt.plot(x, self.loss_list, label='train')
        plt.xlabel('반복 (x' + str(self.eval_interval) + ')')
        plt.ylabel('손실')
        plt.show()
```

In [20]:

```
from matplotlib import font_manager, rc
import matplotlib as mpl
font_path = "C:₩₩Users₩₩이혜림₩₩Desktop₩₩Bita5/malgun.ttf"
font_name = font_manager.FontProperties(fname=font_path).get_name()
rc("font",family=font_name)
mpl.rcParams["axes.unicode_minus"]=False
```

In [36]:

```
from preprocess import *
from create_contexts_target import *
from convert_one_hot import *
import numpy

window_size = 1
hidden_size = 5
batch_size = 3
max_epoch = 1000

text = "You say goodbye and I say hello."
corpus, word_to_id, id_to_word = preprocess(text)

vocab_size = len(word_to_id)
contexts, target = create_contexts_target(corpus, window_size)
target = convert_one_hot(target, vocab_size)
contexts = convert_one_hot(contexts, vocab_size)

model = SimpleCBOW(vocab_size, hidden_size)
optimizer = Adam()
trainer = Trainer(model, optimizer)

trainer.fit(contexts, target, max_epoch, batch_size)
trainer.plot()
```

```
| 에폭 982 |  반복 1 / 2 | 시간 0[s] | 손실 0.76
| 에폭 983 |  반복 1 / 2 | 시간 0[s] | 손실 0.47
| 에폭 984 |  반복 1 / 2 | 시간 0[s] | 손실 0.54
| 에폭 985 |  반복 1 / 2 | 시간 0[s] | 손실 0.65
| 에폭 986 |  반복 1 / 2 | 시간 0[s] | 손실 0.65
| 에폭 987 |  반복 1 / 2 | 시간 0[s] | 손실 0.51
| 에폭 988 |  반복 1 / 2 | 시간 0[s] | 손실 0.47
| 에폭 989 |  반복 1 / 2 | 시간 0[s] | 손실 0.83
| 에폭 990 |  반복 1 / 2 | 시간 0[s] | 손실 0.26
| 에폭 991 |  반복 1 / 2 | 시간 0[s] | 손실 0.90
| 에폭 992 |  반복 1 / 2 | 시간 0[s] | 손실 0.36
| 에폭 993 |  반복 1 / 2 | 시간 0[s] | 손실 0.61
| 에폭 994 |  반복 1 / 2 | 시간 0[s] | 손실 0.58
| 에폭 995 |  반복 1 / 2 | 시간 1[s] | 손실 0.68
| 에폭 996 |  반복 1 / 2 | 시간 1[s] | 손실 0.54
| 에폭 997 |  반복 1 / 2 | 시간 1[s] | 손실 0.58
| 에폭 998 |  반복 1 / 2 | 시간 1[s] | 손실 0.36

