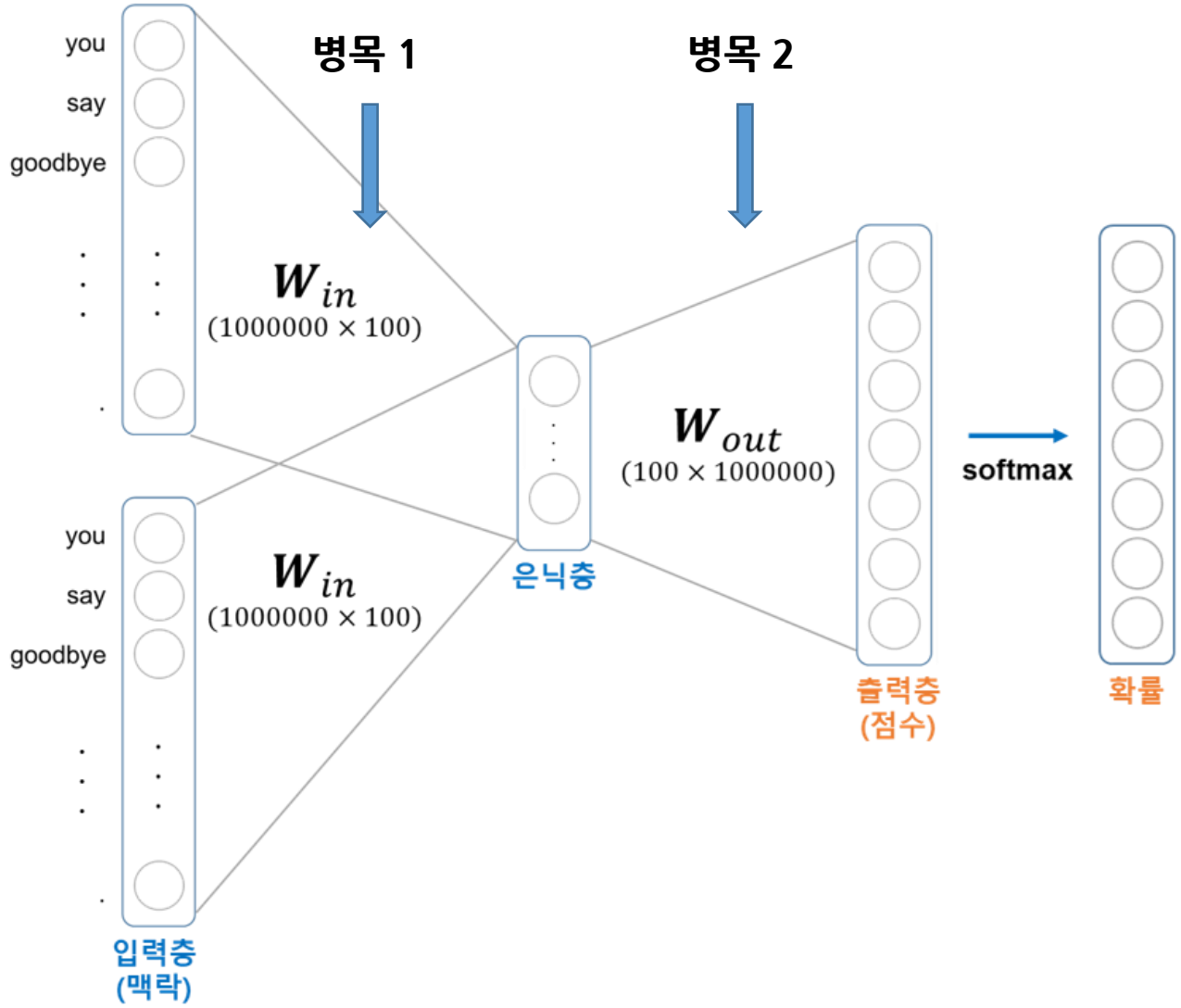
4장 word2vec 개선

CBOW 모델

어휘가 100만 개일 때를 가정한 CBOW 모델



병목

1. 입력층의 원 핫 표현과 가중치 행렬 W(in)의 곱 계산

2. 은닉층과 가중치 행렬 W(out)의 곱 및 Softmax 계층의 계산

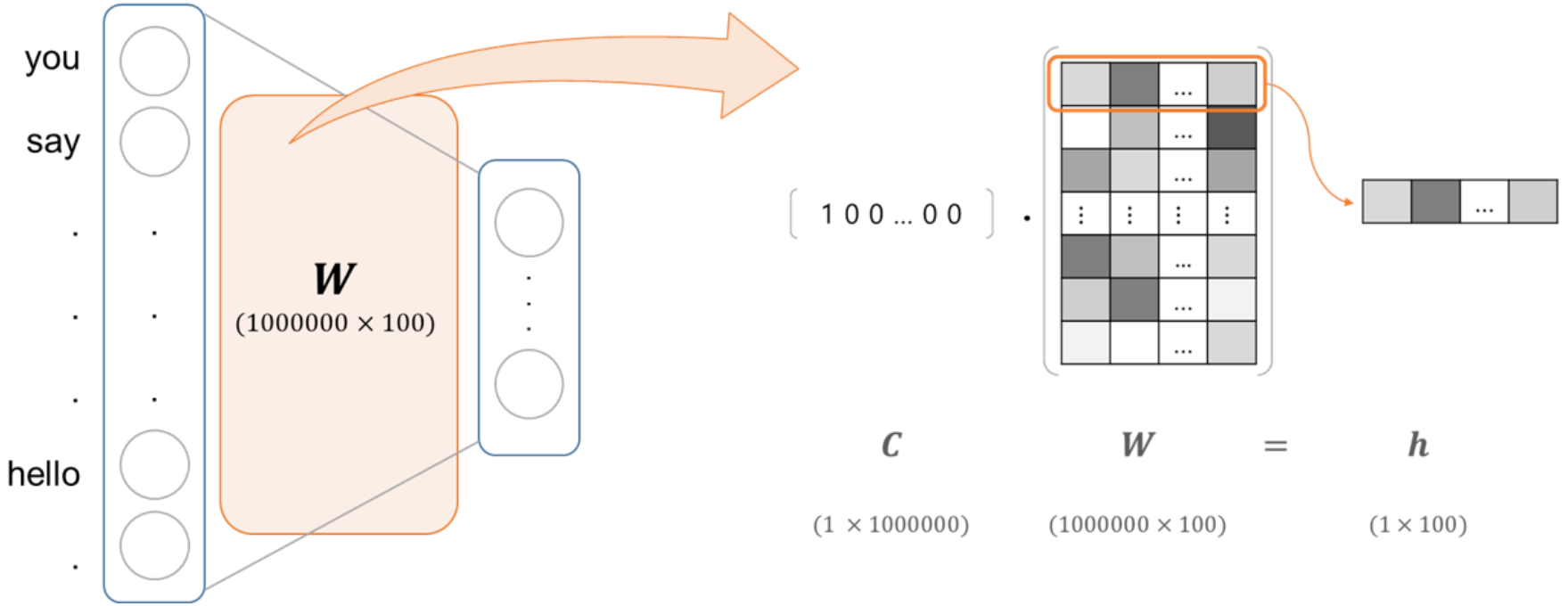
해결 방안

1. Embedding 계층 도입

2. 네거티브 샘플링 사용

CBOW 모델 개선

Embedding 계층 사용



결론

원 핫 표현으로써의 변환과 Matmul 계층의 행렬 곱 계산은 필요 없음 => 가중치 행렬의 행을 추출과 동일

Embedding 계층 : 가중치 행렬의 특정 행을 추출하여 반환하는 계층

CBOW 모델 개선

Embedding 계층 구현

임의의 행렬 정의

```
import numpy as np

W = np.arange(21).reshape(7, 3)
W
```

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14],
       [15, 16, 17],
       [18, 19, 20]])
```

행렬 인덱싱

```
# 두 번째 행 가져오기
# -> index=2에 해당하는 단어 벡터
W[2]
```

```
array([6, 7, 8])
```

여러 행 추출

```
# 가중치 W로 부터 여러행을
# 한꺼번에 추출하는 예제
idx = np.array([1, 0, 3, 0])
W[idx]
```

```
array([[ 3,  4,  5],
       [ 0,  1,  2],
       [ 9, 10, 11],
       [ 0,  1,  2]])
```

CBOW 모델 개선

Embedding 계층 순전파와 역전파

순전파

가중치 W의 특정 행을 추출하는 것

=> 특정 행을 추출해 반환

