Text Processing

(good survey at https://github.com/kk7nc/Text_Classification)

2021. 8

Yongjin Jeong, KwangWoon University

Text Processing

- **Applications**
  - Language translation
  - Sentiment analysis (positive, negative, neutral) (기쁨, 우울, 슬픔, 등)
  - Spam filtering (emails, messages)
  - Information extraction (from SNS, Web, Documents)
  - AI Speaker, Chatbot, Question answering
- **Language**
  - Token: basic text processing unit (character, **word**, n-gram, POS(part-of-speech, 품사), syllable(음절:자음+모음))
  - Corpus (말뭉치): collection of text (or document) dataset
  - Stop words (불용어): most common words in a language (“the”, “is”, etc)
- **Tools**
  - NLTK (Natural Language ToolKit): all sorts of tasks from tokenization, stemming, tagging, parsing, and beyond
  - KoNLPy (NLP library for Korean Language)
  - Gensim.Word2Vec(), tf.Keras.Embedding(), CountVectorizer(), BeautifulSoup()

Text Processing

- **Tokenization**
  - way of separating a piece of text into smaller units called tokens.
  - Tokens can be either words, characters, partial words (n-gram characters)
- **NLTK (Natural Language Toolkit)**

```
1 from nltk.tokenize import word_tokenize
2
3 text = "After sleeping for four hours, he decided to sleep for another four"
4 tokens = word_tokenize(text)
5 print(tokens)
```

```
['After', 'sleeping', 'for', 'four', 'hours', ',', 'he', 'decided', 'to', 'sleep', 'for', 'another', 'four']
```

```
1 example_sent = "This is a sample sentence, showing off the stop words filtration."
2 stop_words = set(stopwords.words('english'))
3 word_tokens = word_tokenize(example_sent)
4 filtered_sentence = [w for w in word_tokens if not w in stop_words]
5 print(word_tokens)
6 print(filtered_sentence)
```

```
['This', 'is', 'a', 'sample', 'sentence', ',', 'showing', 'off', 'the', 'stop', 'words', 'filtration', '.']
['This', 'sample', 'sentence', ',', 'showing', 'stop', 'words', 'filtration', '.']
```

Text Processing

- **KoNLPy (Korean Natural Language Processing for Python: 코엔엘파이)**
  - Morpheme tokenizer (형태소 분석기):
    - Okt(Open Korea Text, 옛이름: Twitter)
    - Mecab(메캅), Komoran(코모란), Hannanum(한나눔)
    - Kkma (꼬꼬마)

```
1 from konlpy.tag import Okt
2 okt=Okt()
3
4 print(okt.morphs("우리가 이 과제를 잘 할 수 있을까?"))
5 print(okt.pos("우리가 이 과제를 잘 할 수 있을까?", norm=True, stem=True))
6 print(okt.nouns("우리가 이 과제를 잘 할 수 있을까?"))
```

```
['우리', '가', '이', '과제', '를', '잘', '할', '수', '있을까', '?']
[('우리', 'Noun'), ('가', 'Josa'), ('이', 'Noun'), ('과제', 'Noun'), ('를', 'Josa'), ('자다', 'Verb'), ('하다', 'Verb'), (
['우리', '이', '과제', '수']
```

```
1 from konlpy.tag import Okt
2 okt=Okt()
3 word_tags = okt.pos("우리가 이 과제를 잘 할 수 있을까?", norm=True, stem=True)
4 print(word_tags)
5 stop_words = [word[0] for word in word_tags if word[1]=="Josa"]
6 print (stop_words)
```

```
[('우리', 'Noun'), ('가', 'Josa'), ('이', 'Noun'), ('과제', 'Noun'), ('를', 'Josa'), ('자다', 'Verb'), ('하다', 'Verb'), (
['가', '를']
```

