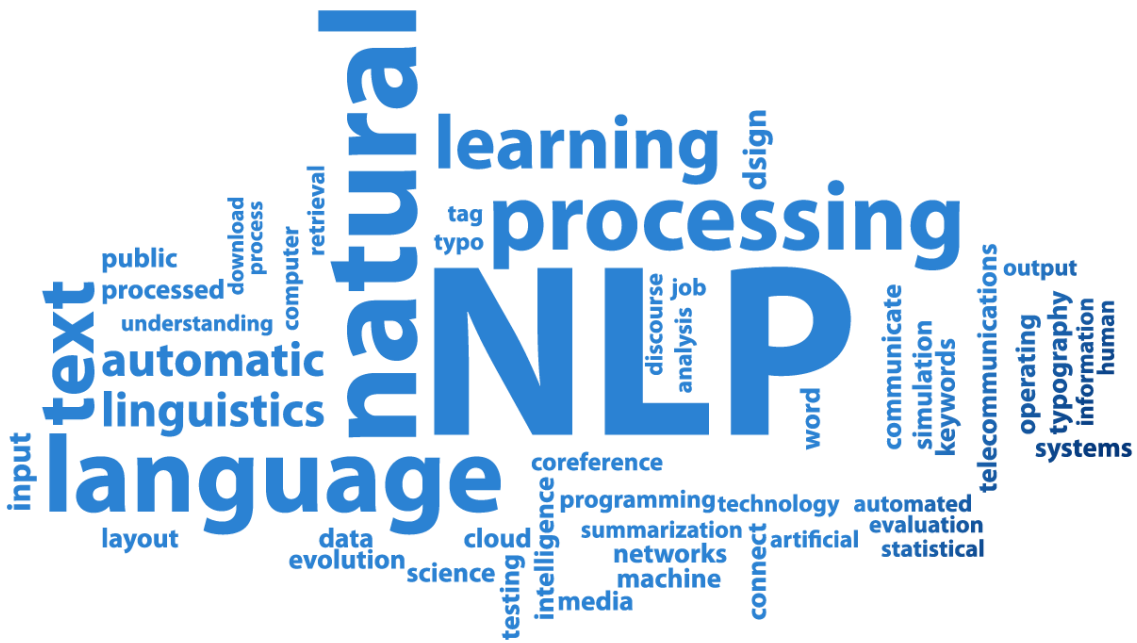
밑바닥부터 시작하는 딥러닝2

20213093 정현우

CONTENTS

1. 분산표현

자연어, 시소러스, 통계 기반 기법



자연어 처리란?

컴퓨터가 우리의 말을 알아듣게 만드는 것

우리가 평소에 쓰는 말 == 자연어

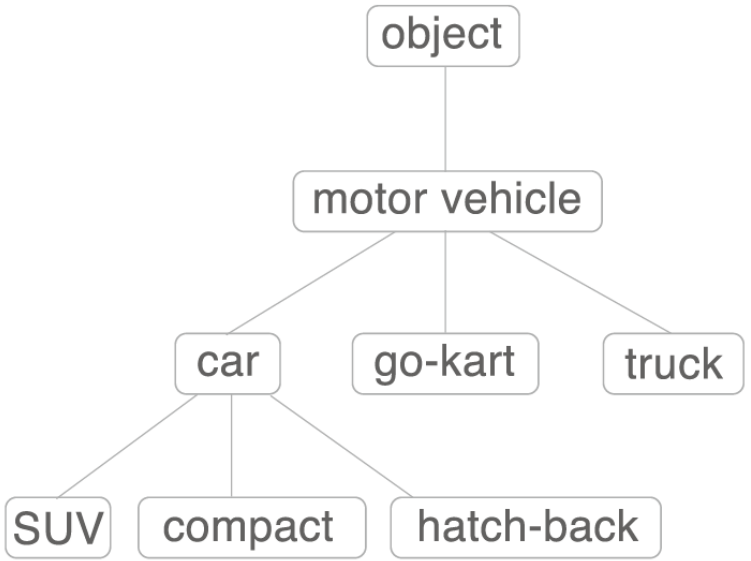
어떻게?

‘ 단어의 의미 ‘ 를 이해시킴

1. 분산표현

자연어, 시소러스, 통계 기반 기법

그림 2-2 단어들을 의미의 상 · 하위 관계에 기초해 그래프로 표현한다(문헌 [14]를 참고하여 그림).



시소러스

유의어 사전, 사람 손으로 만든다.

명백한 단점:

1. **시대 변화 대응 어려움**
2. **사람 쓰는 비용이 큼**
3. **단어의 미묘한 차이 표현할 수 없음**

1. 분산표현

자연어, 시소러스, 통계 기반 기법

```
    id_to_word
  ✓ 0.2s
{0: 'you', 1: 'say', 2: 'goodbye', 3: 'and', 4: 'i', 5: 'hello', 6: '.'}


    word_to_id
  ✓ 0.3s
{'you': 0, 'say': 1, 'goodbye': 2, 'and': 3, 'i': 4, 'hello': 5, '.': 6}
```

단어 전처리

컴퓨터가 단어들을 식별할 수 있게

숫자로 변환해준다.

