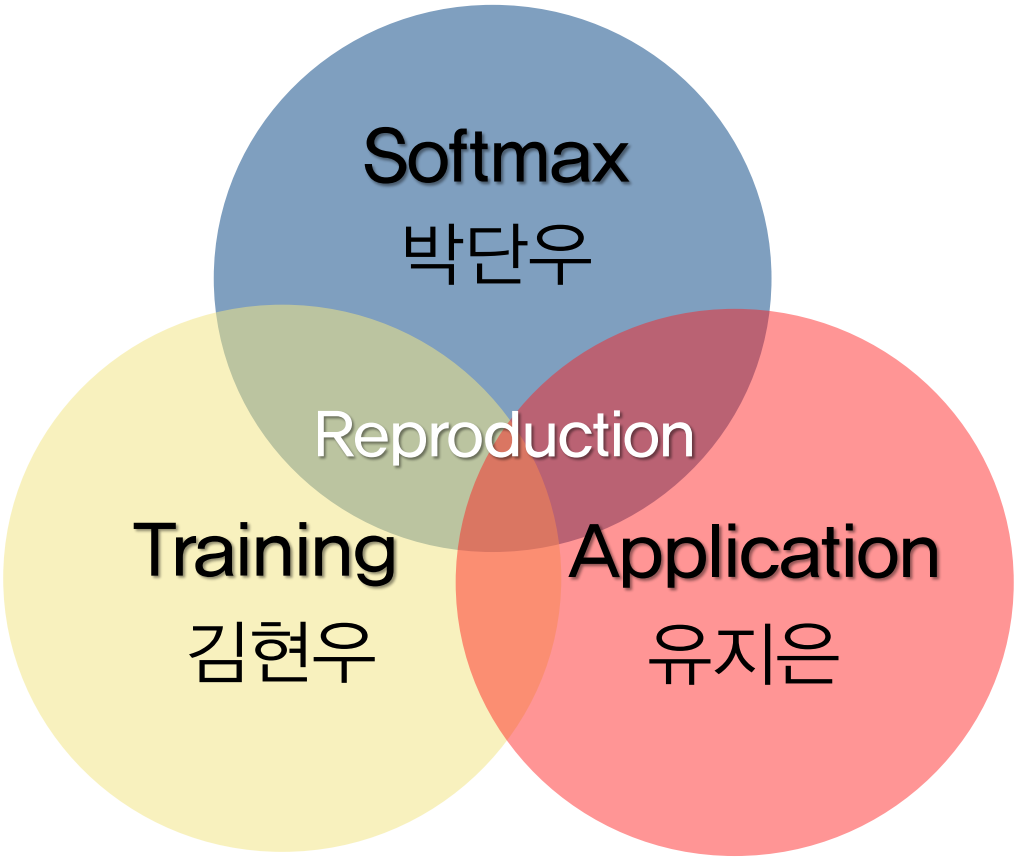
word2vec

Reproduction and Application

2팀

김현우 박단우 유지은

Team Role



1

우리가 이해한 word2vec

복잡하고 어렵게 느껴지던 word2vec을
저희가 이해하기까지의 과정을
설명하고자 합니다.