```
def forward(self, idx):
    W, = self.params
    self.idx = idx
    out = W[idx]
    return out
```

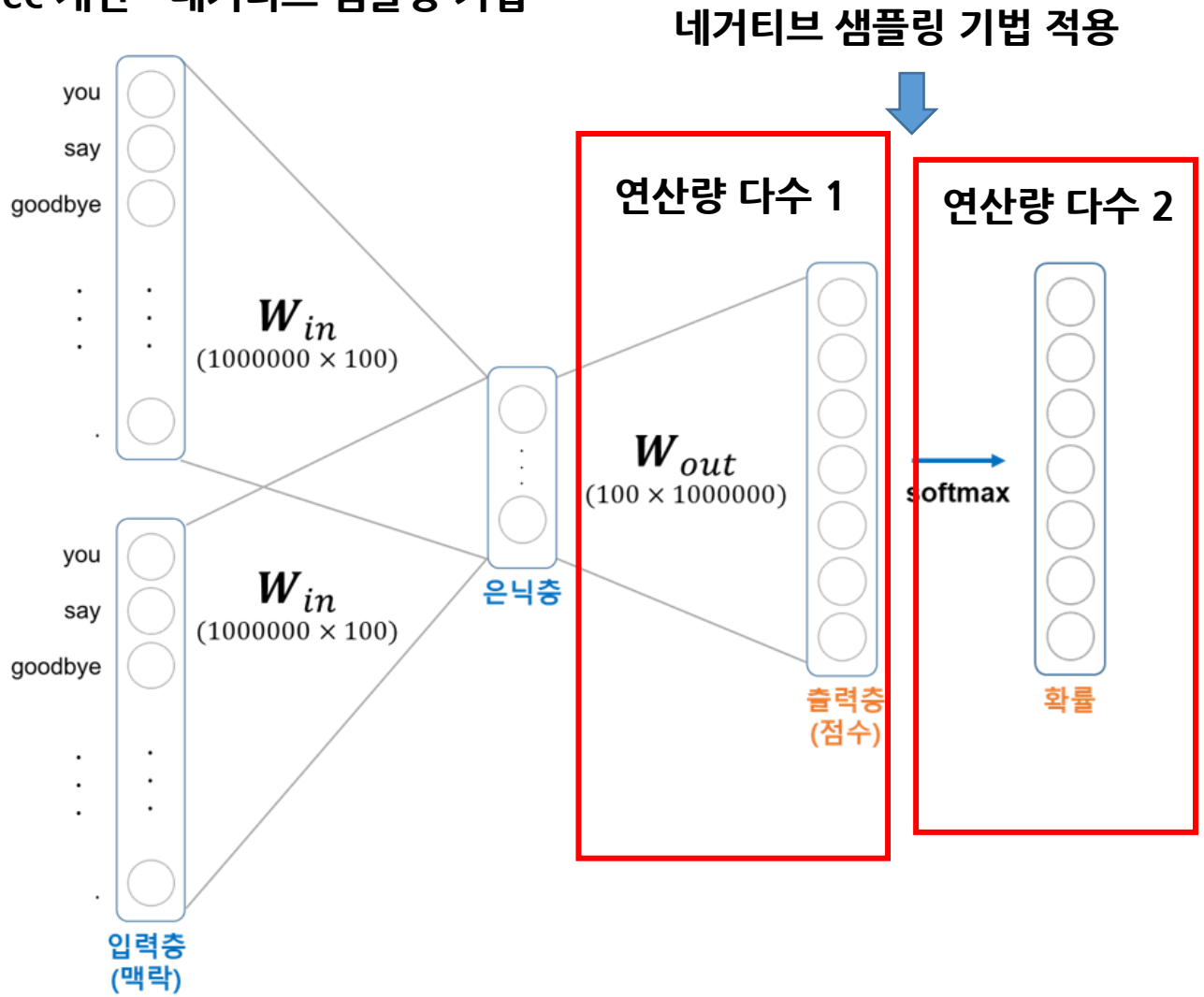
역전파

출력 층 쪽으로부터 전해진 기울기를 그대로 흘려주는 것

