밑바닥부터 시작하는 딥러닝 2
< Chapter 3 Word2vec > 발제 자료

2020. 01. 18

김우정

통계 기반 기법 vs. 추론 기반 기법

통계 기반 기법의 문제점 : 대규모 말뭉치를 다룰 때 $O(n^3)$ 의 비용이 든다.

(review : 말뭉치 → 동시발생 행렬 → SVD 적용 → 단어의 분산 표현)

(100만개 어휘 말뭉치 → 100만 X 100만의 동시발생 행렬 → 100만X100만X100만의 비용…)

통계 기반 기법 : 학습 데이터를 한꺼번에 처리 (배치 학습)

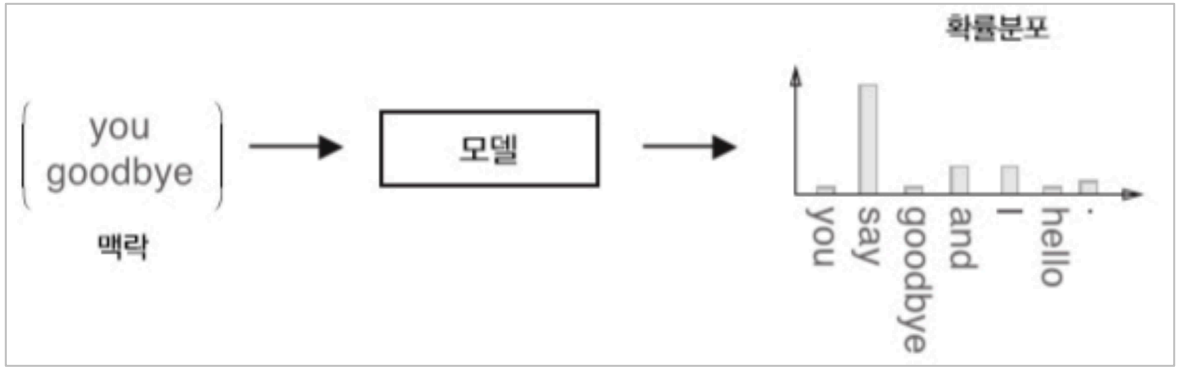
추론 기반 기법 : 학습 데이터의 일부를 사용하여 순차적으로 학습 (미니배치 학습)

미니배치 학습은 계산량이 큰 작업을 처리할 때 효율적. 데이터를 작게 나눠 학습하기 때문에 연산 가능. 또한 병렬 계산도 가능하게 하여 학습 속도 또한 높일 수 있다.

추론 기반 기법이란?

you ? goodbye and I say hello.

주변 단어들을 사용해 ?에 들어갈 단어를 추론하는 작업



추론 기반 기법에서는 신경망 모델을 이용한다.

모델은 위와 같은 추론 문제를 반복해서 풀면서

단어의 출현 패턴을 학습하여 **각 단어 별로 출현 확률을 출력**한다.

신경망에서의 단어 처리

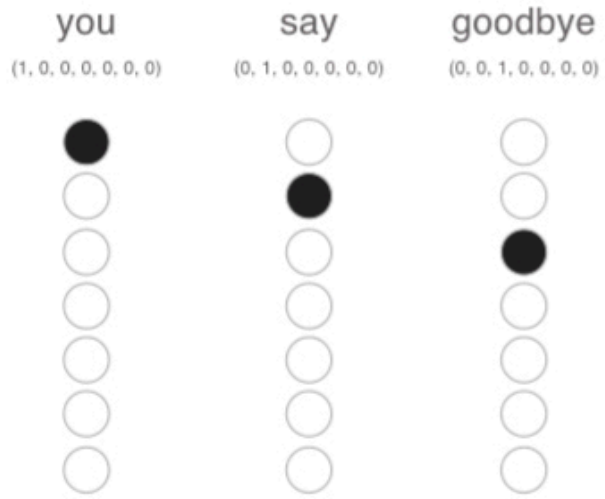
신경망을 이용해 단어를 처리하기 위해서는 단어를 고정 길이의 벡터로 변환해야 한다. (신경망의 입력층에서 뉴런의 수를 고정해야 함)

→ **원핫 표현 (one-hot encoding) 이용**

음식 메뉴	ID	One-Hot Encoding
참치김치찌개	1	[1, 0, 0, 0, 0, 0]
스팸김치찌개	2	[0, 1, 0, 0, 0, 0]
생고기김치찌개	3	[0, 0, 1, 0, 0, 0]