2

Softmax 중심

word2vec을 구현하면서 들었던 의문점들과 실험

왜 이렇게 구현하는 것인지 의문이
들었던 사항들에 대한 생각과 다양한
시도

3

Training 중심

word2vec을 구현하면서 들었던 의문점들과 실험

왜 이렇게 구현하는 것인지 의문이
들었던 사항들에 대한 생각과 다양한
시도

4

우리가 word2vec을 통해 만들어낼 것, 그리고 사회적 의의

우리가 word2vec을 통해 창출해내고자하는 가치,
그리고 그것이 우리에게 가지는 사회적 의의

5

향후 계획

무엇을 보충하고, 고려할 것이며
앞으로 어떻게 진행해나갈 것인지

1st

우리가 이해한 word2vec

Distributional Semantics

By Learning

word2vec

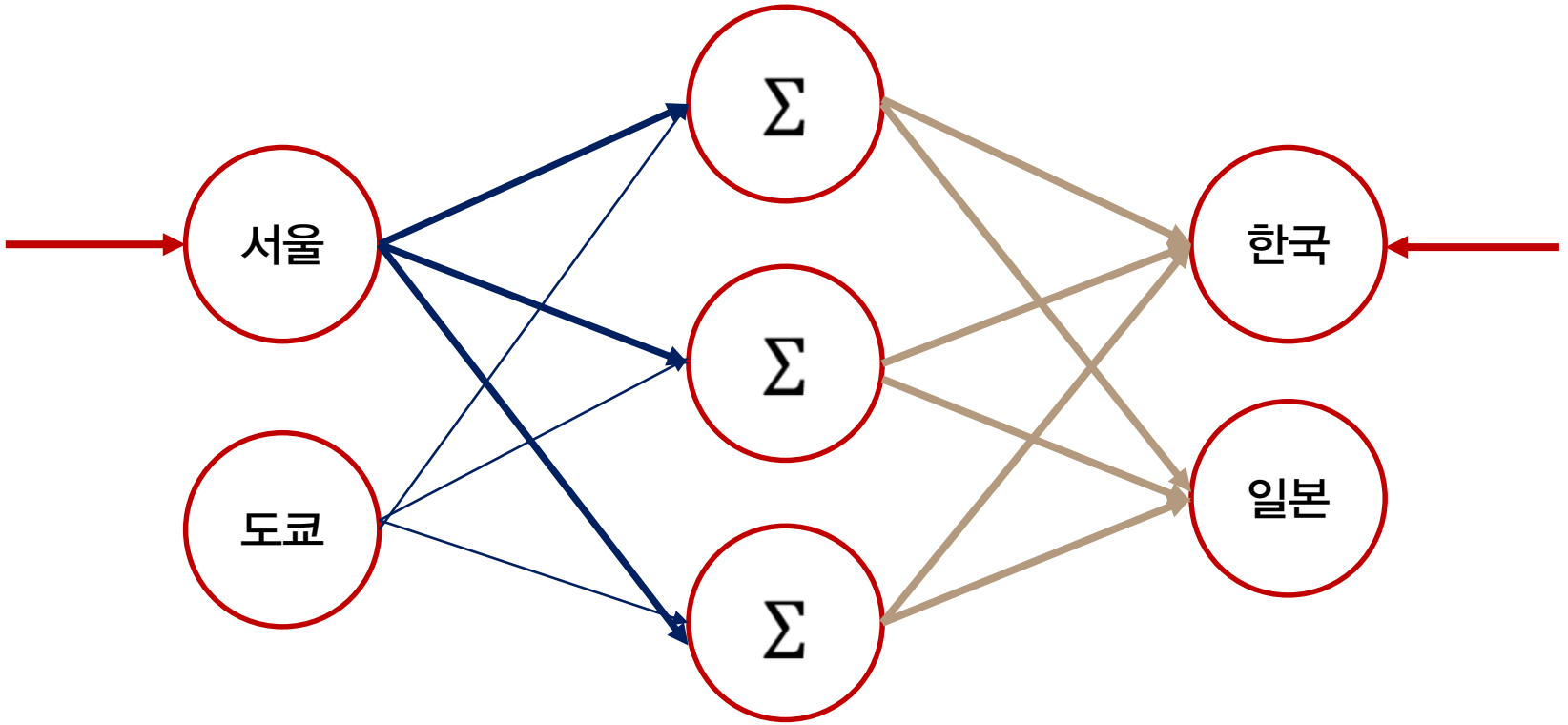
Word meaning

Implicit representation

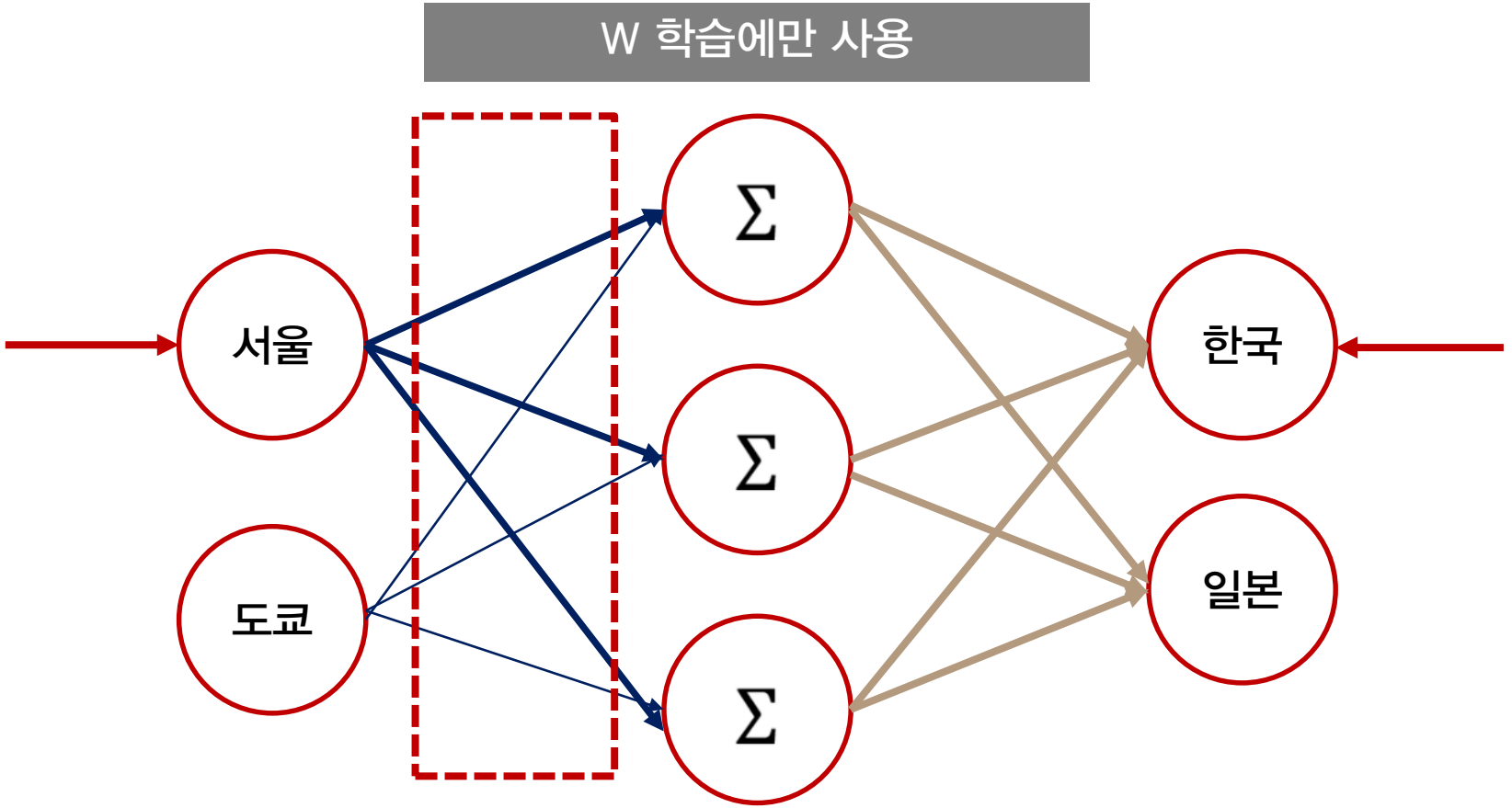
Vector Space
= Linear Regularity

Linear Regularity

히든레이어 = 시그마(시그모이드X)



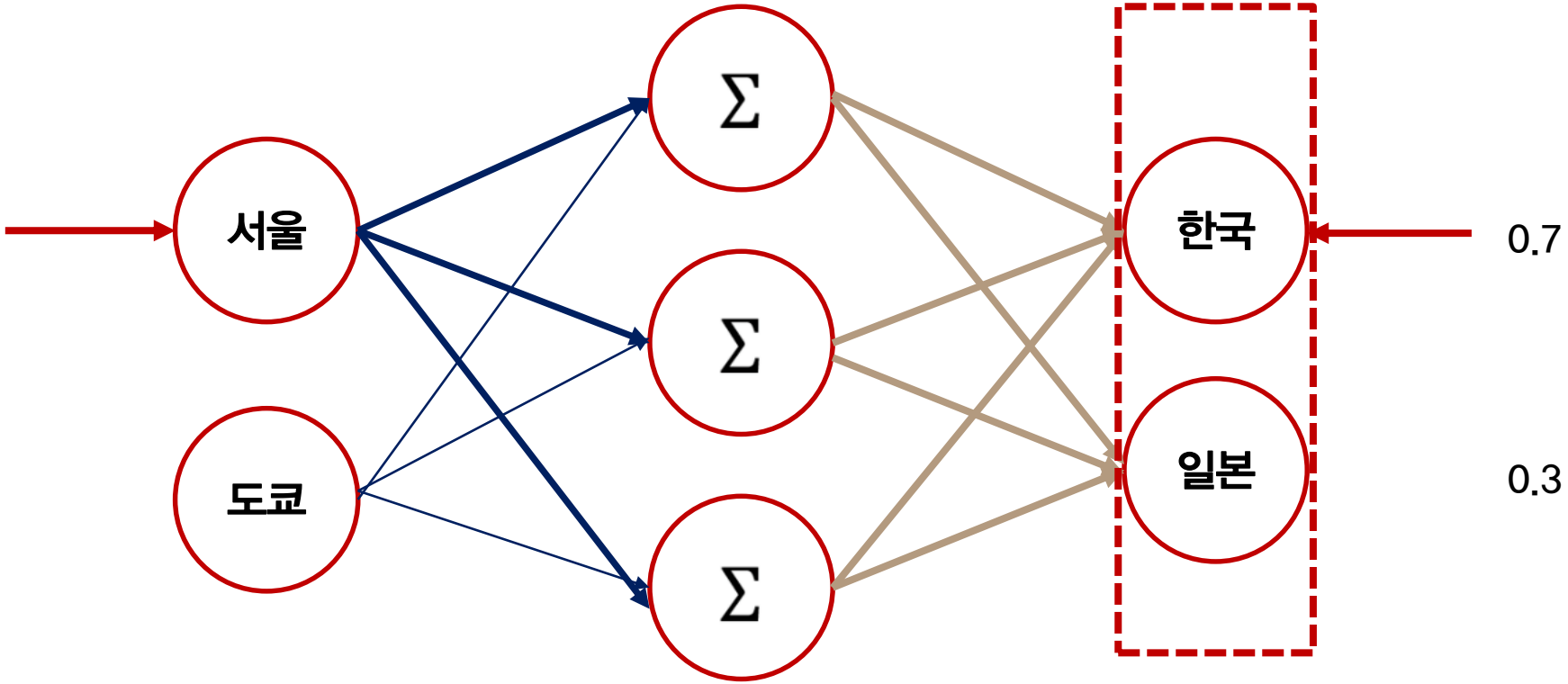
No use of Neural-Net



Softmax classifier

Multiclass classification

$$p(w_j|w_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})}$$



Softmax를 중심으로

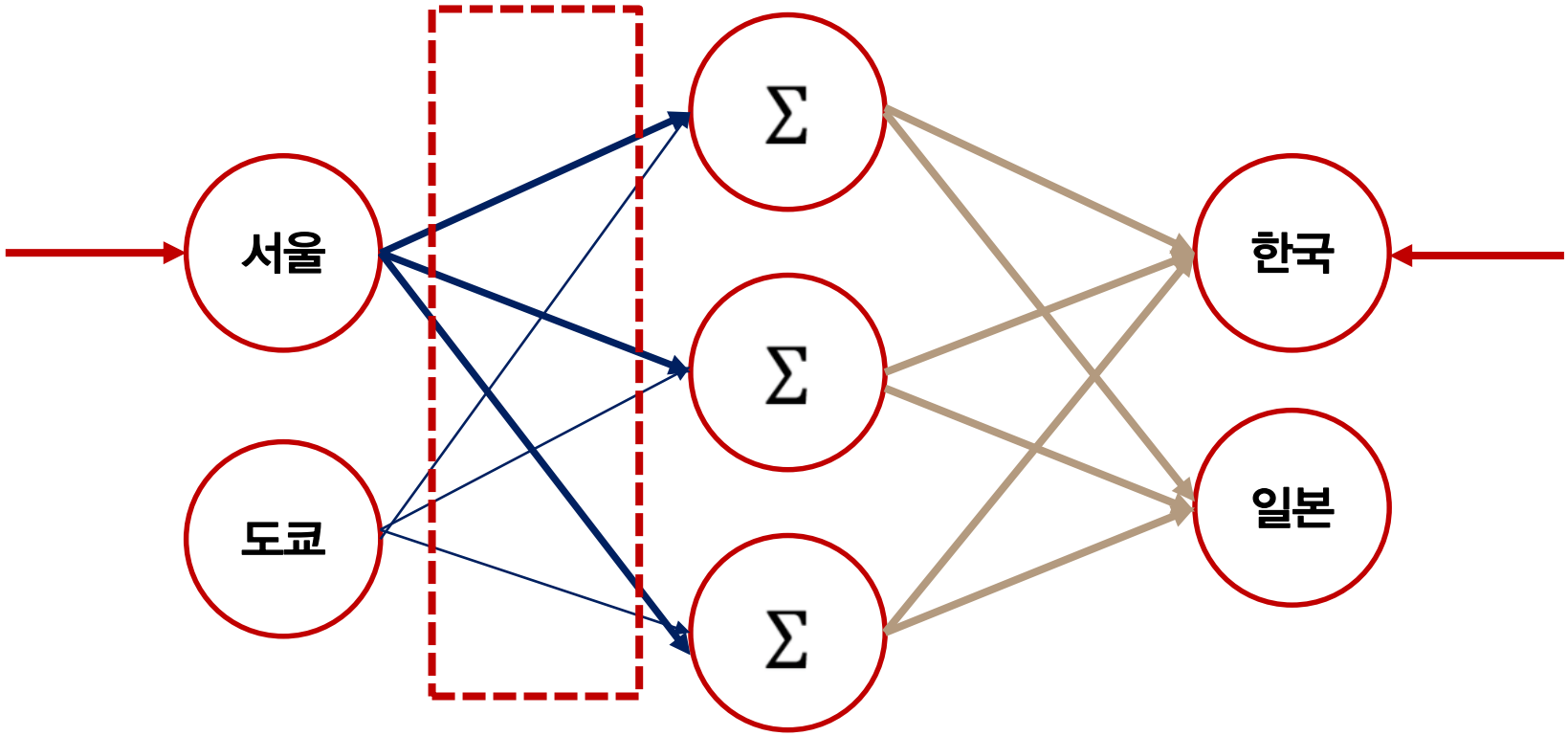
2nd

구현하면서 들었던 의문점들과 실험

Training Optimization

이곳을 update 하는 것은 쉽다.