```
def backward(self, dout):
    dW, = self.grads
    dW[...] = 0
    np.add.at(dW, self.idx, dout)
    return None
```

- CBOW 모델 개선

Word2vec 개선 - 네거티브 샘플링 기법



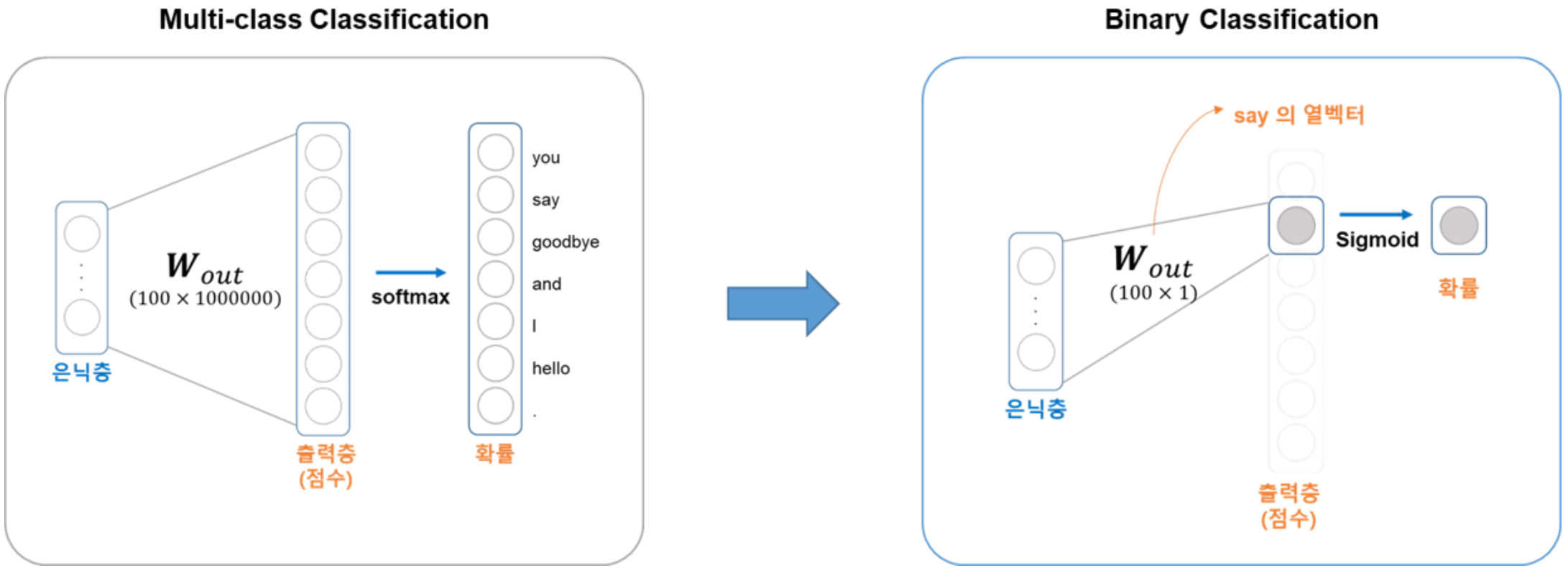
병목

1. 은닉층의 뉴런과 가중치 행렬 W(out)의 곱

2. Softmax 계층의 계산

- CBOW 모델 개선

Word2vec 개선 - 네거티브 샘플링 기법



다중 분류 -> 이진 분류 문제로 근사

예)

You와 goodbye 입력 시 say 에 대해서만 점수를 출력

CBOW 모델 개선

네거티브 샘플링 기법

이진 분류 문제

- 출력 층에 시그모이드 함수 적용 -> 확률로 변환