1. 분산표현

자연어, 시소러스, 통계 기반 기법

동시발생 행렬

그림 2-7 모든 단어 각각의 맥락에 해당하는 단어의 빈도를 세어 표로 정리한다.

	you	say	goodbye	and	i	hello	.
you	0	1	0	0	0	0	0
say	1	0	1	0	1	1	0
goodbye	0	1	0	1	0	0	0
and	0	0	1	0	1	0	0
i	0	1	0	1	0	0	0
hello	0	1	0	0	0	0	1
.	0	0	0	0	0	1	0

“맥락”을 통해 컴퓨터가 단어의 의미를 알 수 있게 만드는 것이다.

이를 활용하여 단어 간의 유사도를 수치화 한다.

유사도는 코사인 유사도를 사용한다.

$$\text{Similarity}(p,q) = \cos\theta = \frac{p \cdot q}{\|p\|\|q\|} = \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^{\,2}}\sqrt{\sum_{i=1}^{n} q_i^{\,2}}}$$

1. 분산표현

자연어, 시소러스, 통계 기반 기법

$$\mathrm{PMI}(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)} = \log_2 \frac{\frac{C(x,y)}{N}}{\frac{C(x)}{N}\frac{C(y)}{N}} = \log_2 \frac{C(x,y)\cdot N}{C(x)C(y)}$$

통계 기반 기법 개선: 상호 정보량

발생횟수가 기준이라면?

많이 나온 단어는 중요하다고 판단할 수 있음

the, car, drive를 비교했을 때 발생횟수 기준이면 [the, car] 가 [car, drive] 보다 큰 연관성

PMI를 사용해 이를 보완함

1. 분산표현

자연어, 시소러스, 통계 기반 기법

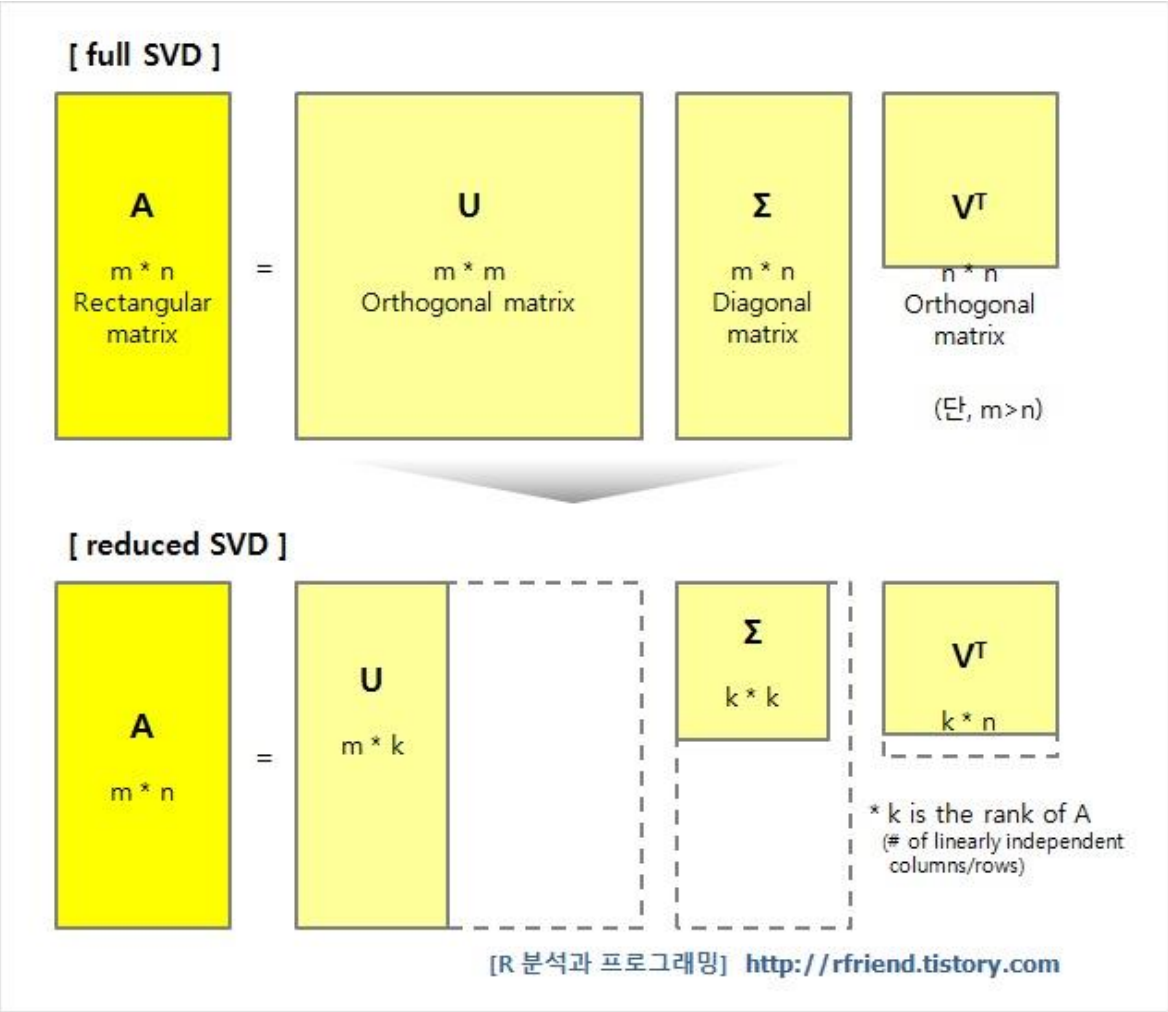
통계 기반 기법 개선: 상호 정보량

$$\mathrm{PMI}(\text{“the”},\text{“car”}) = \log_2 \frac{10 \cdot 10000}{1000 \cdot 20} \approx 2.32$$

$$\mathrm{PMI}(\text{“car”},\text{“drive”}) = \log_2 \frac{5 \cdot 10000}{20 \cdot 10} \approx 7.97$$

1. 분산표현

자연어, 시소러스, 통계 기반 기법



통계 기반 기법 개선: 차원 축소

SVD를 사용함.