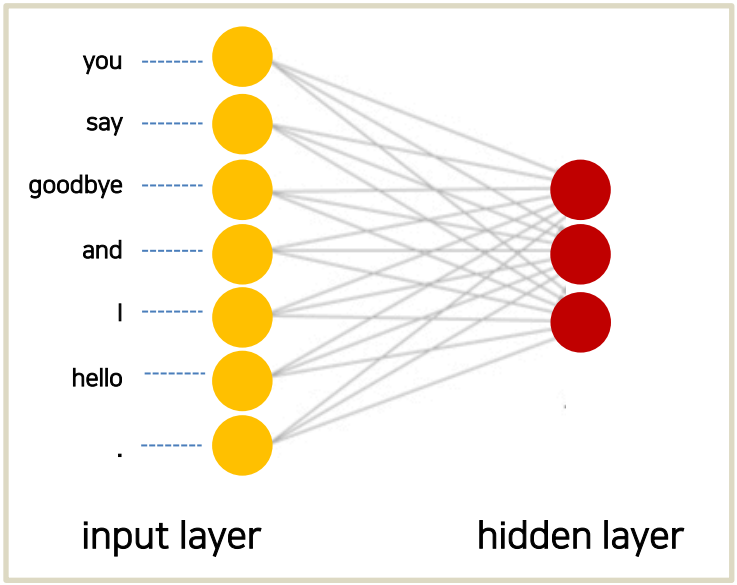
출처 : https://linewalks.com/archives/6408

you say goodbye and I say hello.



- 총 어휘 수(7) 만큼의 벡터 사이즈
- 각 단어마다 아이디에 해당하는 인덱스만 1, 나머지는 0인 벡터를 가짐

추론 기반 기법과 신경망




```
c = np.array([1, 0, 0, 0, 0, 0, 0])
W = np.random.randn(7,3)
h = np.matmul(c,W)
print(h)

# [0.79430496 1.43159989 0.45691755]
```

- 완전연결계층에 의한 원핫 표현 형식의 단어 변환
- 총 7개의 단어를 가진 문장에서 한 단어가 입력으로 들어가고, 이는 3개의 은닉층 뉴런을 통해 변환되어짐
- 인코딩 과정

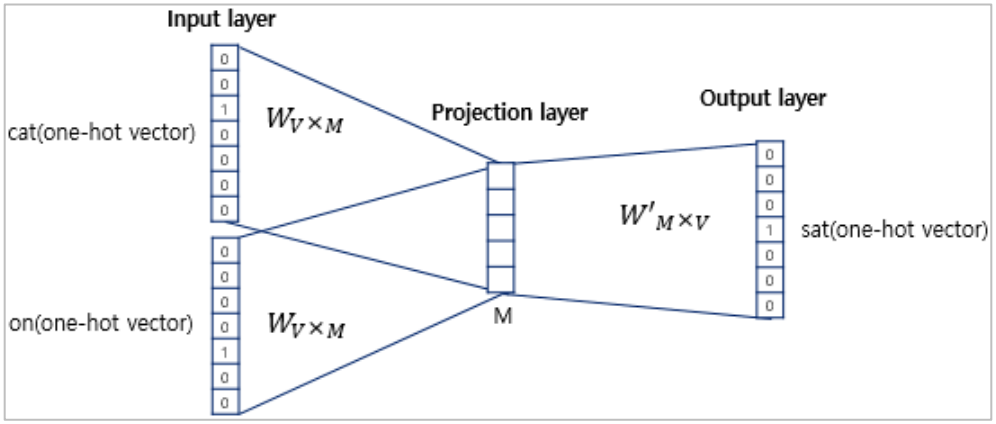
CBOW 모델

CBOW continuous bag-of-words

you say goodbye and I say hello.

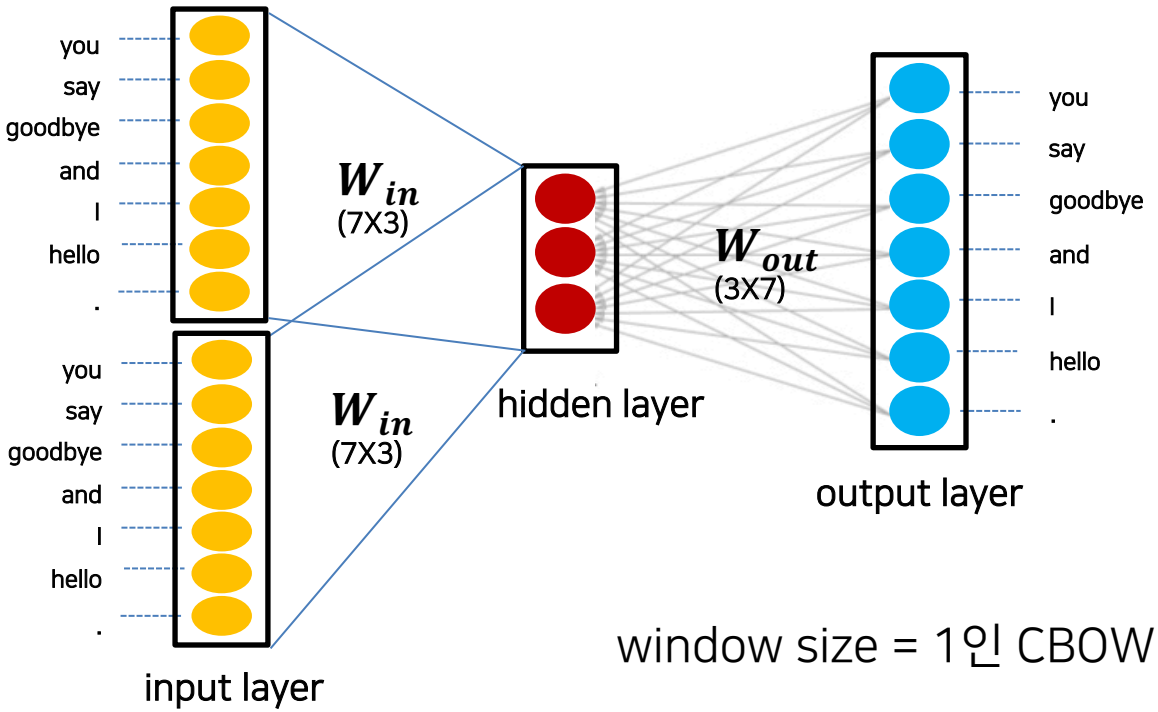
CBOW 모델은 맥락으로부터 타깃(target)을 추측하는 용도의 신경망이다.