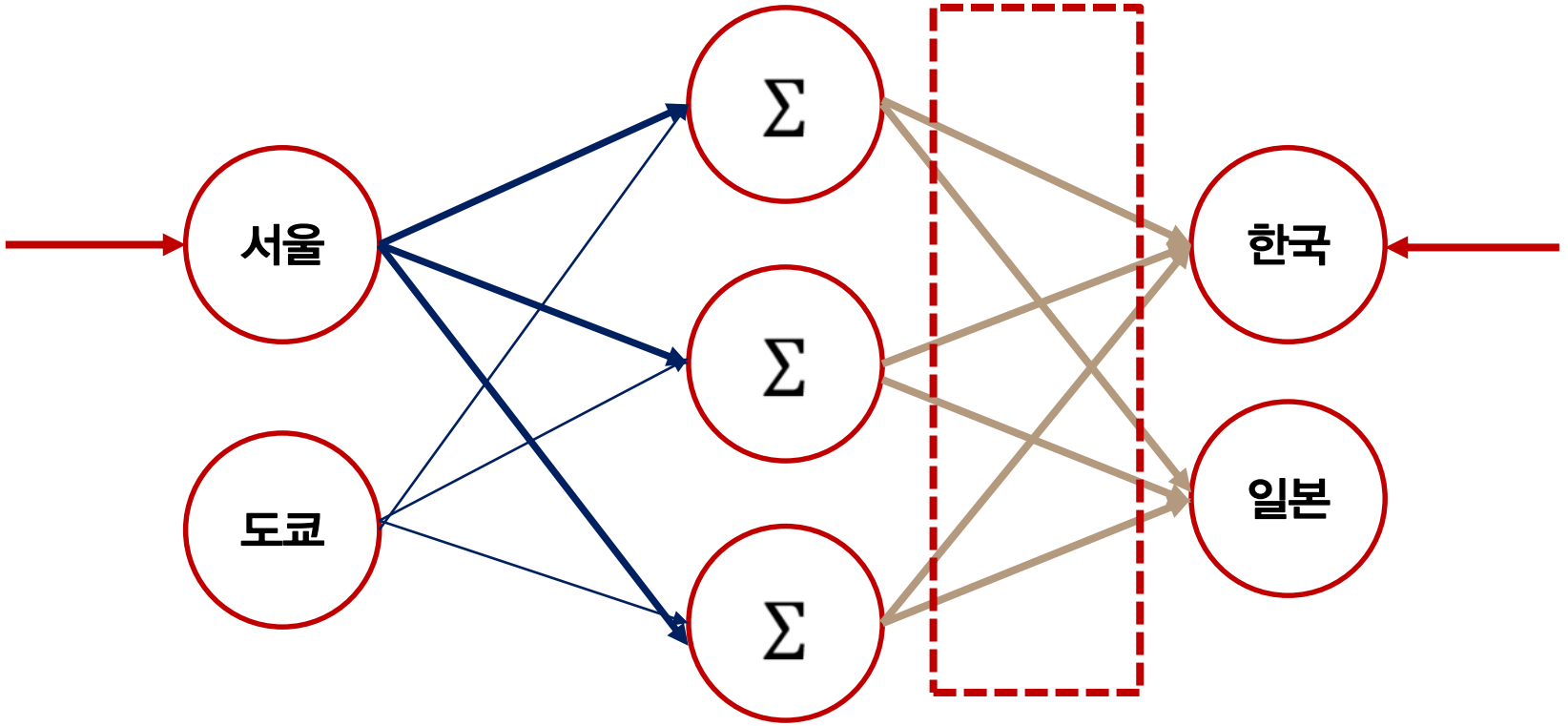
$$\mathbf{v}_{w_I}^{(\text{new})} = \mathbf{v}_{w_I}^{(\text{old})} - \eta \mathrm{EH}^T$$



Training Optimization

이곳을 업데이트 하려면..

$$\mathbf{v}'_{w_j}{}^{(\text{new})} = \mathbf{v}'_{w_j}{}^{(\text{old})} - \eta \cdot e_j \cdot \mathbf{h} \qquad \text{for } j = 1, 2, \cdots, V.$$



허프만 트리 노드

하나의 단어 노드에는 다음과 같은 정보가 들어있다.

0. 단어의 인덱스
1. 해당 단어의 허프만 코드
2. 해당 단어까지 거쳐간 부모 노드

Huffman tree for Hierarchical softmax

단어	빈도수
Apple	5
Orange	7
Rice	10
Juice	2
Milk	9

Hierarchical Softmax Pre-processing

STEP 1 Heap 생성

Huffman Tree 를 생성하기 위해 단어들의 빈도수를 key로 하는 min_heap 을 구성한다.

STEP 2 Huffman Tree 생성

min_heap을 이용하여 Huffman tree를 생성한다.

STEP 3 Huffman Tree 인코딩

생성된 Huffman tree를 탐색하며 코드를 부여한다. 이 때, 거쳐가는 부모 노드의 id값을 기록 해 둔다.

STEP 1 Heap 생성

```
std::vector<Word *> heap = words_;
std::make_heap(heap.begin(), heap.end(), comp);
```

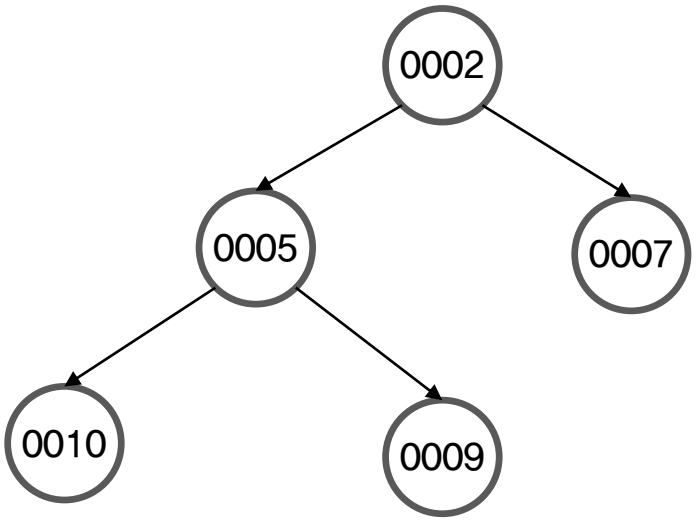
단어 벡터로 힙을 생성한다.

Comp 함수는 word1 과 word2의

빈도수를 비교하여 word1이 크면

True를 반환한다.

이렇게 하면 min_heap 이 생성된다.