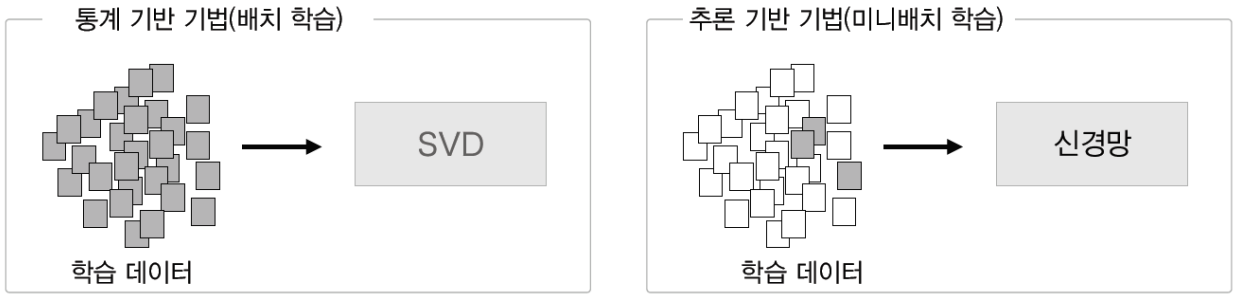
SVD는 numpy 함수를 사용하면 됨.

SVD를 구하고 U 벡터 중에서 일부만 사용하면 차원 축소됨.

2. word2vec

추론 기반 기법, word2vec, CBOW 구현

그림 3-1 통계 기반 기법과 추론 기반 기법 비교



추론 기반 기법

“맥락”을 통해 컴퓨터가 단어의 의미를 알 수 있게 만드는 것이다.

이를 활용하여 단어 간의 유사도를 수치화 한다.

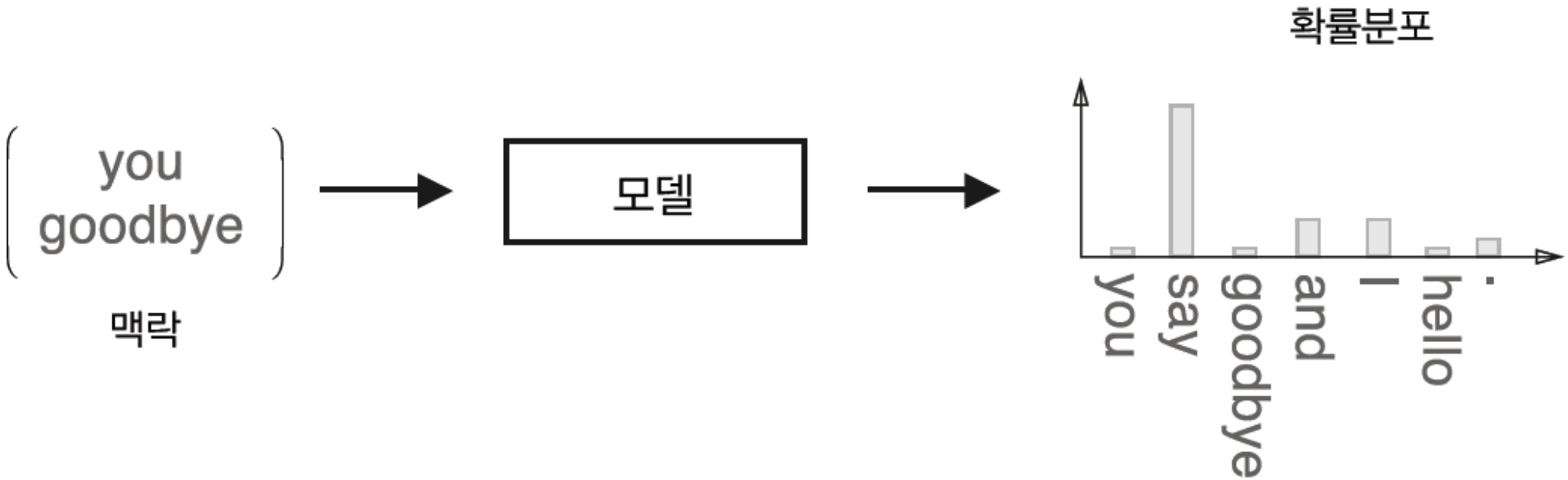
유사도는 코사인 유사도를 사용한다.