- 타깃 : 중앙 단어
- 맥락 : 중앙 단어의 주변 단어들



즉, 신경망 모델의 입력값으로 맥락이 들어가고 타깃이 출력됨.

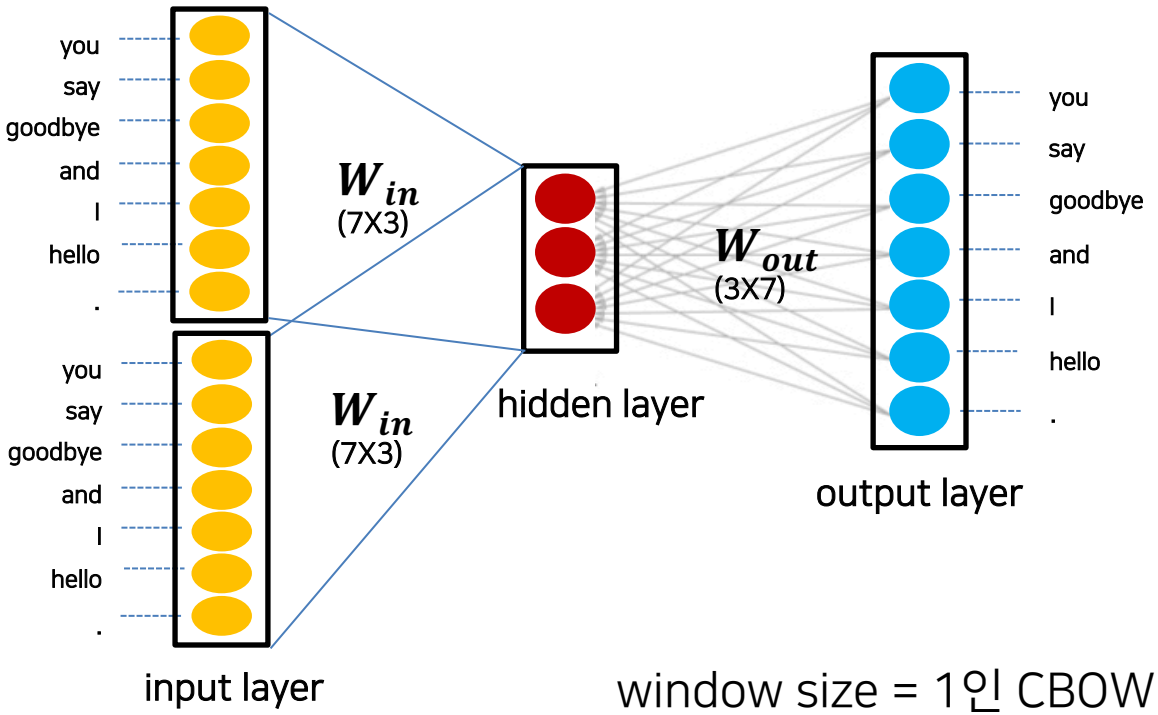
CBOW 모델



window size = 1인 CBOW 모델의 신경망 예시

- 입력층이 2개 있고, 은닉층을 거쳐 출력층에 도달
- 이때 <입력→은닉> 에의 가중치는 W_{in} <은닉→출력> 에의 가중치는 W_{out}
- 두 입력층에서는 가중치를 공유한다.