Text Processing - steps

- **Data preprocessing:**
  - Tokenization: convert sentences to words
  - Removing punctuations and tags
  - Removing stop words
  - Stemming: words are reduced to a root (ex: formalize -> formal, allowance -> allow)
  - Lemmatization (ex: is, are, am -> be, having -> have)
- **Feature Extraction**
  - One-hot encoding
  - BOW (Bag of Words): each sentence is represented with counts (or presence) of words (discards word order, hence ignoring the context and meaning in the document), sparse
    - CountVectorizer(), TfIdfVectorizer()
  - Word Embedding: represent each word with a meaningful n-dimensional vector, dense
    - Word2vec(): put words of similar meaning (or context) to closer places
    - tf.keras.Embedding(): vector representation is trained (only used as the first layer in deep learning)
- **Choose ML algorithms**
  - Classical ML (simple text classification) or Deep Learning (sentiment analysis or language translation)

Text Processing

- **BoW (Bag of Words)**

  - From http://www.datasciencecourse.org/notes/free_text/
    - Doc1 = "The goal of this lecture is to explain the basics of free text processing"
    - Doc2 = "The bag of words model is one such approach"
    - Doc3 = "Text processing via bag of words"

$$X = \begin{bmatrix} 2 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \cdots & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

the is of goal lecture bag words via text approach

Document 1
Document 2
Document 3

$$X = \begin{bmatrix} 0.8 & 0.4 & 0 & 1.1 & \\ 0.4 & 0.4 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & \end{bmatrix}$$

the is of goal

$$\text{idf}_j = \log\left(\frac{\#\text{ documents}}{\#\text{ documents with word } j}\right)$$

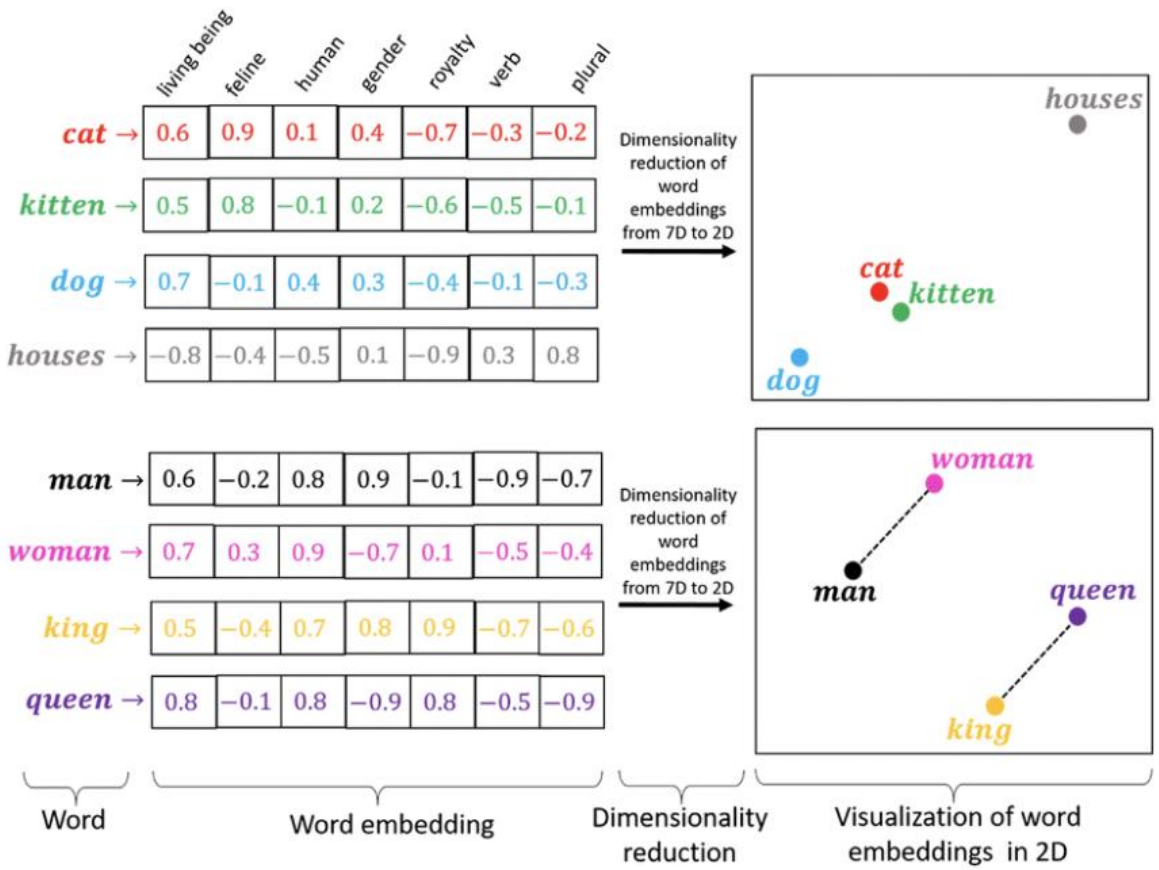
$$\text{idf}_{\text{of}} = \log\left(\frac{3}{3}\right) = 0$$