2. word2vec

추론 기반 기법, word2vec, CBOW 구현

추론 기반 기법

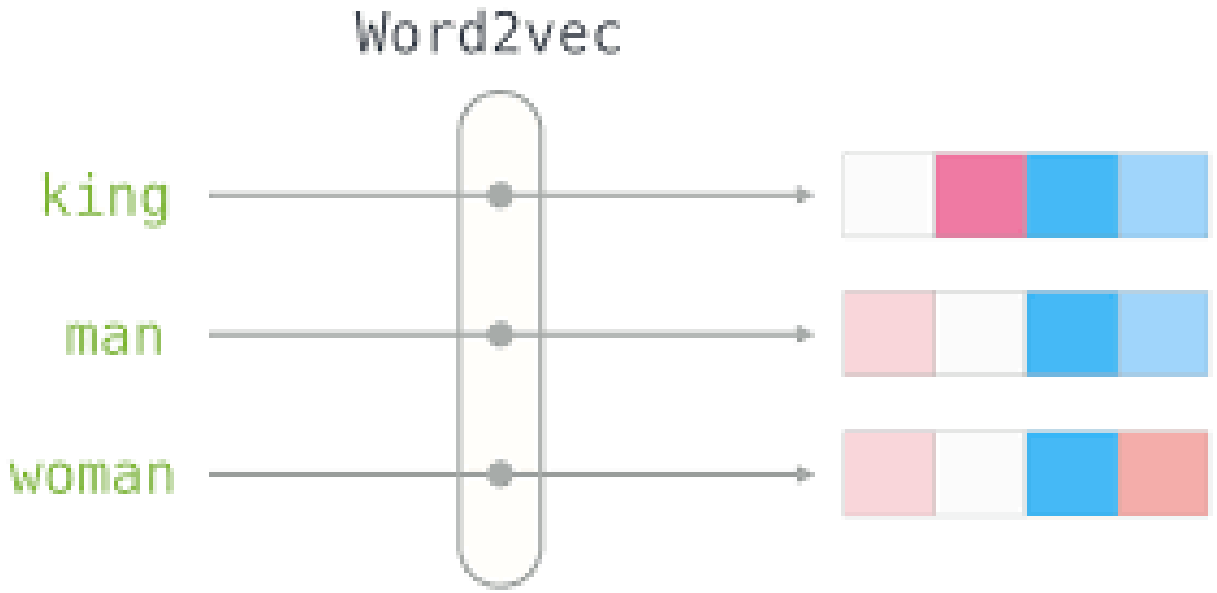
그림 3-3 추론 기반 기법: 맥락을 입력하면 모델은 각 단어의 출현 확률을 출력한다.