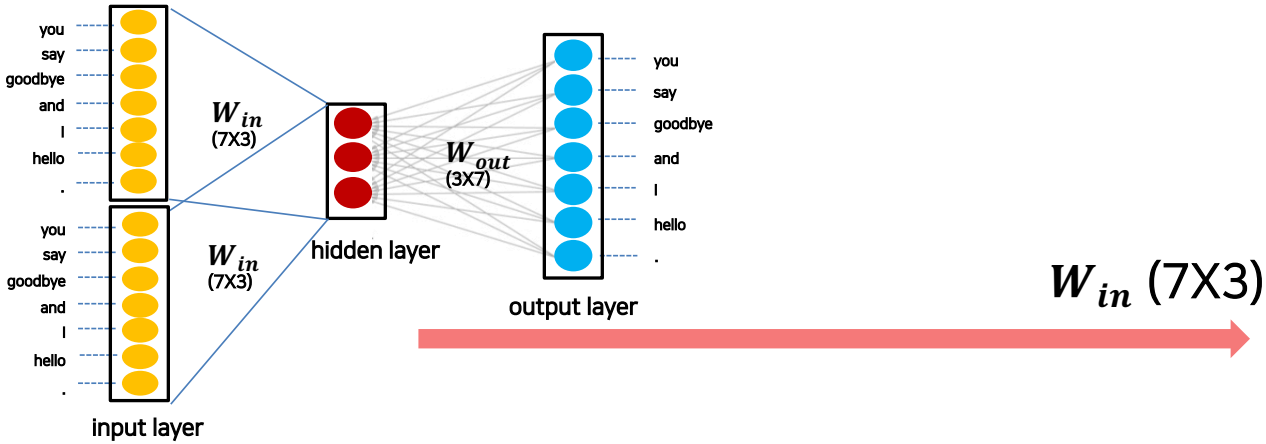
CBOW 모델



window size = 1인 CBOW 모델의 신경망 예시

- 입력층 : 맥락에 포함시킬 단어가 N개라면 입력층도 N개이다.
- 은닉층 : 입력층 전체를 평균낸 값. (h1 + h2 / 2)
- 출력층 : (예시 뉴런의 개수 총 7개). 뉴런 하나하나가 각각의 단어에 대응된다. 출력층 뉴런은 각 단어의 '점수'를 뜻하며 값이 높을수록 대응 단어의 출현 확률도 높아진다.

CBOW 모델



- (예시 그림에서) <입력층→은닉층> 완전연결계층의 가중치 W_{in} 은 7X3 행렬이며, 이 가중치의 각 행이 해당 단어의 분산 표현이다.
- **은닉층의 뉴런 수를 입력층의 뉴런 수보다 적게 하는 것이 중요하다.** 이렇게 해야 은닉층에는 단어 예측에 필요한 정보만을 응축해서 담게 되며 결과적으로 밀집된 벡터 표현을 얻을 수 있다.
- 학습을 진행할수록 **맥락에서 출현하는 단어(target)를 잘 추측하는 방향으로 이 분산 표현들이 갱신**된다.

CBOW 모델의 추론 처리

```
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)

print(s)
# [ 0.11758873  0.00227049 -0.75579113 -0.22458675
# -0.3500984  -0.63682774  -0.63111001]
```

샘플 맥락 데이터.
총 어휘 단어 수는 7개, windowsize=1이라고 가정한다.

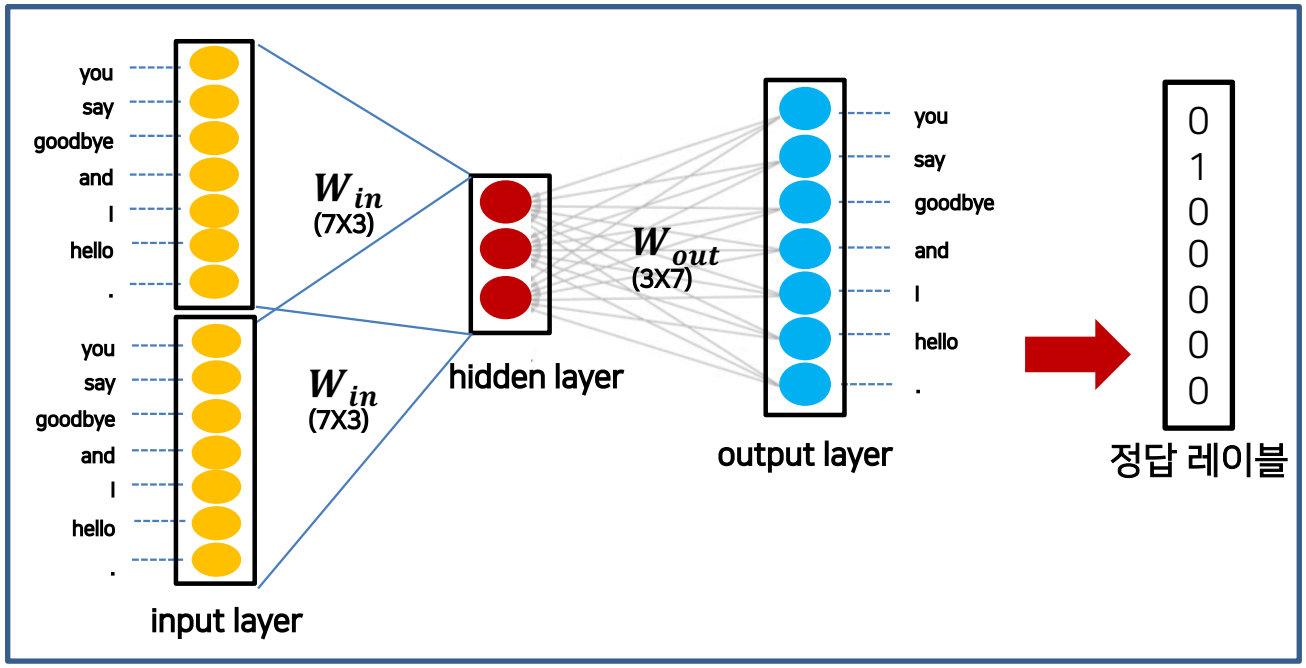
은닉층의 뉴런 수는 3개
가중치는 W_in 과 W_out 두 개만을 활용