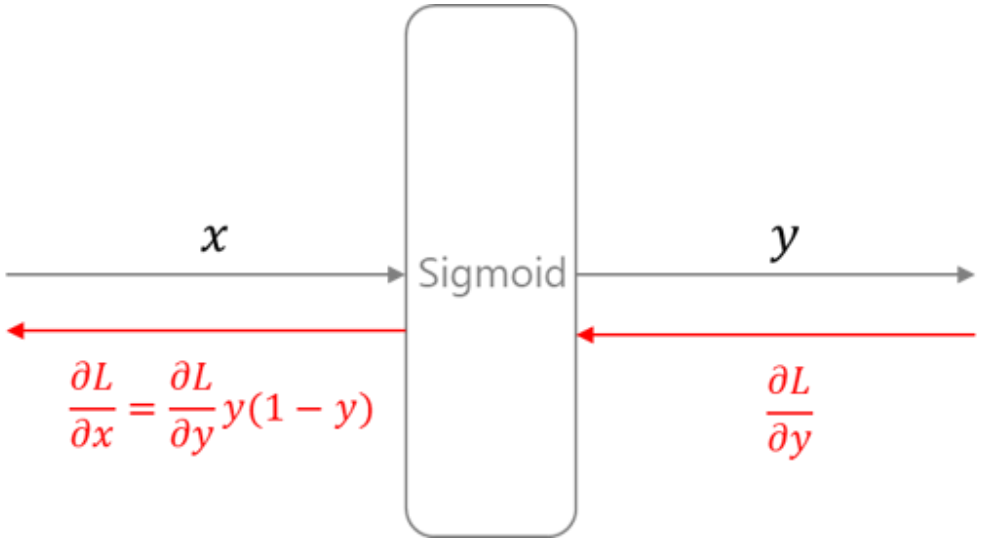
- 손실 함수로 교차 엔트로피 오차를 사용

$$L = -[t\log y + (1-t)\log(1-y)]$$

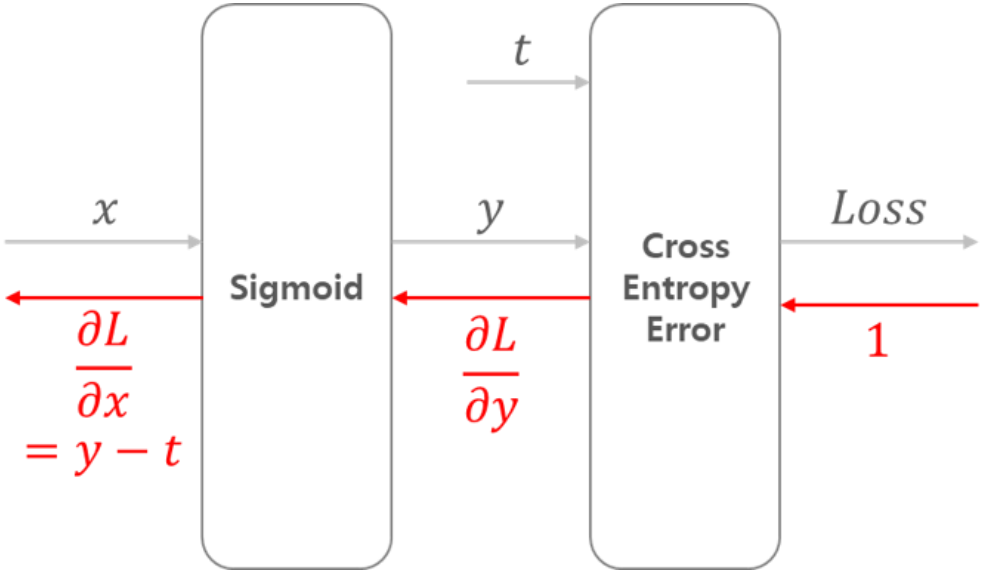
- CBOW 모델 개선

네거티브 샘플링 기법

시그모이드 계층 역전파

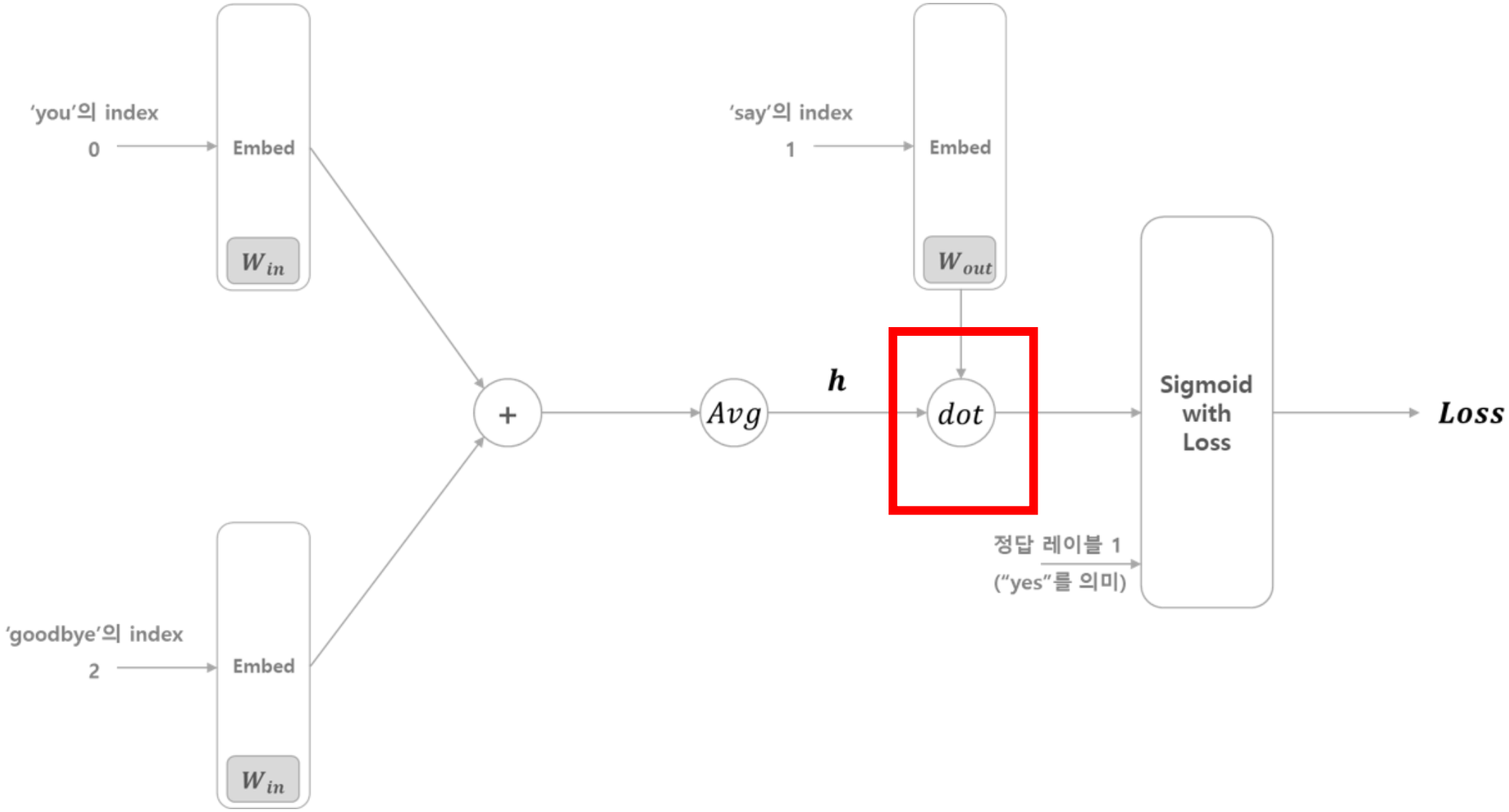


역전파 과정



CBOW 모델 개선

임베딩 계층과 이진 분류 적용



- CBOW 모델 개선

네거티브 샘플링 기법

학습

정답인 단어를 사용해 손실 함수 사용, 정답이 아닌 다른 단어에 대한 확률 값은 어떻게 구할지 잘 학습하지 못함

=> 네거티브 샘플링 사용

예)
입력: you 와 goodbye