2. word2vec

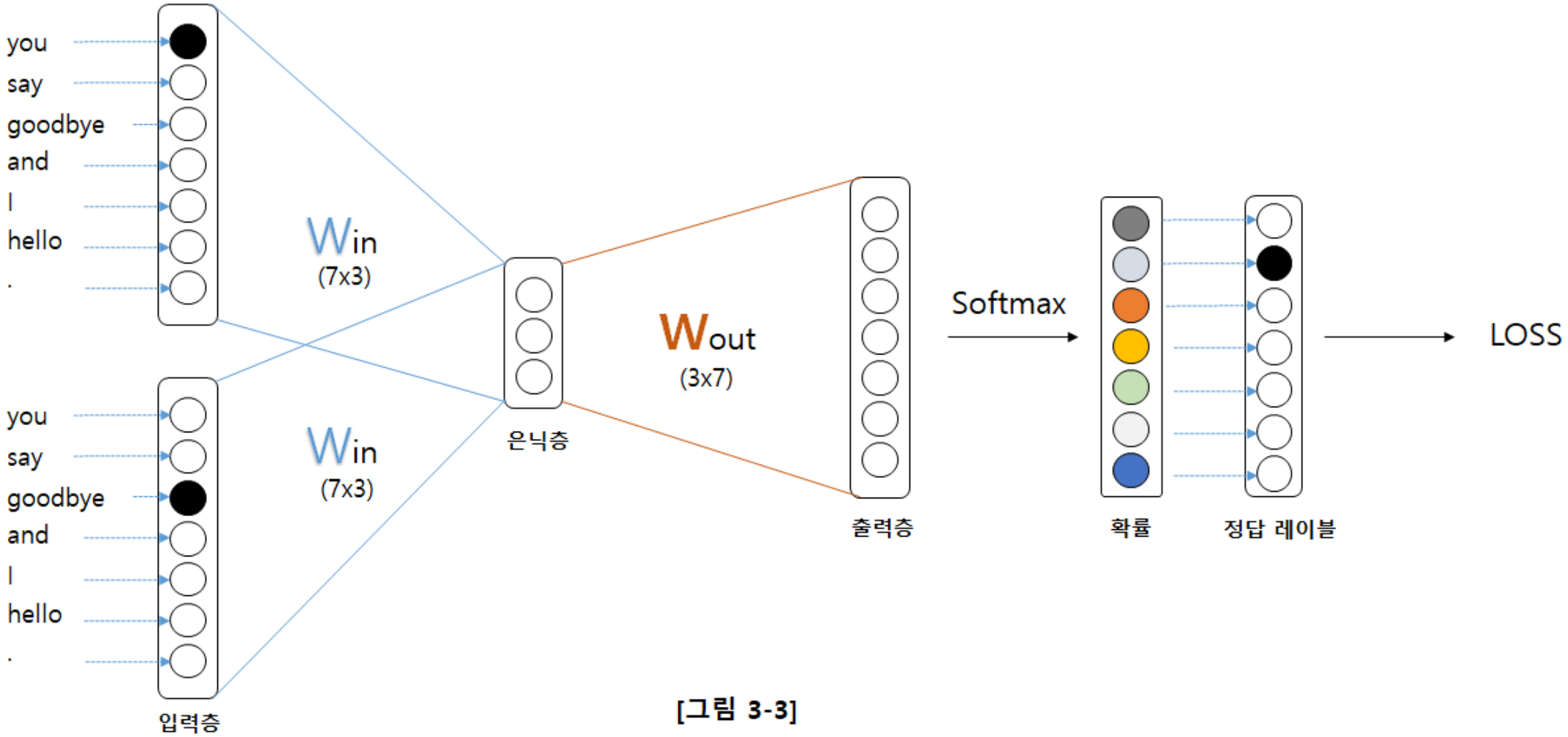
추론 기반 기법, word2vec, CBOW 구현

word2vec



2. word2vec

추론 기반 기법, word2vec, CBOW 구현

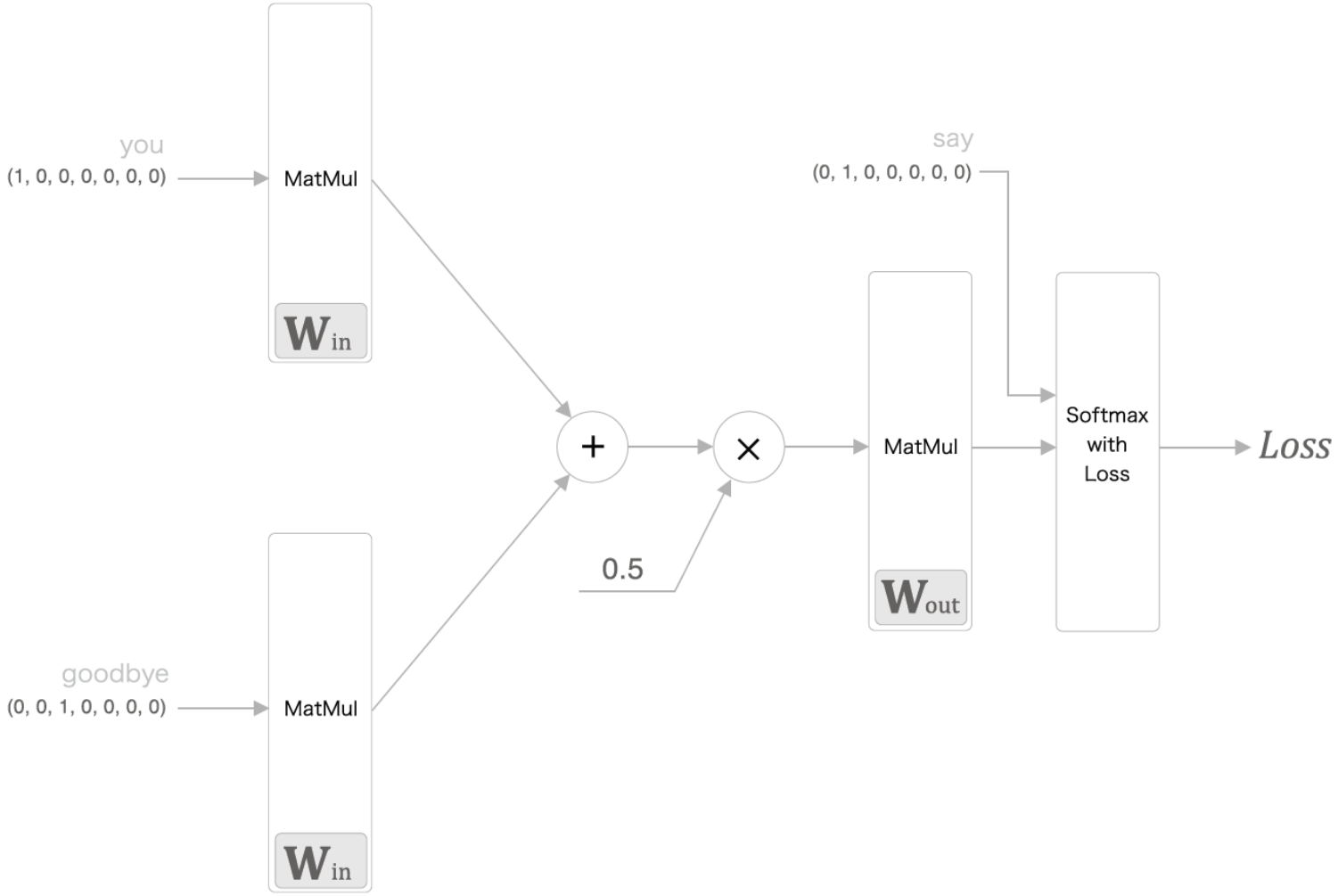


[그림 3-3]

2. word2vec

추론 기반 기법, word2vec, CBOW 구현

그림 3-14 Softmax 계층과 Cross Entropy Error 계층을 Softmax with Loss 계층으로 합침



2. word2vec

추론 기반 기법, word2vec, CBOW 구현

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

        # 왜 xavier 초기값 안 쓰지?
        W_in = 0.01 * np.random.randn(V, H).astype('f')
        W_out = 0.01 * np.random.randn(H, V).astype('f')

        self.in_layer0 = MatMul(W_in)
        self.in_layer1 = MatMul(W_in)
        self.out_layer = MatMul(W_out)
        self.loss_layer = SoftmaxWithLoss()

        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

        self.word_vecs = W_in
```

```python
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(score, target)
    return loss

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
```

2. word2vec

추론 기반 기법, word2vec, CBOW 구현

```python
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 = []
        for t in range(-window_size, window_size + 1):
            if t == 0:
                continue
            cs.append(corpus[idx + t])
        contexts.append(cs)

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

✓ 0.3s

맥락과 타겟

```
context
[[0 2]
 [1 3]
 [2 4]
 [3 1]
 [4 5]
 [1 6]
 [5 7]]
target
[1 2 3 4 1 5 6]
```