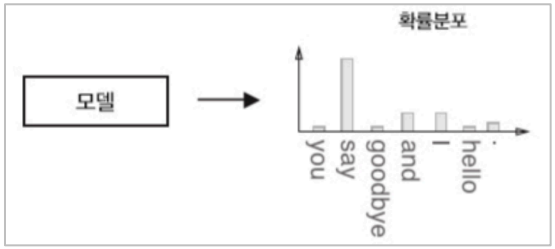
맥락 단어의 개수만큼 입력층 레이어 생성
이때, 가중치는 W_in을 공유한다

입력층 레이어의 평균값이 은닉층으로 들어가게됨

단어 별 점수 (소프트맥스 적용 전)

CBOW 모델의 학습



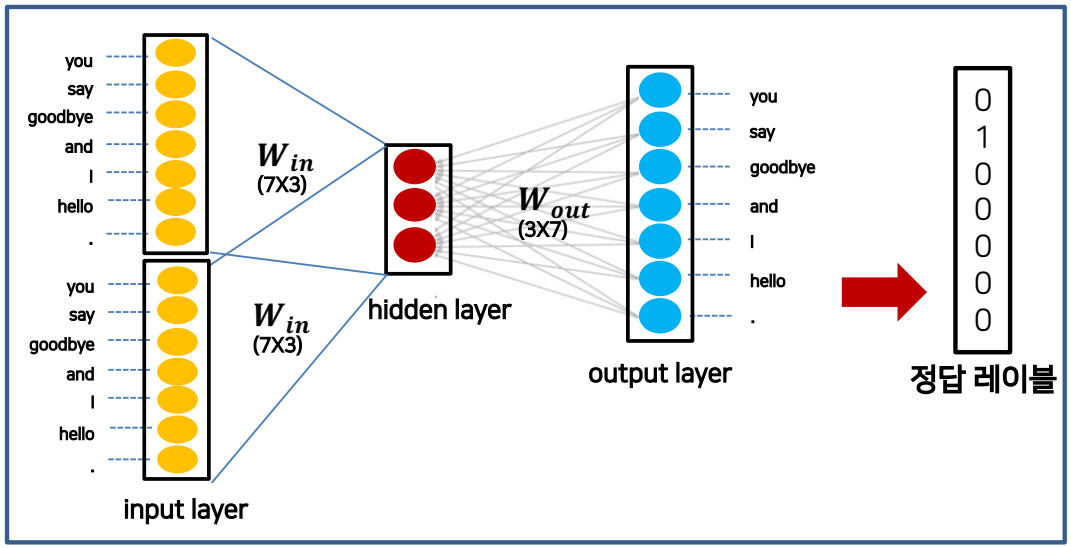


CBOW 모델의 출력값은 맥락 (주변 단어)이 주어졌을 때
그 중앙에 **어떤 단어가 출현할 확률이 높은 지**를 나타낸다.

모델은 이후 실제 타깃에 대한 확률값을 높이는 방향으로 학습된다.

그 결과, 가중치 W_{in} 와 W_{out} 에 단어의 출현 패턴을 파악한 벡터가 담김

W_in vs. W_out



입력 측 완전연결계층의 가중치 W_{in} 와 출력 측 완전연결계층의 가중치 W_{out} 둘 다 각 단어의 분산 표현을 가지고 있다.