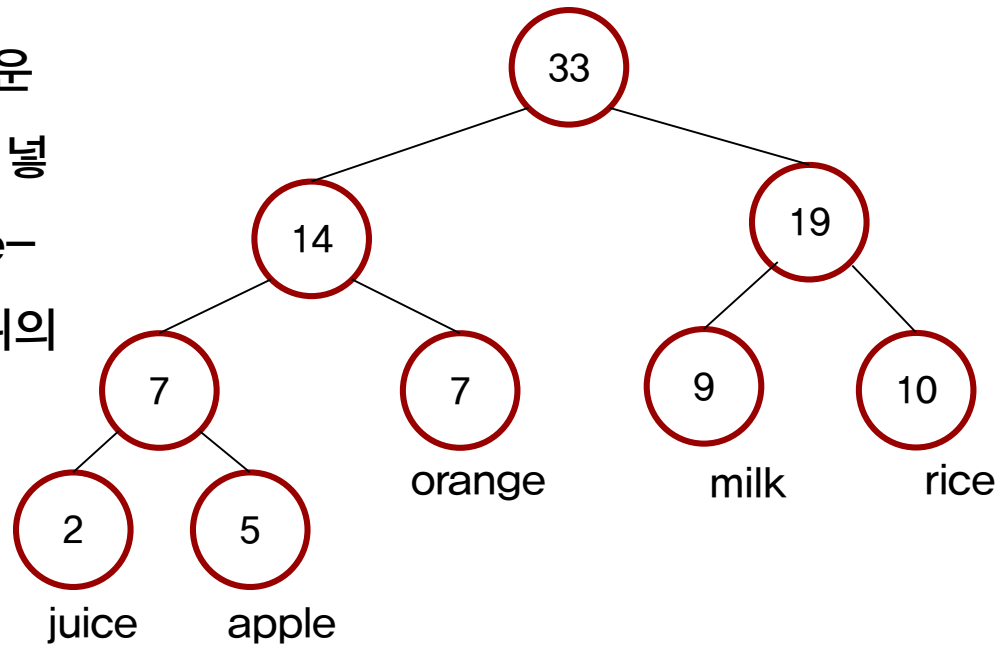


STEP 2 Huffman Tree 생성

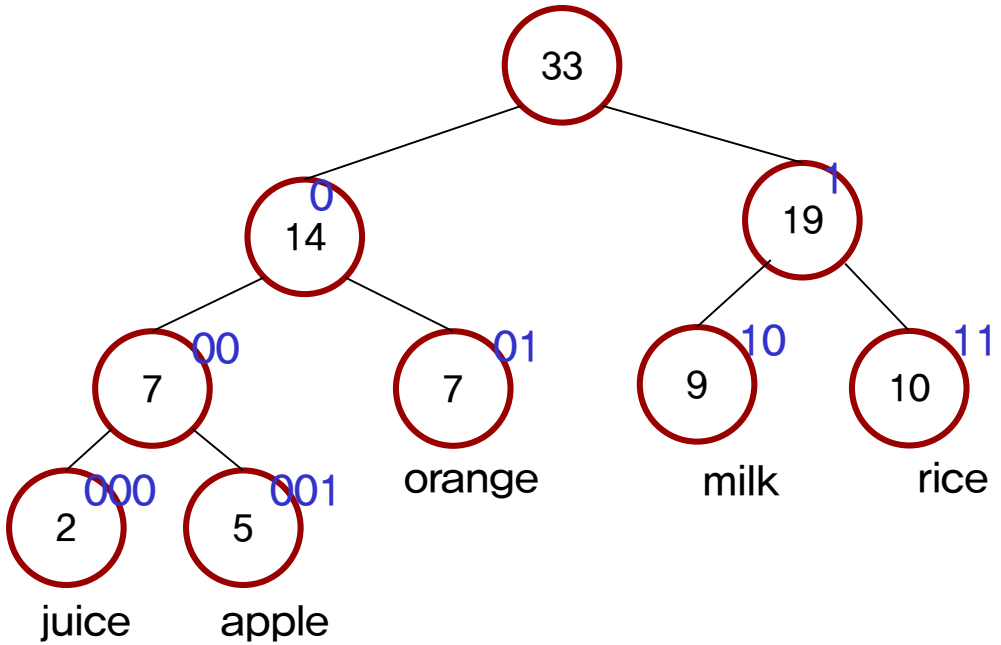
```
for (int i=0; i<n_words-1; ++i) {
    // 힙의 루트노드(여기에서는 빈도수가 가장 낮은 노드)를 힙의 맨 뒤로 보내고 heap을
    // pop_heap() 을 한 뒤 heap.back()을 하면 가장 작은 노드를 얻음과 동시에 he
    std::pop_heap(heap.begin(), heap.end(), comp);
    auto min1 = heap.back(); heap.pop_back();
    std::pop_heap(heap.begin(), heap.end(), comp);
    auto min2 = heap.back(); heap.pop_back();

    // tmp 는 작은 노드 2개를 합쳐서 새로운 부모 노드를 생성하기 위함.
    // tmp를 다시 min_heap 에 넣어준다.
    // 이 때 단어의 index는 i+n_words 로 초기화 해 준다.
    // 즉, 새로 생성된 부모 노드는 항상 n_words보다 i만큼 크게 된다. 이 때 i는 생성
    tmp.emplace_back(WordP(new Word{i + n_words, Cvt<String>::from_u

    // 힙의 마지막에 새로 생성한 노드 삽입.
    heap.push_back(tmp.back().get());

    // 힙을 새로 구성한다.
    std::push_heap(heap.begin(), heap.end(), comp);
}
```

STEP 2 Huffman Tree 생성 – 트리 구성

min_heap 에서 2개를 뽑아 새로운 부모 노드를 만들어 min_heap에 넣고 힙을 재구성한다. 이것을 node–1 번 반복하면 최종적으로는 맨 위의 부모 노드 하나만 남게 된다.



STEP 3 Huffman Tree 인코딩 – 단어에 코드 부여



STEP 3 Huffman Tree 인코딩 – 단어에 코드 부여

부여된 코드가 어떻게 사용되는가?

$$\mathbf{v}'^{(\text{new})}_j = \mathbf{v}'^{(\text{old})}_j - \eta \left(\sigma(\mathbf{v}'^{T}_j \mathbf{h}) - t_j\right) \cdot \mathbf{h}$$

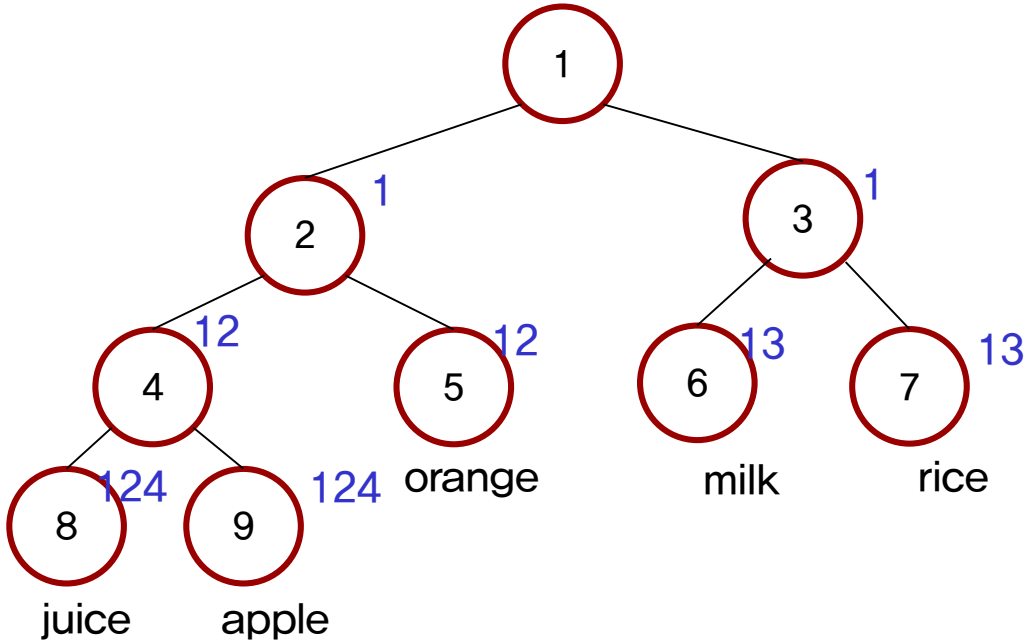
$$\frac{\partial E}{\partial \mathbf{h}} = \sum_{j=1}^{L(w)-1} \frac{\partial E}{\partial \mathbf{v}'_j \mathbf{h}} \cdot \frac{\partial \mathbf{v}'_j \mathbf{h}}{\partial \mathbf{h}}$$

$$= \sum_{j=1}^{L(w)-1} \left(\sigma(\mathbf{v}'^{T}_j \mathbf{h}) - t_j\right) \cdot \mathbf{v}'_j$$

$$:= \mathrm{EH}$$

t_j = 1이면 왼쪽 child,
0이면 오른쪽 child

STEP 3 Huffman Tree 인코딩 – 단어에 경로 기록



STEP 3 Huffman Tree 인코딩 – 단어에 경로 기록

부여된 경로가 어떻게 사용되는가?

$$\mathbf{v}'^{(\text{new})}_j = \mathbf{v}'^{(\text{old})}_j - \eta \left(\sigma(\mathbf{v}'^{T}_j \mathbf{h}) - t_j\right) \cdot \mathbf{h}$$

$\mathbf{v}'_j$ = W' Matrix의 한 행.

(j 는 1 ~ V−1)

$$\begin{aligned}
\frac{\partial E}{\partial \mathbf{h}} &= \sum_{j=1}^{L(w)-1} \frac{\partial E}{\partial \mathbf{v}'_j \mathbf{h}} \cdot \frac{\partial \mathbf{v}'_j \mathbf{h}}{\partial \mathbf{h}} \\
&= \sum_{j=1}^{L(w)-1} \left(\sigma(\mathbf{v}'^{T}_j \mathbf{h}) - t_j\right) \cdot \mathbf{v}'_j \\
&:= \mathrm{EH}
\end{aligned}$$

허프만트리 코드, 경로의 의미

코드 | 해당노드까지 오면서 왼쪽을 타고 왔는지,
오른쪽을 타고 왔는지.

경로 | 해당노드까지 오면서 거쳐간 부모 노드들

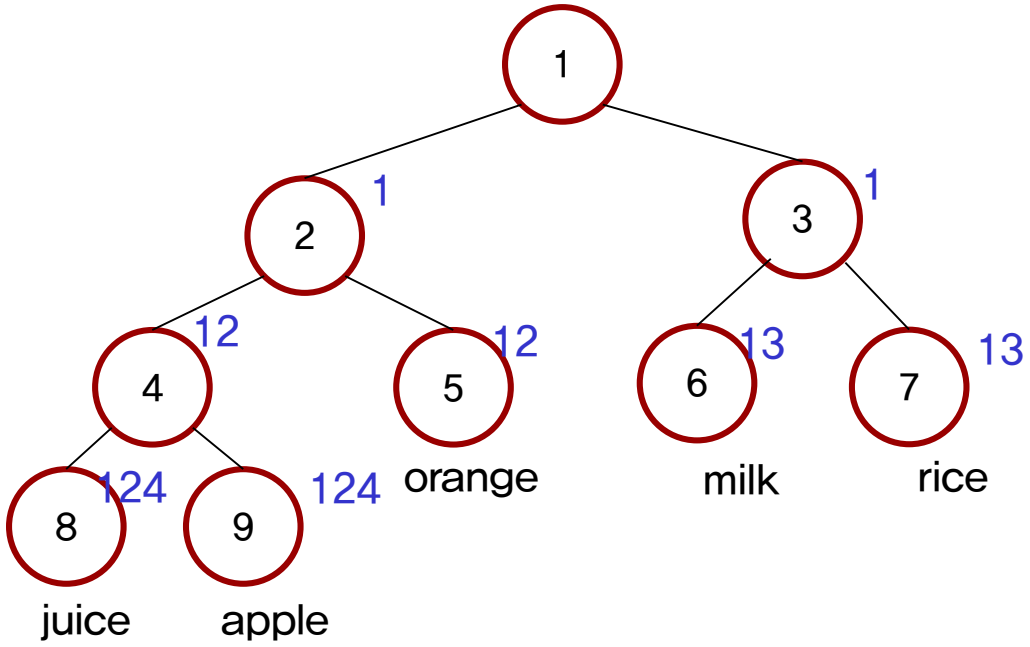
Hidden -〉 Output layer training 예시

for (int j = 0; j 〈 len(w.code); ++j) {

$$\mathbf{v}_j'^{(\text{new})} = \mathbf{v}_j'^{(\text{old})} - \eta \left(\sigma(\mathbf{v}_j'^{T} \mathbf{h}) - t_j \right) \cdot \mathbf{h}$$