2. word2vec

추론 기반 기법, word2vec, CBOW 구현

```
import matplotlib.pyplot as plt

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()

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

✓ 0.5s

추론 기반 기법

모델 : SimpleCBOW

Optimizer : Adam

위도우 사이즈 : 1

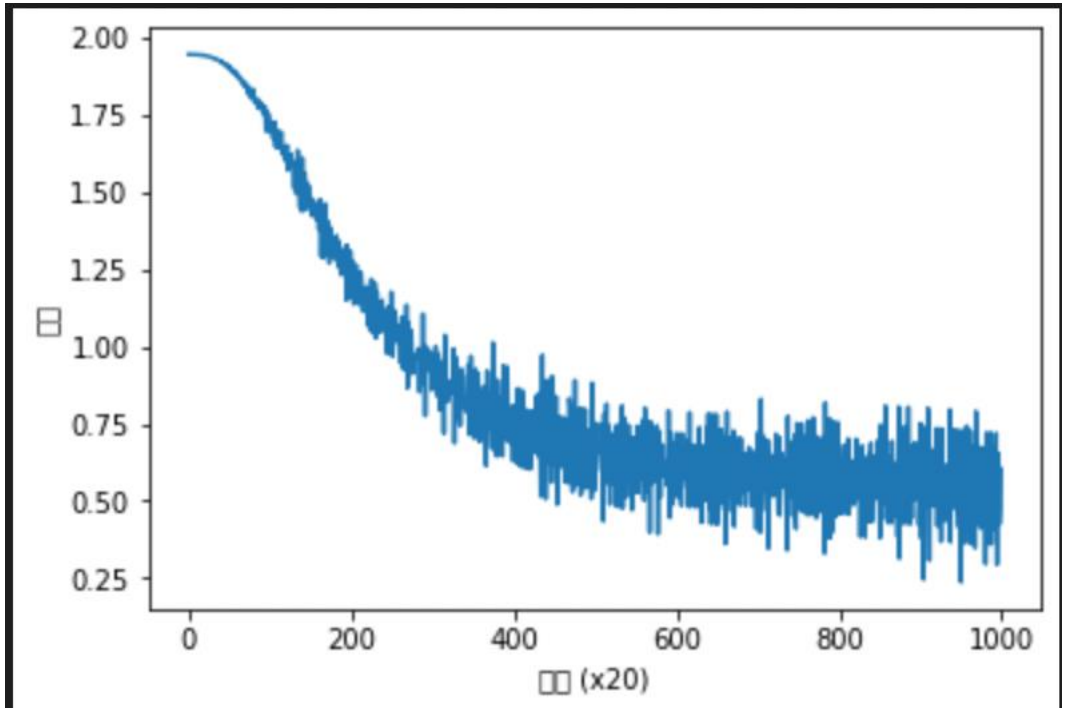
2. word2vec

추론 기반 기법, word2vec, CBOW 구현

```python
import matplotlib.pyplot as plt

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()