출력으로 say를 예측하고, 이외의 단어에는 출력이 0에 가까워야 하기 위해 정답 레이블 이외의 단어 또한 학습에 포함

- CBOW 모델 개선

네거티브 샘플링 기법

학습

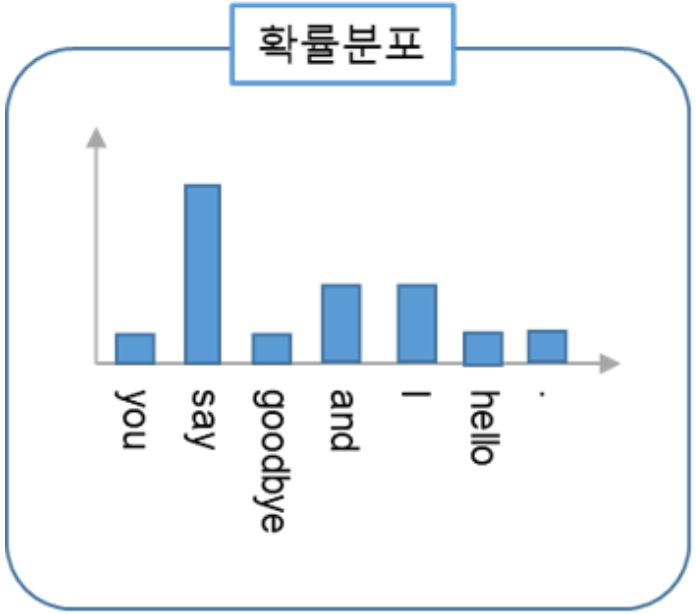
정답인 단어를 사용해 손실 함수 사용, 정답이 아닌 다른 단어에 대한 확률 값은 어떻게 구할지 잘 학습하지 못함

=> 네거티브 샘플링 사용

예)
입력: you 와 goodbye
출력으로 say를 예측하고, 이외의 단어에는 출력이 0에 가까워야 하기 위해 정답 레이블 이외의 단어 또한 학습에 포함

- CBOW 모델 개선

네거티브 샘플링 기법



샘플링

you

say

goodbye

and

I

hello

.

무작위 추출보다 성능이 좋은 방법

=> 확률 분포에 기반해 추출하는 방법

- CBOW 모델 개선

네거티브 샘플링 기법

확률 분포 제곱