3가지 활용 방안

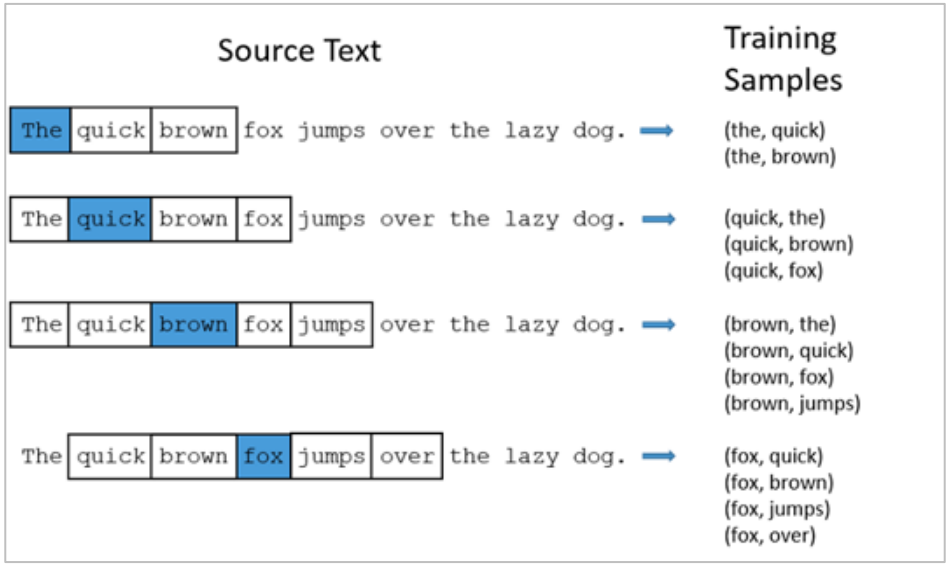
A. 입력 측의 가중치만 이용 - W_{in} → 가장 보편적으로 쓰이는 방법

B. 출력 측의 가중치만 이용 - W_{out}

C. 양쪽 가중치 모두 이용 - (W_{in}, W_{out})

(참고) GloVe에서는 두 가중치를 더했을 때 더 좋은 결과를 얻음

학습 데이터 준비



말뭉치 데이터를 각 타깃 단어 별로 생성해야함!

window size가 1일 경우, 각 타깃 단어마다 문맥 단어 2개 보유

window size가 2일 경우, 각 타깃 단어마다 문맥 단어 4개 보유

학습 데이터 준비

```
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: '.'}
```

먼저, 전처리 과정을 거쳐 각 단어에 대한 인덱스를 부여한다.

**자연어를 다루기 전에 가장 먼저 해야하는 작업

학습 데이터 준비

```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 = [] # context_per_target
        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)
```

맥락의 개수가 채워지지 않는 양 끝 단어는 타깃에서 제외

target=0을 기준으로 window_size만큼 좌우

맥락과 타깃을 만드는 함수

contexts	target
[[0 2]	[1
[1 3]	2
[2 4]	3
[3 1]	4
[4 5]	1
[1 6]]	5]

맥락의 형상: (6, 2) 타깃의 형상: (6,)

학습 데이터 준비

contexts

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

target

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

맥락의 형상: (6, 2) 타깃의 형상: (6,)

contexts

```
([[[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]]],
```

target

```
[[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]],
```

원핫 표현으로 변환

CBOW 모델 구현

```python
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')

        # 계층 생성
        self.in_layer0 = MatMul(W_in)
        self.in_layer1 = MatMul(W_in)
        self.out_layer = MatMul(W_out)
        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 단어의 분산 표현을 저장한다.
        self.word_vecs = W_in
```

인수로 어휘 수와 은닉층의 뉴런 수를 받는다.

입력 측의 맥락을 처리하는 MatMul 계층은 contexts의 개수만큼 생성 (즉, window_size*2 만큼 생성)

출력층의 MatMul 계층 하나

SoftmaxWithLoss 계층 하나

인스턴스 변수에 단어의 분산 표현을 저장한다.

CBOW 모델 구현

신경망의 순전파 메서드
: 인수로 맥락과 타깃을 받아서 loss를 반환

```
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
```

contexts.shape = (6, 2, 7)
0번째 차원의 원소 수 : 미니배치의 수
1번째 차원의 원소 수 : 맥락의 윈도우 크기,
2번째 차원 : 원 핫 벡터 크기

target.shape = (6, 7)