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

✓ 0.5s

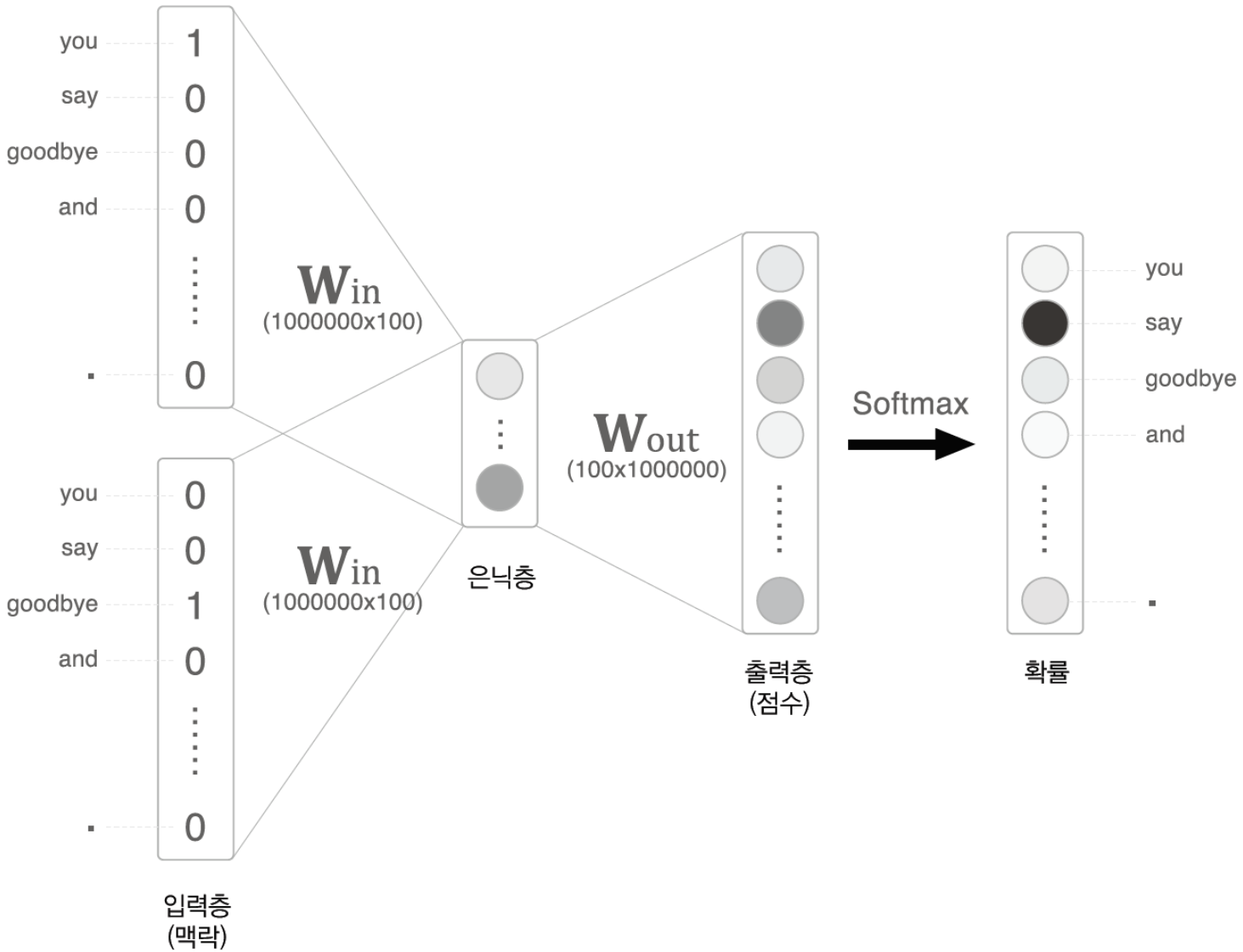
추론 기반 기법



3. 속도개선

임베딩 계층, 이진분류, 네거티브 샘플링

그림 4-2 어휘가 100만 개일 때를 가정한 CBOW 모델



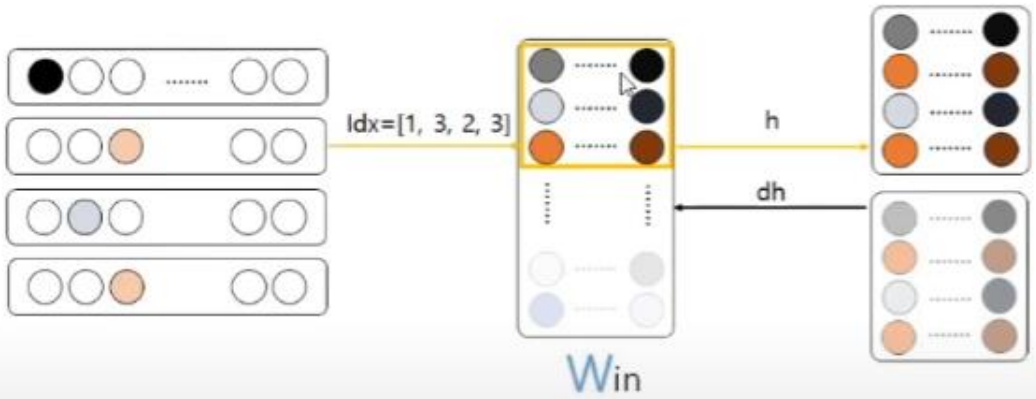
3. 속도개선

임베딩 계층, 이진분류, 네거티브 샘플링

Embedding 층

순전파 : 배치처리를 하므로 W_{in}을 해당 단어의 인덱스로 슬라이싱

역전파 : 미분의 각 행을 해당 단어 인덱스 위치에 더한다.