```
p = [0.7, 0.29, 0.01]
new_p = np.power(p, 0.75)

new_p /= np.sum(new_p)
print(new_p)
```

```
[0.64196878 0.33150408 0.02652714]
```

$$P'(w_i) = \frac{P(w_i)^{0.75}}{\sum_j^n p(w_j)^{0.75}}$$

- 분포에서 희소한 단어가 더 쉽게 샘플링 되도록 하는 방법

- CBOW 모델 개선

네거티브 샘플링 파이썬 구현

```
class NegativeSamplingLoss:
    def __init__(self, W, corpus, power=0.75, sample_size=5):
        self.sample_size = sample_size
        self.sampler = UnigramSampler(corpus, power, sample_size)
        self.loss_layers = [SigmoidWithLoss() for _ in range(sample_size + 1)]
        self.embed_dot_layers = [EmbeddingDot(W) for _ in range(sample_size + 1)]

        self.params, self.grads = [], []
        for layer in self.embed_dot_layers:
            self.params += layer.params
            self.grads += layer.grads

    def forward(self, h, target):
        batch_size = target.shape[0]
        negative_sample = self.sampler.get_negative_sample(target)
```

- CBOW 모델 개선

네거티브 샘플링 순전파

```
def forward(self, h, target):
    batch_size = target.shape[0]
    negative_sample = self.sampler.get_negative_sample(target)

    # 긍정적 예 순전파
    score = self.embed_dot_layers[0].forward(h, target)
    correct_label = np.ones(batch_size, dtype=np.int32)
    loss = self.loss_layers[0].forward(score, correct_label)

    # 부정적 예 순전파
    negative_label = np.zeros(batch_size, dtype=np.int32)
    for i in range(self.sample_size):
        negative_target = negative_sample[:, i]  # embed_dot에 해당하는
        score = self.embed_dot_layers[1 + i].forward(h, negative_target)
        loss += self.loss_layers[1 + i].forward(score, negative_label)

    return loss
```

네거티브 샘플링 역전파

```
def backward(self, dout=1):
    dh = 0
    for l0, l1 in zip(self.loss_layers, self.embed_dot_layers):
        dscore = l0.backward(dout)
        dh += l1.backward(dscore)

    return dh
```

CBOW 모델 개선

개선 word2vec 학습

```
class CBOW:
    def __init__(self, vocab_size, hidden_size, window_size, corpus):
        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(V, H).astype('f')

        # 레이어 생성
        self.in_layers = []
        for i in range(2 * window_size):
            layer = Embedding(W_in)  # Embedding 계층 사용
            self.in_layers.append(layer)
        self.ns_loss = NegativeSamplingLoss(W_out, corpus, power=0.75, sample_size=5)

        # 모든 가중치와 기울기를 배열에 모은다.
        layers = self.in_layers + [self.ns_loss]
        self.params, self.grads = [], []
        for layer in layers:
            self.params += layer.params
            self.grads += layer.grads