$$\text{idf}_{\text{is}} = \log\left(\frac{3}{2}\right) = 0.405$$

$$\text{idf}_{\text{goal}} = \log\left(\frac{3}{1}\right) = 1.098$$

Text Processing

- **Word Embedding**
  - Every word has a unique word embedding (or "vector"), and similar words end up with similar embedding values.
  - Word2vec from google: based on proximity
  - Glove: pre-trained (from Stanford)
  - Embedding() layer in deep learning: trained for the specific purpose




```
1 from tensorflow.keras.models import Sequential
2 from tensorflow.keras.layers import SimpleRNN, Dense, LSTM, GRU
3
4 model = Sequential()
5 model.add(Embedding(input_dim=10000, output_dim=8,
6                     input_length=maxlen))
7 model.add(LSTM(128))
8 model.add(Flatten())
9 model.add(Dense(1, activation='sigmoid'))
```

One-Hot encoding

- **One-Hot encoding**
  - Each word is written or encoded as one hot vector, with each one hot vector being **unique**.
  - One word is represented as a **vector,** therefore a sentence is represented as an array of vectors or a **matrix.**
  - A list of sentences will end up with a **three dimensional tensor** which can be fed to the Neural network.

The cat sat on the mat

The: [0 1 0 0 0 0 0]

cat: [0 0 1 0 0 0 0]

sat: [0 0 0 1 0 0 0]

on: [0 0 0 0 1 0 0]

the: [0 0 0 0 0 1 0]

mat: [0 0 0 0 0 0 1]

Bag of Words (BoW)

- **A representation of text that describes the occurrence of words within a document**
- **Just keep track of word counts and disregard the grammatical details (word order), that's why called "bag" of words**
- **Occurrence of words:**
  - Counts
  - Frequencies
- **Limitations**
  - Sparsity: little information with large representation space
  - Vocabulary: should be carefully designed due to impact on sparsity
  - Meaning: context and meaning of words (semantics) are disregarded
- **Need text cleaning techniques**
  - Ignoring cases, punctuations, frequent words ("a", "of", etc)
  - Fixing misspelled words
  - Reducing words to their stem (e.g. "play" from "playing"): stemization

Bag of Words (BoW)

- **N-gram**
  - N-token sequence of words
  - For example, 2-gram (more commonly called a bigram) is a two-word sequence of words (e.g. “please turn”, “turn your”, or “your homework”)
  - Often a simple bigram approach is better than a 1-gram bag-of-words model for tasks like documentation classification.
- **CountVectorizer()**
  - Document Term matrix
- **TfIdfVectorizer()**
  - Term frequency (Tf) : frequency of the words
  - Inverse Document frequency (Idf) : how rare the word is across documents

Document Term Matrix

- From http://www.datasciencecourse.org/notes/free_text/
  - Doc1 = "The goal of this lecture is to explain the basics of free text processing"
  - Doc2 = "The bag of words model is one such approach"
  - Doc3 = "Text processing via bag of words"

$$X = \begin{array}{c} \begin{matrix} \text{the} & \text{is} & \text{of} & \text{goal} & \text{lecture} & \text{bag} & \text{words} & \text{via} & \text{text} & & \text{approach} \end{matrix} \\ \begin{bmatrix} 2 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & \cdots & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & & 0 \end{bmatrix} \end{array} \begin{matrix} \\ \text{Document 1} \\ \text{Document 2} \\ \text{Document 3} \end{matrix}$$