}

Hidden –〉 Output layer training 예시



Index 1: 1 1 1 1

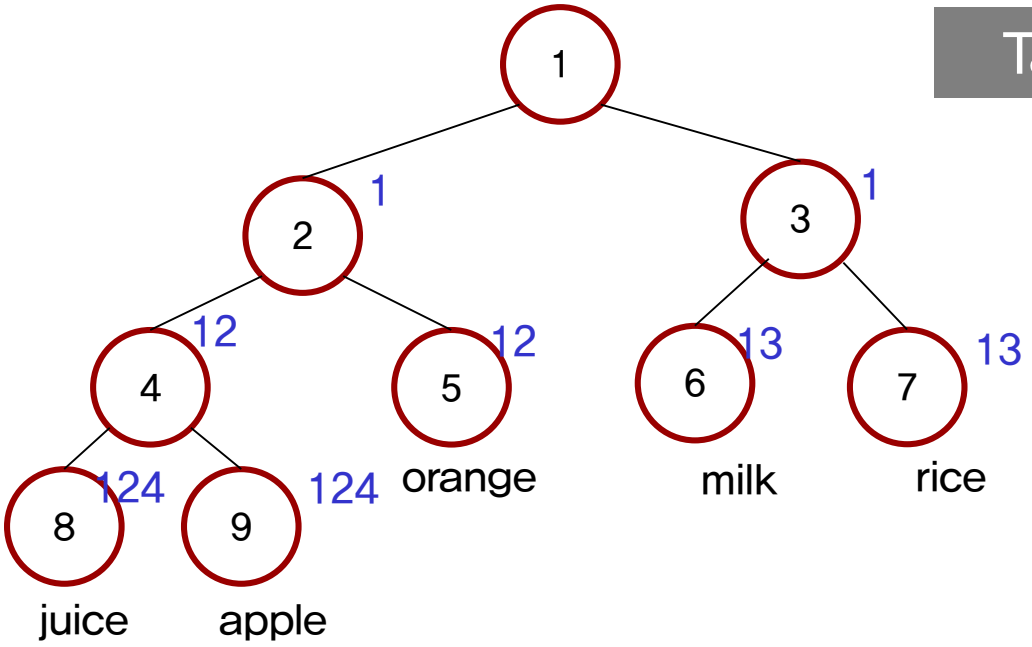
Index 2: 1 1 1 1

Index 3: 1 1 1 1

Index 4: 1 1 1 1

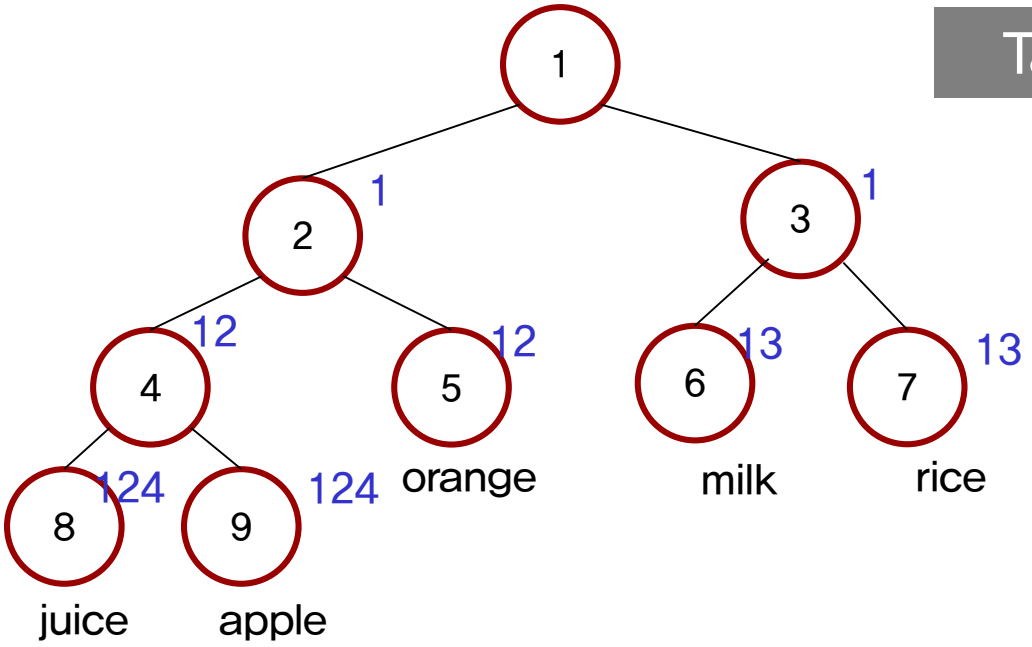
Index 5: 1 1 1 1

Hidden –〉 Output layer training 예시



Target word : orange

Hidden –〉 Output layer training 예시

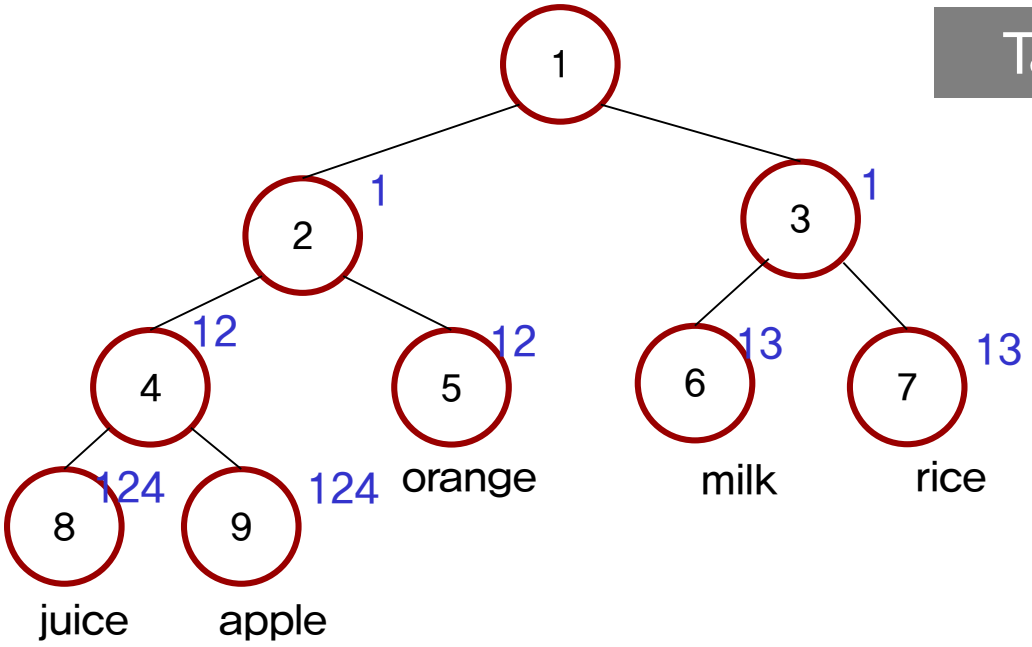


Target word : orange

Code : 0, 1

Path : 1, 2

Hidden –〉 Output layer training 예시



Target word : orange

Code 0, 1
Path 1, 2

Index 1: 1 1 1 1

Index 2: 1 1 1 1

Index 3: 1 1 1 1

Index 4: 1 1 1 1

Index 5: 1 1 1 1

$$\mathbf{v}'^{(\text{new})}_j = \mathbf{v}'^{(\text{old})}_j - \eta \left(\sigma(\mathbf{v}'^{T}_j \mathbf{h}) - t_j \right) \cdot \mathbf{h}$$

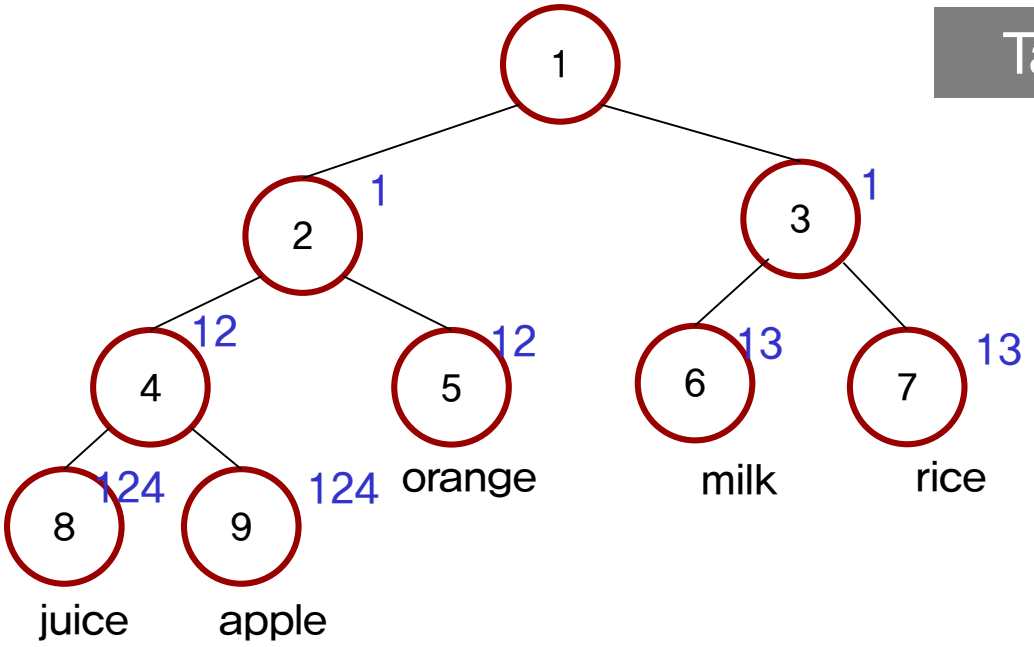
h ⇨ one word context의 경우

- Input –〉 hidden (hidden node 개수가 3이라고 가정) =〉 V X H 행렬
- One–hot vector인 input vector [0 1 0 0 0] 과 곱해지면?