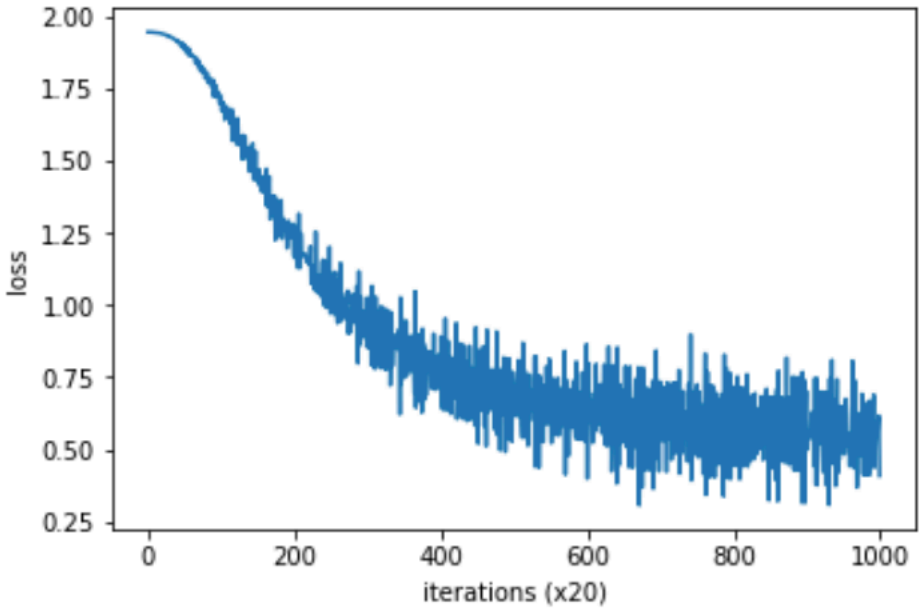
신경망의 역전파 메서드
: 기울기를 순전파 때와는 반대 방향으로 전파 ; 학습의 핵심

```
def backward(self, dout=1): # 1에서 시작
    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
```

CBOW 모델 학습

- window_size = 1
- hidden_size = 5
- batch_size = 3 : 몇 개의 샘플로 가중치를 갱신할 것인지 지정
- max_epoch = 1000 : 학습 반복 횟수

```
| epoch 1 |  iter 1 / 2 | time 0[s] | loss 1.95
| epoch 2 |  iter 1 / 2 | time 0[s] | loss 1.95
| epoch 3 |  iter 1 / 2 | time 0[s] | loss 1.95

| epoch 192 |  iter 1 / 2 | time 0[s] | loss 1.27
| epoch 193 |  iter 1 / 2 | time 0[s] | loss 1.29
| epoch 194 |  iter 1 / 2 | time 0[s] | loss 1.27

| epoch 995 |  iter 1 / 2 | time 0[s] | loss 0.58
| epoch 996 |  iter 1 / 2 | time 0[s] | loss 0.54
| epoch 997 |  iter 1 / 2 | time 0[s] | loss 0.55
| epoch 998 |  iter 1 / 2 | time 0[s] | loss 0.59
| epoch 999 |  iter 1 / 2 | time 0[s] | loss 0.62
| epoch 1000 |  iter 1 / 2 | time 0[s] | loss 0.41

| epoch 304 |  iter 1 / 2 | time 0[s] | loss 0.90
| epoch 305 |  iter 1 / 2 | time 0[s] | loss 0.99
| epoch 306 |  iter 1 / 2 | time 0[s] | loss 0.79
| epoch 307 |  iter 1 / 2 | time 0[s] | loss 1.07

| epoch 636 |  iter 1 / 2 | time 0[s] | loss 0.79
| epoch 637 |  iter 1 / 2 | time 0[s] | loss 0.62
| epoch 638 |  iter 1 / 2 | time 0[s] | loss 0.62
| epoch 639 |  iter 1 / 2 | time 0[s] | loss 0.58
```



CBOW 모델 학습

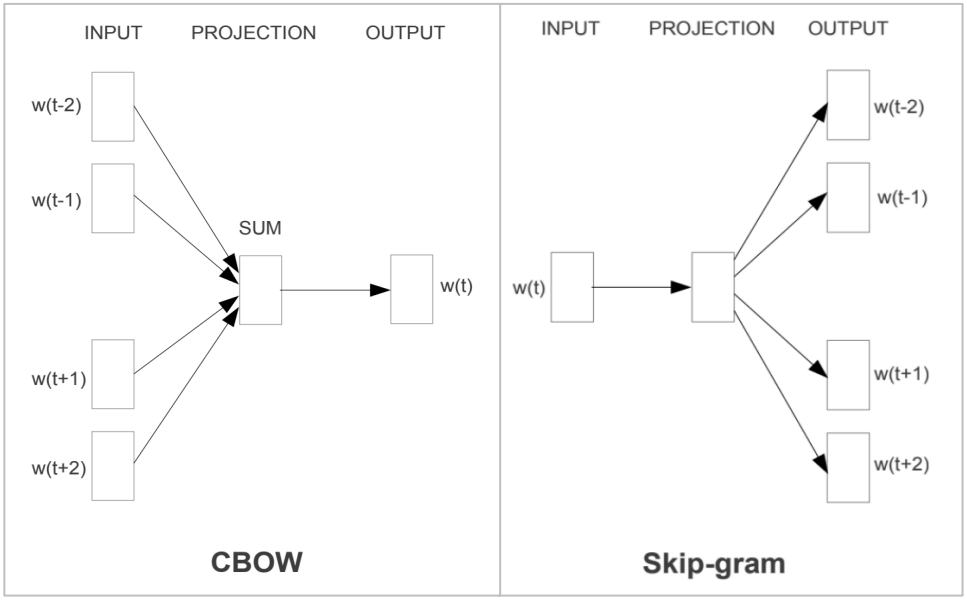
```
# 학습이 끝난 뒤의 가중치 매개변수는 인스턴스 변수 word_vecs에 담겨있음
word_vecs = model.word_vecs
for word_id, word in id_to_word.items():
    print(word,word_vecs[word_id],end = '\n\n')
```

```
you [ 1.3351159 -1.1248811  1.1296217 -1.1195978 -1.0375124]

say [-0.01697864  1.1917418  -1.1481713   1.1842325   1.2066556 ]

goodbye [ 0.61621106 -0.75953335  0.81836444 -0.70477396 -0.8977362 ]

and [-1.8688438   1.0109028  -1.097344    0.9846056   0.96556264]

i [ 0.6094434  -0.76561135  0.8301478  -0.71917397 -0.8725316 ]

hello [ 1.3669877 -1.1133312  1.1342381 -1.1258147 -1.0214459]

. [ 1.6230313   1.0678545  -0.64506924  1.0918348   1.0815095 ]
```