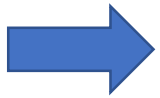
TFIDF

- **Term frequency**
  - Counts of each word in a document
  - tf $_{i,j}$ = frequency of word j in document i
- **Inverse document frequency**
  - Term frequencies tend to be "overloaded" with very common words ("the", "is", "of", etc)
  - Idea if inverse document frequency weight words negatively in proportion to how often they occur in the entire set of documents

$$\text{idf}_j = \log\left(\frac{\#\text{ documents}}{\#\text{ documents with word } j}\right)$$

TFIDF

$$X = \begin{bmatrix} \text{the} & \text{is} & \text{of} & \text{goal} & \text{lecture} & \text{bag} & \text{words} & \text{via} & \text{text} & & \text{approach} \\ 2 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & \cdots & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & & 0 \end{bmatrix} \begin{matrix} \\ \text{Document 1} \\ \text{Document 2} \\ \text{Document 3} \end{matrix}$$



$$X = \begin{bmatrix} \text{the} & \text{is} & \text{of} & \text{goal} & \\ 0.8 & 0.4 & 0 & 1.1 & \\ 0.4 & 0.4 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & \end{bmatrix}$$

$$\text{idf}_{\text{of}} = \log\left(\frac{3}{3}\right) = 0$$

$$\text{idf}_{\text{is}} = \log\left(\frac{3}{2}\right) = 0.405$$

$$\text{idf}_{\text{goal}} = \log\left(\frac{3}{1}\right) = 1.098$$

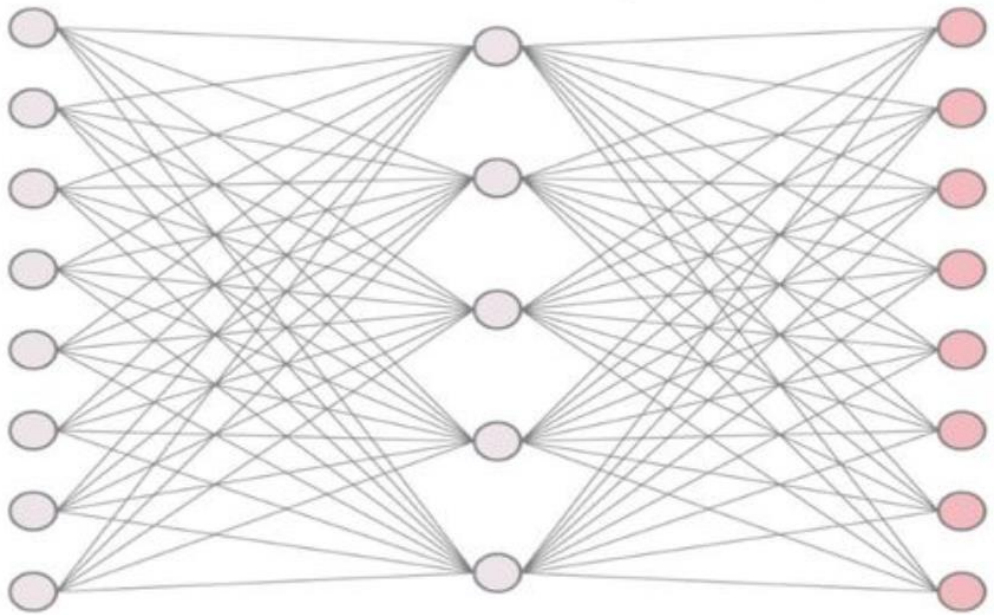
[note] In TfidfVectorizer(), if smooth_idf=true (default), the constant "1" is added to the numerator and denominator of the idf as if an extra document was seen containing every term in the collection exactly once, which prevents zero divisions: idf(t) = log [(1 + n) / (1 + df(t))] + 1

Word Embedding

- Techniques to compute Word embedding
  - Using supervised learning
    - Take an NLP problem and try to solve it. In that pursuit as a side effect, you get word embedding.
    - Keras Embedding() layer
  - Using self-supervised learning
    - Word2Vec (more popular)
    - Glove