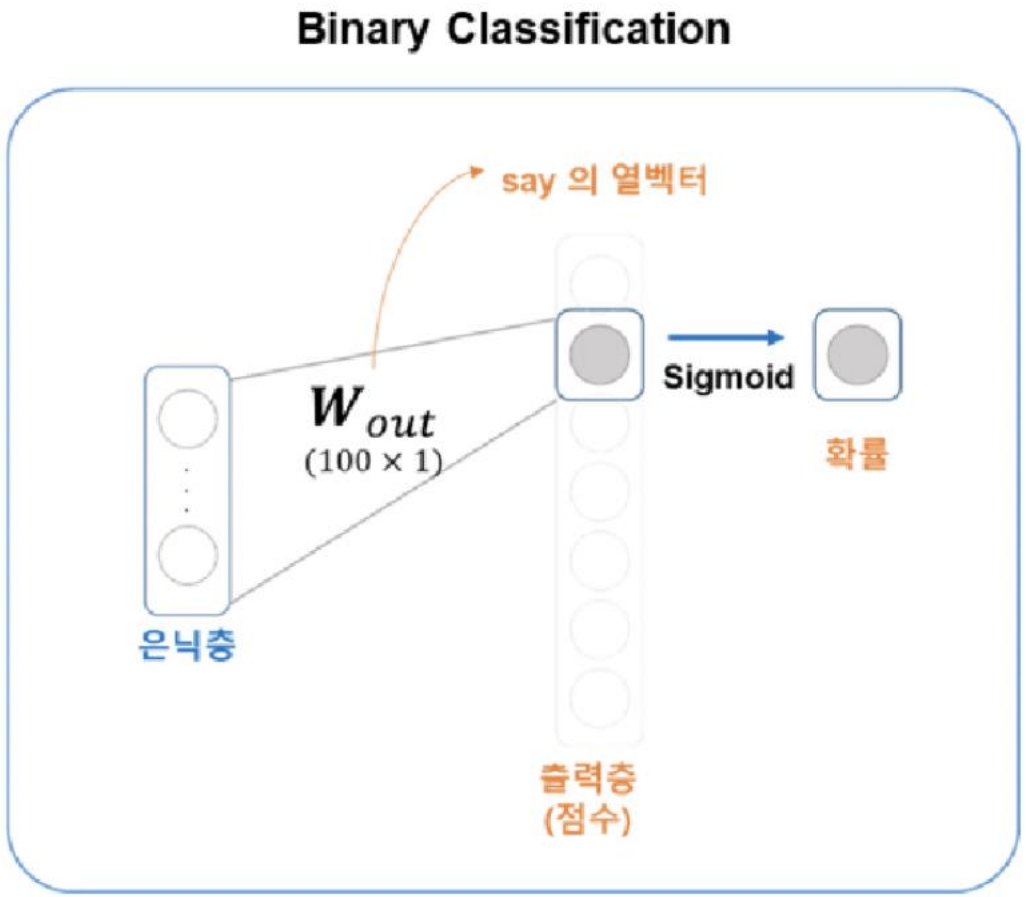


임베딩 계층

```python
class Embedding:
  def __init__(self, W):
    self.params = [W]
    self.grads = [np.zeros_like(W)]
    self.idx = None
  def forward(self, idx):
    W, = self.params
    self.idx= idx
    out = W[idx]
    return out
  def backward(self, dout):
    dW, = self.grads
    dW[...] = 0
    for i , word in enumerate(self.idx):
      dW[word_id] += dout[i]

    return None
```

3. 속도개선

임베딩 계층, 이진분류, 네거티브 샘플링

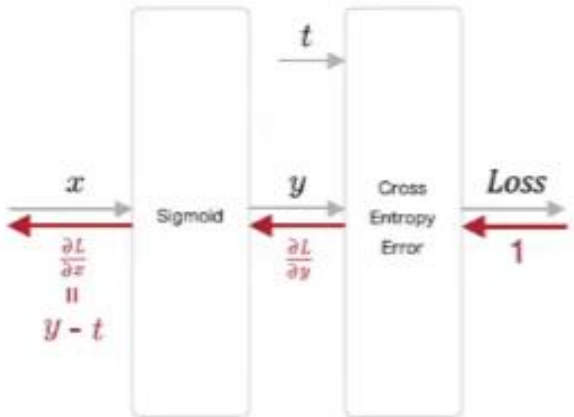


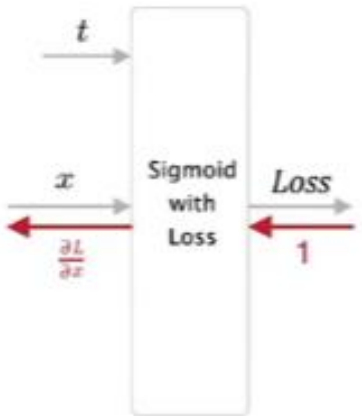
이진 분류로!

3. 속도개선

임베딩 계층, 이진분류, 네거티브 샘플링

Sigmoid with Loss 층





0 : No, 1 : Yes

$(1-y, y)$와 $(0,1)$의 cross entropy $= -\log y$

$(1-y, y)$와 $(1,0)$의 cross entropy $= -\log(1-y)$

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

$$\begin{aligned}\frac{dL}{dx} &= \frac{dL}{dy} \times \frac{dy}{dx} \\ &= -\left(\frac{t}{y} + \frac{t-1}{1-y}\right) \times y(1-y) \\ &= -t(1-y) + (1-t)y \\ &= -t + ty + y - ty \\ &= y - t\end{aligned}$$

3. 속도개선

임베딩 계층, 이진분류, 네거티브 샘플링

네거티브 샘플링

입력과 레이블의 변화

입력1	입력2	레이블
cat	The	1
cat	fat	1
cat	sat	1
cat	on	1

Negative Sampling

입력1	입력2	레이블
cat	The	1
cat	fat	1
cat	**pizza**	**0**
cat	**computer**	**0**
cat	sat	1
cat	on	1

단어 집합에서 랜덤으로
선택된 단어들을
레이블 0의 샘플로 추가.

3. 속도개선

임베딩 계층, 이진분류, 네거티브 샘플링

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

        # 왜 xavier 초기값 안 쓰지?
        W_in = 0.01 * np.random.randn(V, H).astype('f')
        W_out = 0.01 * np.random.randn(H, V).astype('f')

        self.in_layer0 = MatMul(W_in)
        self.in_layer1 = MatMul(W_in)
        self.out_layer = MatMul(W_out)
        self.loss_layer = SoftmaxWithLoss()

        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

        self.word_vecs = W_in
```

개선판 구현

```python
class CBOW:
    def __init__(self, vocab_size, hidden_size, window_size, corpus):
        V, H = vocab_size, hidden_size
        self.in_layers = []
        for i in range(2 * window_size):
            layer = Embedding(W_in)
            self.in_layers.append(layer)
        self.ns_loss = NegativeSamplingLoss(W_out, corpus, power=0.75, sample_size=5)

        layers = self.in_layers + [self.ns_loss]
        self.params, self.grads = [], []
        for layer in layers:
            self.params += layer.params
            self.grads += layer.grads

        self.word_vecs = W_in
```

3. 속도개선

임베딩 계층, 이진분류, 네거티브 샘플링

```
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(score, target)
  return loss

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
```

개선판 구현

```
def forward(self, contexts, target):
  h = 0
  for i, layer in enumerate(self.in_layers):
    h += layer.forward(contexts[:, i])
  h += 1 / len(self.in_layers)
  loss = self.ns_loss.forward(h, target)
  return loss
def backward(self, dout = 1):
  dout = self.np_loss.backward(dout)
  dout += 1 / len(self.in_layers)
  for layer in self.in_layers:
    layer.backward(dout)
  return None
```

Thank you