학습이 끝난 뒤의 가중치 매개변수에 각 단어에 대한 분산 표현이 저장됨

skip-gram 모델

skip-gram은 CBOW에서 다루는 맥락과 타깃을 역전시킨 모델이다.



CBOW 모델
맥락이 여러 개 있고, 그 여러 맥락으로부터 중앙의 단어(타깃)을 추측한다.

you ? goodbye and I say hello.

skip-gram 모델
중앙의 단어(타깃)으로부터 주변의 여러 단어(맥락)을 추측한다.

? say ? and I say hello.

CBOW < skip-gram

- 단어 분산 표현의 정밀도 면에서 skip-gram 모델의 결과가 더 좋은 경우가 많음. 특히, 말뭉치가 커질수록 이러한 경향성은 커진다.

- 반면, 학습 속도 면에서는 CBOW 모델이 더 빠르다. skip-gram 모델은 손실을 맥락의 수만큼 구해야 해서 계산 비용이 그만큼 커지게 된다.

- (작가생각) skip-gram 모델은 하나의 단어로부터 그 주변 단어를 예측. skip-gram 모델이 CBOW 모델보다 더 어려운 문제를 예측한다고 볼 수 있다. 더 어려운 상황에서 학습하는 만큼 skip-gram 모델로부터 생성된 단어의 분산 표현이 더 정교하고 뛰어날 가능성이 커진다고도 생각해볼 수 있다.

CBOW vs. skip-gram

```python
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')

        # 계층 생성
        self.in_layer0 = MatMul(W_in)
        self.in_layer1 = MatMul(W_in)
        self.out_layer = MatMul(W_out)
        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 단어의 분산 표현을 저장한다.
        self.word_vecs = W_in

    # 신경망의 순전파 메서드
    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

    # 신경망의 역전파
    def backward(self, dout=1): # 1에서 시작
        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
```

CBOW와는 반대로 출력층에 맥락의 개수만큼 레이어 생성

모든 맥락의 손실을 합친 것이 skip-gram의 손실

```python
class SimpleSkipGram:
    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')

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

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

        # 단어의 분산 표현 저장
        self.word_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
```

skip-gram에서도 동일하게 입력층 가중치 사용

통계 기반 vs. 추론 기반

1. 어휘에 추가할 새 단어가 생겨서 단어의 분산 표현을 갱신해야 하는 상황

통계 기반 기법 :

계산 처음부터 다시 해야 한다. 단어의 분산 표현을 조금만 수정하여도, 동시발생 행렬을 다시 만들고 SVD를 수행하는 일련의 작업을 다시 해야 한다.

추론 기반 기법 :

매개변수를 다시 학습할 수 있다. 지금까지 학습한 가중치를 초깃값으로 사용해 다시 학습하면 되는데 (신경망 모델의 특성), 이런 특성 덕분에 기존에 학습한 경험을 해치지 않으면서 단어의 분산 표현을 효율적으로 갱신할 수 있다.

통계 기반 기법 vs. 추론 기반 기법

2. 두 기법으로 얻는 단어의 분산 표현의 성격이나 정밀도 측면

통계 기반 기법 :

주로 단어의 유사성이 인코딩된다.

추론 기반 기법 : 단어의 유사성 뿐만 아니라, 그리고 복잡한 단어 사이의 패턴도 파악되어 인코딩된다.

BUT, 실제로 단어의 유사성을 정량적으로 평가한 연구 결과, 추론 기반과 통계 기반 기법의 우열이 드러나지 않음!

(기타)

두 기법은 사실 상 연관되어 있다. 구체적으로는 skip-gram + negative sampling을 이용한 모델은 동시발생 행렬에 특수한 행렬 분해를 적용한 것과 같다.

GloVe는 추론 기반 기법 + 통계 기반 기법. 말뭉치 전체의 통계 정보를 손실 함수에 도입해 미니배치 학습을 하는 것.

EndOfPresentation

Q & A
Thank you!