2 2 2
2 2 2
2 2 2
2 2 2
2 2 2

- Transpose (h) = transpose(Weight [Input –〉 Hidden]) X (input vector(one–hot))
  = transpose(one row of Weight [Input –〉 Hidden])
- 양 변을 Transpose하면 h = one row of Weight [Input –〉 Hidden]

Hidden –〉 Output layer training 예시



Target word : orange

Code : 0, 1
Path : 1, 2

Index 1: 3 2 7 5

Index 2: 1 1 1 1

Index 3: 1 1 1 1

Index 4: 1 1 1 1

Index 5: 1 1 1 1

$$\mathbf{v}'^{(\text{new})}_j = \mathbf{v}'^{(\text{old})}_j - \eta \left(\sigma(\mathbf{v}'^{T}_j \mathbf{h}) - t_j \right) \cdot \mathbf{h}$$

Performance evaluation

- 허프만 트리 VS 일반 완전 이진 트리
- 허프만 트리는 빈번하게 등장하는 단어들에 대해 짧은 높이를 부여한다.
- 따라서 update 해야 하는 weight vector 개수가 빈도가 높은 target word일수록 줄어든다.
- 만약 허프만 인코딩을 사용하지 않는다면?

No Huffman encoding vs Huffman encoding

```
training alpha: 0.0006 progress: 97.73% words per sec: 435.777K
training alpha: 0.0004 progress: 98.34% words per sec: 424.424K
training alpha: 0.0003 progress: 98.96% words per sec: 430.300K
training alpha: 0.0001 progress: 99.54% words per sec: 412.915K
train: 39.1542 seconds
save model: 4.7224 seconds
```

```
training alpha: 0.0003 progress: 98.95% words per sec: 436.738K
training alpha: 0.0001 progress: 99.56% words per sec: 437.580K
training alpha: 0.0001 progress: 99.57% words per sec: 438.297K
training alpha: 0.0001 progress: 99.87% words per sec: 402.003K
train: 30.6084 seconds
save model: 4.7383 seconds
```

Training을 중심으로

3rd

구현하면서 들었던 의문점들과 실험

Training

- Sigmoid값을 계산해 놓았습니다.

$$\sigma(u) = \frac{1}{1 + e^{-u}}$$

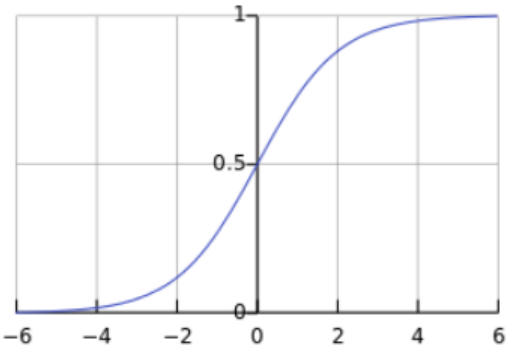
```
const int max_size = 1000;
const float max_exp = 6.0;
const static std::vector<float> table = [&]() {
std::vector<float> x(max_size);
for (size_t i = 0; i<max_size; ++i) { float f = exp((i / float(max_size) *
2 − 1) * max_exp); x[i] = f / (f + 1); }
return x;
}();
```

Training

- float f = v::dot(l1, l2); // OUTPUT 값

$$\sigma(u) = \frac{1}{1 + e^{-u}}$$

- If (f <= −max_exp || f >= max_exp)
- Continue;



- int fi = int((f + max_exp) * (max_size / max_exp / 2.
- // OUTPUT에 해당하는 sigmoid 함수의 input 값 계산
- f = table[fi];

Training

```
int len = sentence.words_.size();
int reduced_window = rand() % window_;
for (int i = 0; i<len; ++i) {
    const Word& current = *sentence.words_[i];
    size_t codelen = current.codes_.size();


    int j = std::max(0, i - window_ + reduced_window);
    int k = std::min(len, i + window_ + 1 - reduced_window);
```

```
int len = sentence.words_.size();
int reduced_window = rand() % window_;
for (int i = 0; i<len; ++i) {
    const Word& current = *sentence.words_[i];
    size_t codelen = current.codes_.size();

    int j = std::max(0, i - window_ + reduced_window);
    int k = std::min(len, i + window_ + 1 - reduced_window);
    for (; j < k; ++j) {
        const Word *word = sentence.words_[j];
        if (j == i || word->codes_.empty())
            continue;
        int word_index = word->index_;
        auto& l1 = syn0_[word_index];
        std::fill(work.begin(), work.end(), 0);
        int m = work.size();
        for (size_t b = 0; b<codelen; ++b) {
            int idx = current.points_[b];
            auto& l2 = syn1_[idx];

            float f = v::dot(l1, l2);
            if (f <= -max_exp || f >= max_exp)
                continue;
            int fi = int((f + max_exp) * (max_size / max_exp / 2.0));
            f = table[fi];

            float g = (1 - current.codes_[b] - f) * alpha;
            for (int n = 0; n < m; n++) {
                work[n] += g * l2[n];
            }
            for (int n = 0; n < m; n++) {
                l2[n] += g * l1[n];
            }
        }
        for (int n = 0; n < m; n++) {
            l1[n] += work[n];        }
}
```

Training

Single

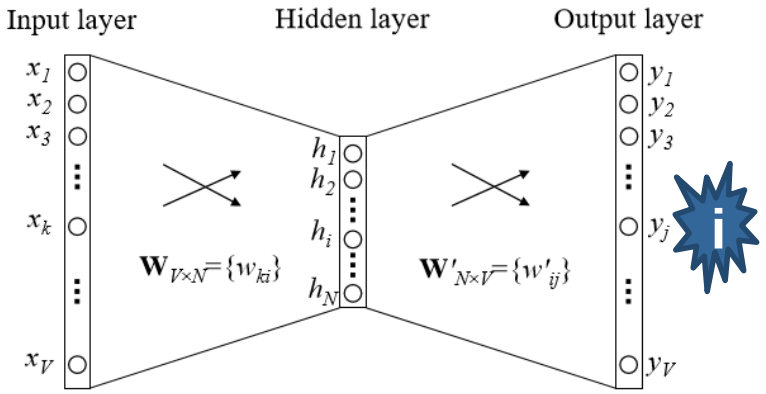
```
training alpha: 0.0006 progress: 97.70% words per sec: 147.916K
training alpha: 0.0004 progress: 98.31% words per sec: 147.247K
training alpha: 0.0003 progress: 98.93% words per sec: 148.591K
training alpha: 0.0001 progress: 99.54% words per sec: 147.955K
training alpha: 0.0001 progress: 100.00% words per sec: 140.787K
train: 111.9073 seconds
save model: 4.9080 seconds
```

Multi-4

```
training alpha: 0.0006 progress: 97.74% words per sec: 513.359K
training alpha: 0.0004 progress: 98.35% words per sec: 500.283K
training alpha: 0.0003 progress: 98.97% words per sec: 492.001K
training alpha: 0.0001 progress: 99.58% words per sec: 441.080K
training alpha: 0.0001 progress: 99.71% words per sec: 380.738K
train: 32.0756 seconds
save model: 5.4163 seconds
```

Training

```
int len = sentence.words_.size();
int reduced_window = rand() % window_;
for (int i = 0; i<len; ++i) {
    const Word& current = *sentence.words_[i];
    size_t codelen = current.codes_.size();


    int j = std::max(0, i − window_ +
reduced_window);
    int k = std::min(len, i + window_ + 1 −
reduced_window);
```



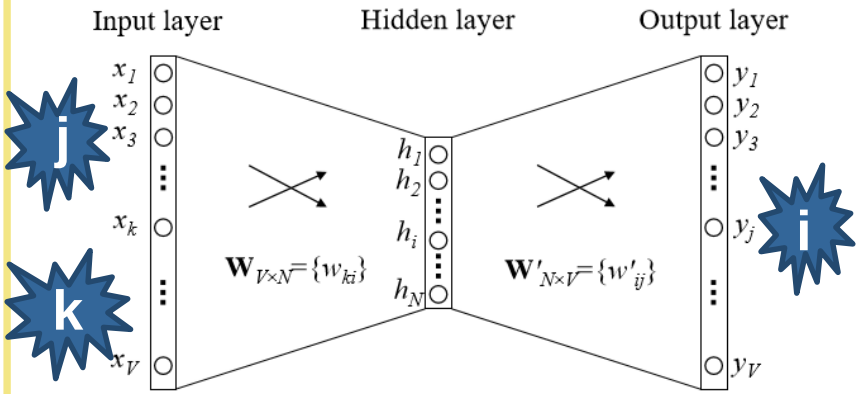
Training

```
for (; j < k; ++j) {
    const Word *word = sentence.words_[j];
    if (j == i || word->codes_.empty())
        continue;

    int word_index = word->index_;
    auto& l1 = syn0_[word_index];

    std::fill(work.begin(), work.end(), 0);
    int m = work.size();
```

```
int len = sentence.words_.size();
int reduced_window = rand() % window_;
for (int i = 0; i<len; ++i) {
    const Word& current = *sentence.words_[i];
    size_t codelen = current.codes_.size();

    int j = std::max(0, i - window_ + reduced_window);
    int k = std::min(len, i + window_ + 1 - reduced_window);
    for (; j < k; ++j) {
        const Word *word = sentence.words_[j];
        if (j == i || word->codes_.empty())
            continue;
        int word_index = word->index_;
        auto& l1 = syn0_[word_index];
        std::fill(work.begin(), work.end(), 0);
        int m = work.size();
        for (size_t b = 0; b<codelen; ++b) {
            int idx = current.points_[b];
            auto& l2 = syn1_[idx];

            float f = v::dot(l1, l2);
            if (f <= -max_exp || f >= max_exp)
                continue;
            int fi = int((f + max_exp) * (max_size / max_exp / 2.0));
            f = table[fi];

            float g = (1 - current.codes_[b] - f) * alpha;
            for (int n = 0; n < m; n++) {
                work[n] += g * l2[n];
            }
            for (int n = 0; n < m; n++) {
                l2[n] += g * l1[n];
            }
        }
        for (int n = 0; n < m; n++) {
            l1[n] += work[n];        }
}
```