| 에폭 999 |  반복 1 / 2 | 시간 1[s] | 손실 0.79
```

단어의 분산표현

In [17]:

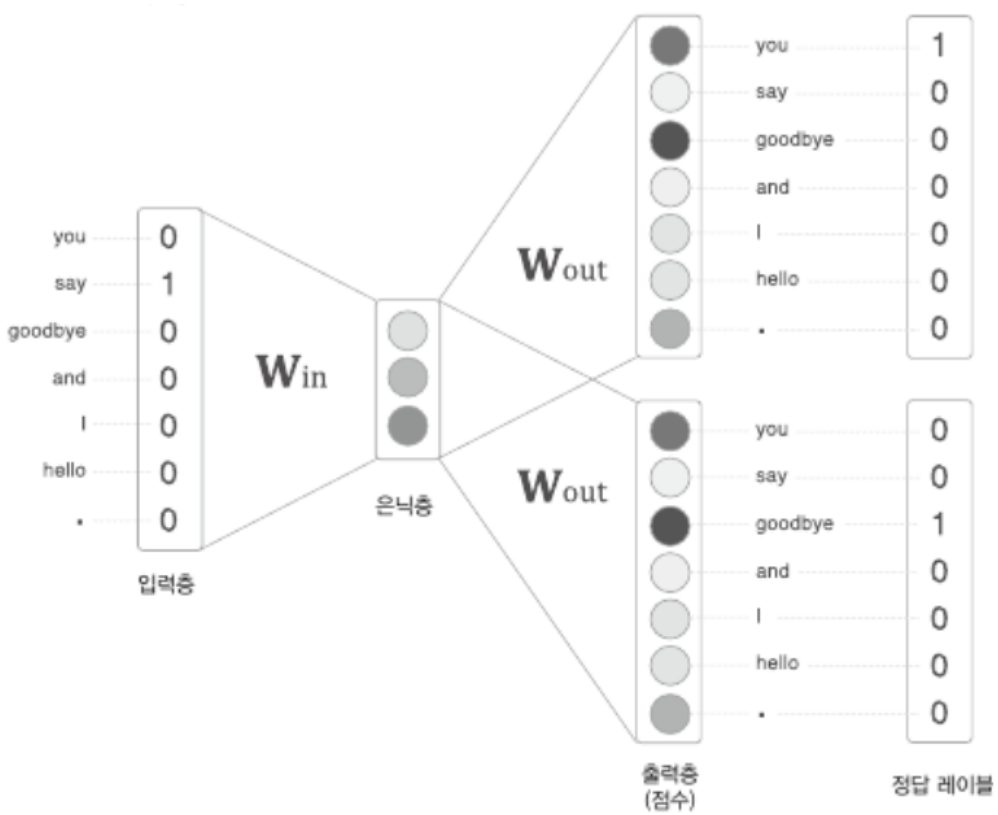
```
word_vecs = model.words_vecs
for word_id, word in id_to_word.items():
    print(word, word_vecs[word_id])
```

```
you [ 0.9558914   1.6459398   0.80795175 -0.95786256  1.0635065 ]
say [-1.1562741   1.3770623 -1.1650794   1.156217   -0.7613188]
goodbye [ 0.93601257  0.13669725  1.123671   -0.8965539   0.7909736 ]
and [-0.64508307  0.8818006  -0.30211514  0.8941325  -2.009504  ]
i [ 0.9491232   0.13179895  1.1173044  -0.92354035  0.82773477]
hello [ 0.97107685  1.6165062   0.78321266 -0.9693891   1.0641565 ]
. [-1.3314737   1.4029845 -1.474378    1.1280481   1.5401313]
```

4. skip-gram model

- CBOW 모델이 맥락으로부터 중앙의 단어를 추축한다면, skip-gram model은 중앙으로부터 맥락들의 단어를 추측
- 같은 output weight를 통하여 같은 output matrix결과를 내지만, contexts가 다르기 때문에 각 loss가 달라짐. 각 손실을 더함으로써 최종 loss를 구함

- input : 1개로 고정
- output : 맥락의 크기에 따라 달라짐, 같은 output weight를 공유



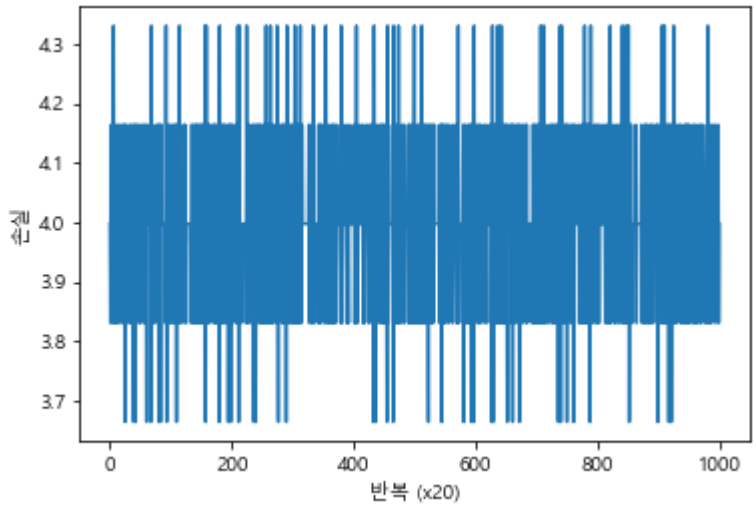
skip-gram model 구현

In [30]:

```
import sys
import os
function_path = os.path.join("..","function")
sys.path.append(function_path)
from base_layer import SoftmaxWithLoss, MatMul
import numpy as np

class SimpleSkipGram:
    def __init__(self, vacab_size, hidden_size):
        V, H  = vocab_size, hidden_size

        # 가중치 초기화
        W_in = 0.01*np.random.randn(V,H).astype("f")
        W_out = 0.01*np.random.randn(H,V).astype("f")

        # layer 생성
        self.in_layer = MatMul(W_in)
        self.out_layer = MatMul(W_out)
        self.loss_layer1 = SoftmaxWithLoss()
        self.loss_layer2 = SoftmaxWithLoss()

        layers = [self.in_layer, self.out_layer]
        self.params, self.grads = [], []
        for layer in layers:
            self.params += layer.params
            self.grads += layer.grads

        # 인스턴스 변수에 단어의 분산 표현을 저장
        self.words_vecs = W_in

    def forward(self, contexts, target):
        h = self.in_layer.forward(target)
        s = self.out_layer.forward(h)
        l1 = self.loss_layer1.forward(s, contexts[:,0])
        l2 = self.loss_layer2.forward(s, contexts[:,1])
        loss = l1+l2
        return loss

    def backward(self, dout=1):
        dl1 = self.loss_layer1.backward(dout)
        dl2 = self.loss_layer2.backward(dout)
        ds = dl1+dl2
        dh = self.out_layer.backward(ds)
        self.in_layer.backward(dh)
        return None
```

In [37]:

```
from preprocess import *
from create_contexts_target import *
from convert_one_hot import *
import numpy

window_size = 1
hidden_size = 5
batch_size = 3
max_epoch = 1000

text = "You say goodbye and I say hello."
corpus, word_to_id, id_to_word = preprocess(text)

vocab_size = len(word_to_id)
contexts, target = create_contexts_target(corpus, window_size)
target = convert_one_hot(target, vocab_size)
contexts = convert_one_hot(contexts, vocab_size)

model = SimpleSkipGram(vocab_size , hidden_size)
optimizer = Adam()
trainer = Trainer(model, optimizer)

trainer.fit(contexts, target, max_epoch, batch_size)
trainer.plot()
```



5. CBOW vs skip-gram

- 단어 분산 표현
  - skip-gram model이 좀 더 어려운 상황에서 단련하기 때문에 단어 분산 표현의 정밀도 면에서 더 좋은 결과를 내는 경우가 많음
- 유추 문제 성능
  - 특히 말뭉치가 커질수록 저빈도 단어나 유추문제의 성능 면에서 더 뛰어남
- 학습 속도
  - CBOW가 더 빠름
  - skip-gram의 경우 손실을 맥락의 수만큼 구해야 해서 계산비용이 커짐

6. 통계 기반 vs 추론 기반

- 갱신성
  - 통계 기반 학습 : 동시발생 행렬을 새로 다시 만들고 SVD를 수행하는 일련의 작업을 다시 거쳐야함
  - 추론 기반 학습 : 학습된 가중치를 초기값으로 놓고 다시 학습을 진행함으로써 새로운 데이터가 들어왔을 때 단어의 분산 표현을 효율적으로 갱신할 수 있음
- 단어의 분산 표현/정밀도
  - 통계 기반 학습 : 단어의 유사성이 인코딩
  - 추론 기반 학습 : 단어의 유사성 & 복잡한 단어 사이의 패턴

- but, 단어의 유사성 관련 작업은 정확성은 하이퍼파라미터에 크게 의존하며, 우열을 명확히 가릴 수 없음

통계기반 + 추론기반

- 두 기법은 (특정 조건 하에서) 서로 연결되어 있음
- 나아가 추론 기반 기법과 통계 기반 기법을 융합한 GloVe 기법이 등장
  - 말뭉치 전체의 통계 정보를 손실 함수에 도입해 미니배치 학습을 진행