        # 인스턴스 변수에 단어의 분산 표현을 저장한다.
        self.word_vecs1 = W_in
        self.word_vecs2 = W_out
```

- CBOW 모델 개선

개선 word2vec 학습

하이퍼파라미터 설정

```
window_size = 5
hidden_size = 100
batch_size = 100
max_epoch = 10

# 데이터 읽기
corpus, word_to_id, id_to_word = ptb.load_data('train')
vocab_size = len(word_to_id)

contexts, target = create_contexts_target(corpus, window_size)
if config.GPU:
    contexts, target = to_gpu(contexts), to_gpu(target)
```

모델 생성

```
model = SkipGram(vocab_size, hidden_size, window_size, corpus)
optimizer = Adam()
trainer = Trainer(model, optimizer)
```

학습 시작

```
# 학습 시작
trainer.fit(contexts, target, max_epoch, batch_size, eval_interval=2000)
trainer.plot()
```

- CBOW 모델 개선

CBOW 모델 평가

학습한 단어에 대해 거리가 가장 가까운 단어들을 추출

```
pkl_file = './cbow_params.pkl'
with open(pkl_file, 'rb') as f:
    params = pickle.load(f)
```

```
word_vecs = params['word_vecs']
word_to_id = params['word_to_id']
id_to_word = params['id_to_word']
```

```
# 가장 비슷한(most similar) 단어 뽑기
querys = ['you', 'year', 'car', 'toyota']
for query in querys:
    most_similar(query, word_to_id, id_to_word, word_vecs, top=5)
```

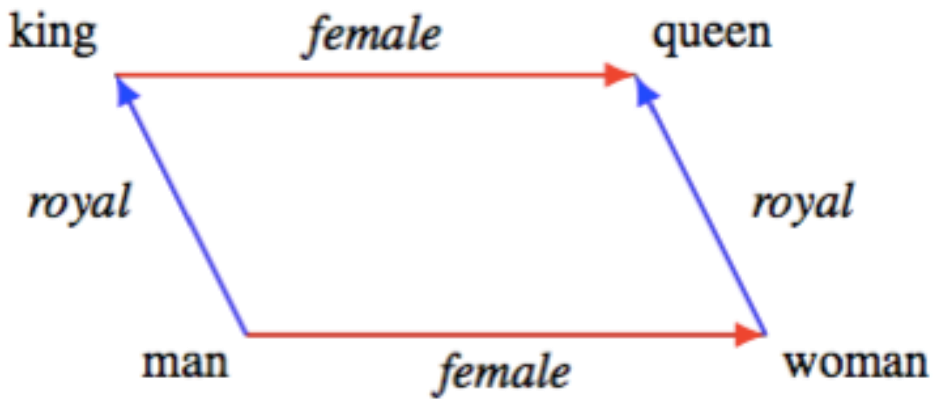
출력

```
[query] you
 we: 0.6103515625
 someone: 0.59130859375
 i: 0.55419921875
 something: 0.48974609375
 anyone: 0.47314453125

[query] year
 month: 0.71875
 week: 0.65234375
 spring: 0.62744140625
 summer: 0.6259765625
 decade: 0.603515625

[query] car
 luxury: 0.497314453125
 arabia: 0.47802734375
 auto: 0.47119140625
 disk-drive: 0.450927734375
 travel: 0.4091796875

[query] toyota
 ford: 0.55078125
 instrumentation: 0.509765625
 mazda: 0.49365234375
 bethlehem: 0.47509765625
 nissan: 0.474853515625
```

- CBOW 모델 개선

CBOW 모델 평가

유추 문제



예)

Vec(woman) - Vec(man) = Vec(?) - Vec(King)

유추 작업 코드

```
# 유추(analogy) 작업
print('-'*50)
analogy('king', 'man', 'queen',  word_to_id, id_to_word, word_vecs)
analogy('take', 'took', 'go',  word_to_id, id_to_word, word_vecs)
analogy('car', 'cars', 'child',  word_to_id, id_to_word, word_vecs)
analogy('good', 'better', 'bad',  word_to_id, id_to_word, word_vecs)
```

```
[analogy] king:man = queen:?
 woman: 5.16015625
 veto: 4.9296875
 ounce: 4.69140625
 earthquake: 4.6328125
 successor: 4.609375
```

코드

예)

King : man = queen : vec(?)

=> Woman

감사합니다