Training

```
for (; j < k; ++j) {
    const Word *word = sentence.words_[j];
    if (j == i || word->codes_.empty())
        continue;

    int word_index = word->index_;
    auto& l1 = syn0_[word_index];

    std::fill(work.begin(), work.end(), 0);
    int m = work.size();
```



Training

```
for (size_t b = 0; b<codelen; ++b) {
    int idx = current.points_[b];
    auto& l2 = syn1_[idx];
    float f = v::dot(l1, l2);
    if (f <= -max_exp || f >= max_exp)
        continue;

    int fi = int((f + max_exp) * (max_size /
max_exp / 2.0));
    f = table[fi];

    float g = (1 - current.codes_[b] - f) * alpha;
```

```
int len = sentence.words_.size();
int reduced_window = rand() % window_;
for (int i = 0; i<len; ++i) {
    const Word& current = *sentence.words_[i];
    size_t codelen = current.codes_.size();

    int j = std::max(0, i - window_ + reduced_window);
    int k = std::min(len, i + window_ + 1 - reduced_window);
    for (; j < k; ++j) {
        const Word *word = sentence.words_[j];
        if (j == i || word->codes_.empty())
            continue;
        int word_index = word->index_;
        auto& l1 = syn0_[word_index];
        std::fill(work.begin(), work.end(), 0);
        int m = work.size();
        for (size_t b = 0; b<codelen; ++b) {
            int idx = current.points_[b];
            auto& l2 = syn1_[idx];

            float f = v::dot(l1, l2);
            if (f <= -max_exp || f >= max_exp)
                continue;
            int fi = int((f + max_exp) * (max_size / max_exp / 2.0));
            f = table[fi];

            float g = (1 - current.codes_[b] - f) * alpha;
            for (int n = 0; n < m; n++) {
                work[n] += g * l2[n];
            }
            for (int n = 0; n < m; n++) {
                l2[n] += g * l1[n];
            }
        }
        for (int n = 0; n < m; n++) {
            l1[n] += work[n];        }
}
```

3

Training

```
float g = (1 − current.codes_[b] − f) * alpha;
```

$$\frac{\partial E}{\partial \mathbf{v}'_j \mathbf{h}} = \left(\sigma([\![\cdot]\!]\mathbf{v}'^T_j \mathbf{h}) - 1\right)[\![\cdot]\!]$$

$$= \begin{cases} \sigma(\mathbf{v}'^T_j \mathbf{h}) - 1 & ([\![\cdot]\!] = 1) \\ \sigma(\mathbf{v}'^T_j \mathbf{h}) & ([\![\cdot]\!] = -1) \end{cases}$$

$$= \sigma(\mathbf{v}'^T_j \mathbf{h}) - t_j$$

$$\mathbf{v}'^{(\text{new})}_j = \mathbf{v}'^{(\text{old})}_j - \eta\left(\sigma(\mathbf{v}'^T_j \mathbf{h}) - t_j\right) \cdot \mathbf{h}$$

Training

```
for (int n = 0; n < m; n++) {
        work[n] += g * l2[n];
}
for (int n = 0; n < m; n++) {
        l2[n] += g * l1[n];
}
}
for (int n = 0; n < m; n++) {
        l1[n] += work[n];
}
```

```
int len = sentence.words_.size();
int reduced_window = rand() % window_;
for (int i = 0; i<len; ++i) {
    const Word& current = *sentence.words_[i];
    size_t codelen = current.codes_.size();

    int j = std::max(0, i - window_ + reduced_window);
    int k = std::min(len, i + window_ + 1 - reduced_window);
    for (; j < k; ++j) {
        const Word *word = sentence.words_[j];
        if (j == i || word->codes_.empty())
            continue;
        int word_index = word->index_;
        auto& l1 = syn0_[word_index];
        std::fill(work.begin(), work.end(), 0);
        int m = work.size();
        for (size_t b = 0; b<codelen; ++b) {
            int idx = current.points_[b];
            auto& l2 = syn1_[idx];

            float f = v::dot(l1, l2);
            if (f <= -max_exp || f >= max_exp)
                continue;
            int fi = int((f + max_exp) * (max_size / max_exp / 2.0));
            f = table[fi];

            float g = (1 - current.codes_[b] - f) * alpha;
            for (int n = 0; n < m; n++) {
                work[n] += g * l2[n];
            }
            for (int n = 0; n < m; n++) {
                l2[n] += g * l1[n];
            }
        }
        for (int n = 0; n < m; n++) {
            l1[n] += work[n];        }
    }
```

4

Training

```
for (int n = 0; n < m; n++) {
        work[n] += g * l2[n];
}
for (int n = 0; n < m; n++) {
        l2[n] += g * l1[n];
}
}
for (int n = 0; n < m; n++) {
        l1[n] += work[n];
}
```

$$\mathbf{v}_j'^{(\text{new})} = \mathbf{v}_j'^{(\text{old})} - \eta \left(\sigma(\mathbf{v}_j'^{T}\mathbf{h}) - t_j \right) \cdot \mathbf{h}$$

$$\begin{aligned} \frac{\partial E}{\partial \mathbf{h}} &= \sum_{j=1}^{L(w)-1} \frac{\partial E}{\partial \mathbf{v}_j'\mathbf{h}} \cdot \frac{\partial \mathbf{v}_j'\mathbf{h}}{\partial \mathbf{h}} \\ &= \sum_{j=1}^{L(w)-1} \left(\sigma(\mathbf{v}_j'^{T}\mathbf{h}) - t_j \right) \cdot \mathbf{v}_j' \\ &:= \mathrm{EH} \end{aligned}$$

4th

우리가 word2vec을 통해 만들어낼 것, 그리고 사회적 의의

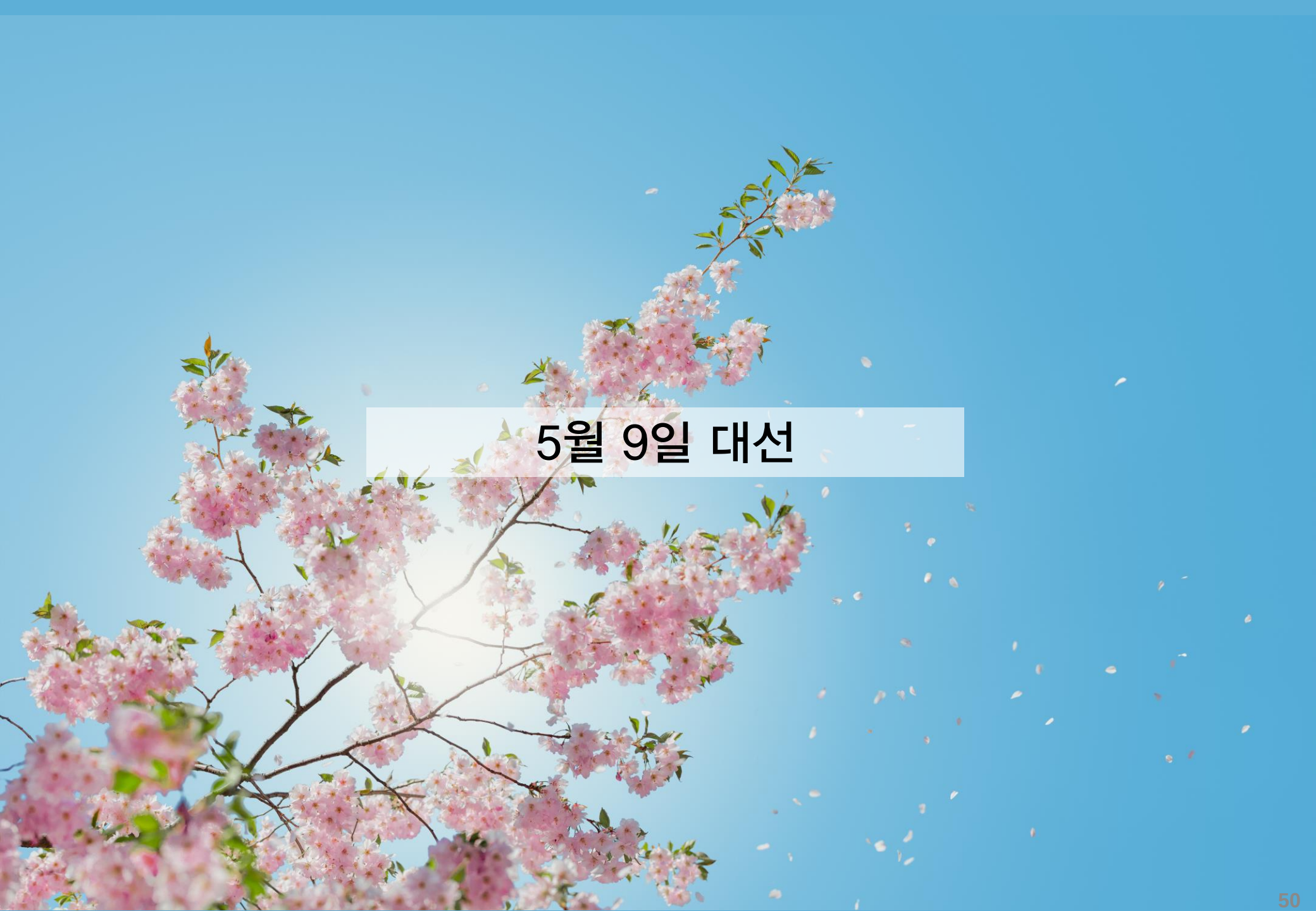
정치인2VEC

정치인2VEC?

내가 선호하는 정치인을 입력하면,
정치인의 정당과, 안보/교육/경제/민생 등 여러부문에
대한 스탠스를 고려하여 나의 성향과 비슷한 대선후보
(및 정치인)를 추천해주는 서비스

왜 정치인2vec일까?


국민들의 높아진 정치적 관심

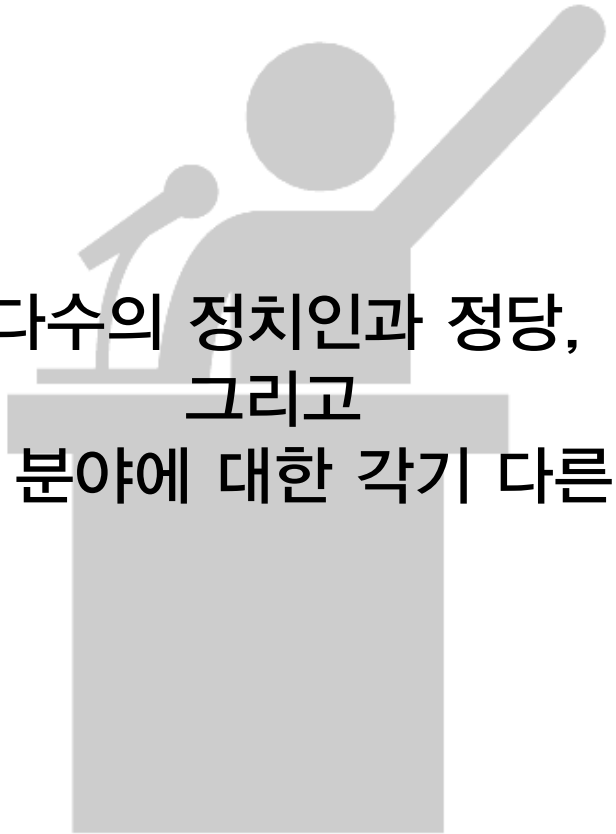

5월 9일 대선

정치는 패키지다

정치는 패키지다

진보정당의 정치인이라고 해서 모든 분야의 스탠스가 진보적인 것도 아니고, 보수정당의 정치인이라고 해서 모든 분야의 스탠스가 보수적인 것도 아니다.

정치인2vec



다수의 정치인과 정당,
그리고
다양한 분야에 대한 각기 다른 입장

구현 계획

STEP 1 Training

데이터를 통해 유사한 스탠스 취하는 정치인끼리 같은 context내에 있도록 Training

STEP 2 Cosine Similiarity

사용자에게 입력받은 정치인을 (입력받은 정치인의 벡터는 이미 training 되어있음) 다른 정치인들의 벡터와 내적하여 cosine similiarity를 구함

STEP 3 Sort(descending order) & print output

cosine similiarity가 높은 정치인의 순서로 정렬하고 상위 몇 명의 정치인을 출력한다.

구현 계획

STEP 1 Training

데이터를 통해 유사한 스탠스 취하는 정치인끼리 같은 context내에 있도록 Training

구현 계획

STEP 1 Training

데이터를 통해 유사한 스탠스 취하는 정치인끼리 같은 context내에 있도록 Training

HOW ?

HOW ?

데이터를 어디서 얻어서 training_set을 어떻게 만들 것 인가?

1. **기사 크롤링**
   기사를 통해 인물의 발화를 분석하고, 같은 스탠스를 취하는 의원끼리 같은 context로 분류

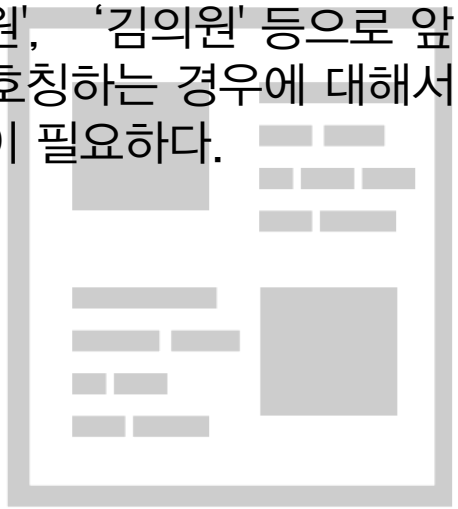
2. **의안정보시스템**
   법안의 발의를 같이 한 의원들 끼리 같은 context로 분류

정치인2vec

기사 크롤링의 어려운 점

Semantic analysis를 하지 않는다면 하나의 기사를 Context로 볼 것이기 때문에 기사 내에서 등장하는 정치인들 끼리는 뜻을 같이한다고 간주해야 한다. 때문에 그런 기사들만을 걸러내는 것이 필요하다.

또한 의원의 이름을 직접 지칭하지 않고 '박의원', '김의원' 등으로 앞에서 언급한 의원에 대해 성씨 + '의원'으로 호칭하는 경우에 대해서 지칭하는 대상을 알아내는 담화분석이 필요하다.



정치인2vec

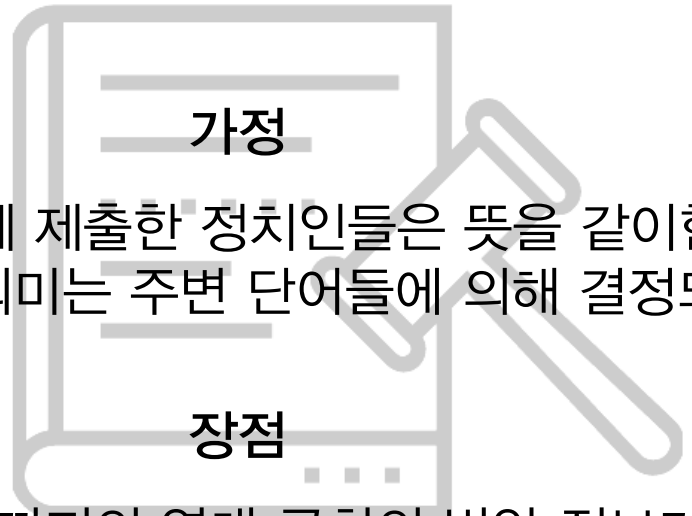
대안?

대안?!

의안정보시스템

의안정보시스템

의안정보시스템에서 의안 제출에 동의한 의원들의 이름을 볼 수 있는 것을 이용. 이것을 크롤링 하여 하나의 Context로 본다.



가정

의안을 제출할 때 함께 제출한 정치인들은 뜻을 같이한다고 가정.
word2vec에서 단어의 의미는 주변 단어들에 의해 결정되는 것과 같음.

장점

1대 국회의원부터 현재까지의 역대 국회의 법안 정보가 모두가 있는 방대한 데이터,약 7만건

데이터 전처리 시 유의할 점

False Negative

정치인의 소속정당은 바뀔 수 있으므로 (의원이름, 정당)으로 학습하면 false negative(실제론 동일 인물인데 아니라고 판단하는 오류)가 발생할 수 있다. 그리고 소속정당의 이름이 바뀌는 경우도 있다.

False Positive

동명이인인데 동일인으로 판단하는 오류를 막기 위해서 한자를 고려하여 구분. (그리고 한자까지 같은 경우는 거의 없다고 가정한다.)

정치인2vec 의의

자신의 성향과 비슷한 정치인을 손쉽게 알아볼 수 있게 한다.
따라서 본인과 뜻을 같이하는 정치인에 관심을 갖고, 선거에서 합리적인 선택을 할 수 있도록 도울 수 있다.
또한, 잘 알지 못하는 정치인이 미디어에서 언급될 때 정치인 2vec에 랭크되었던 정치인이라면 더욱 관심을 가지고 볼 수 있다.

5th

향후 진행 계획

향후 진행 계획

1. Word2vec

작성한 word2vec을 정치인2vec에 맞게 변형

2. 정치인2vec

데이터 acquisition을 위한 크롤링 스크립트 짜기

3. 정치인2vec

cbow와 skip-gram간의 성능 비교 분석 및 취사선택, efficiency향상

4. 정치인2vec

인터페이스 제작, UI/UX강화

향후 진행 계획

1 2 3 4 5 6

word2vec 변형

정치인2Vec Training

정치인2Vec 구현, 성능 개선

고려대학교 컴퓨터학과
데이터과학 2팀 발표자료
word2vec reproduce & application

Thank you

Q & A