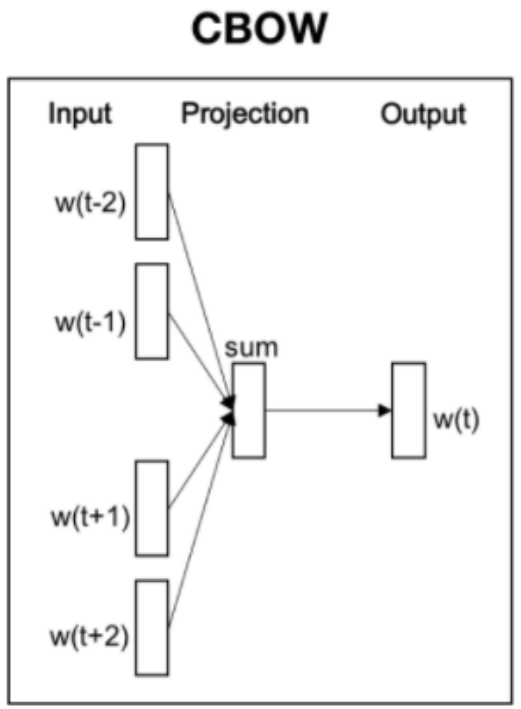
Word2Vec

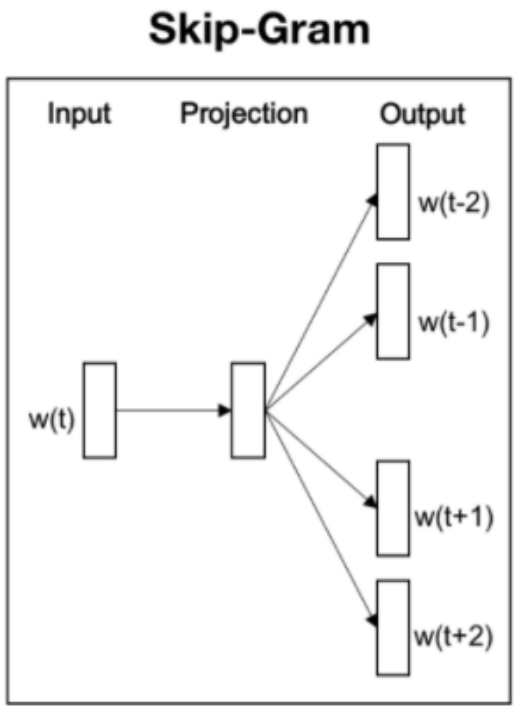
- **Predict words using context**
  - Concept: '비슷한 위치 (근처)에서 등장하는 단어들은 비슷한 의미를 가진다'
- **Word2Vec Neural Network model**
  - NN with single hidden layer
  - Often used for auto-encoder to compress input vector in hidden layer



Word2Vec

- **Two versions: CBOW(continuous bag of words) and Skip-Gram**
  - CBOW: 문맥 단어를 보고 기준 단어가 무엇인지 예측하는 모델
  - Skip-Gram: 기준 단어를 보고 어떤 문맥 단어가 등장할지 예측하는 모델 (in most cases, skip-gram is better)





https://wooono.tistory.com/244

Word2Vec - CBOW

- (ex) “I like playing football with my friends” with window = 2

center word	context words
[1,0,0,0,0,0,0]	[0,1,0,0,0,0,0] [0,0,1,0,0,0,0]
[0,1,0,0,0,0,0]	[1,0,0,0,0,0,0] [0,0,1,0,0,0,0] [0,0,0,1,0,0,0]
[0,0,1,0,0,0,0]	[1,0,0,0,0,0,0] [0,1,0,0,0,0,0] [0,0,0,1,0,0,0] [0,0,0,0,1,0,0]
[0,0,0,1,0,0,0]	[0,1,0,0,0,0,0] [0,0,1,0,0,0,0] [0,0,0,0,1,0,0] [0,0,0,0,0,1,0]
[0,0,0,0,1,0,0]	[0,0,1,0,0,0,0] [0,0,0,1,0,0,0] [0,0,0,0,0,1,0] [0,0,0,0,0,0,1]
[0,0,0,0,0,1,0]	[1,0,0,1,0,0,0] [0,0,0,0,1,0,0] [0,0,0,0,0,0,1]
[0,0,0,0,0,0,1]	[0,0,0,0,1,0,0] [0,0,0,0,0,1,0]

Training samples

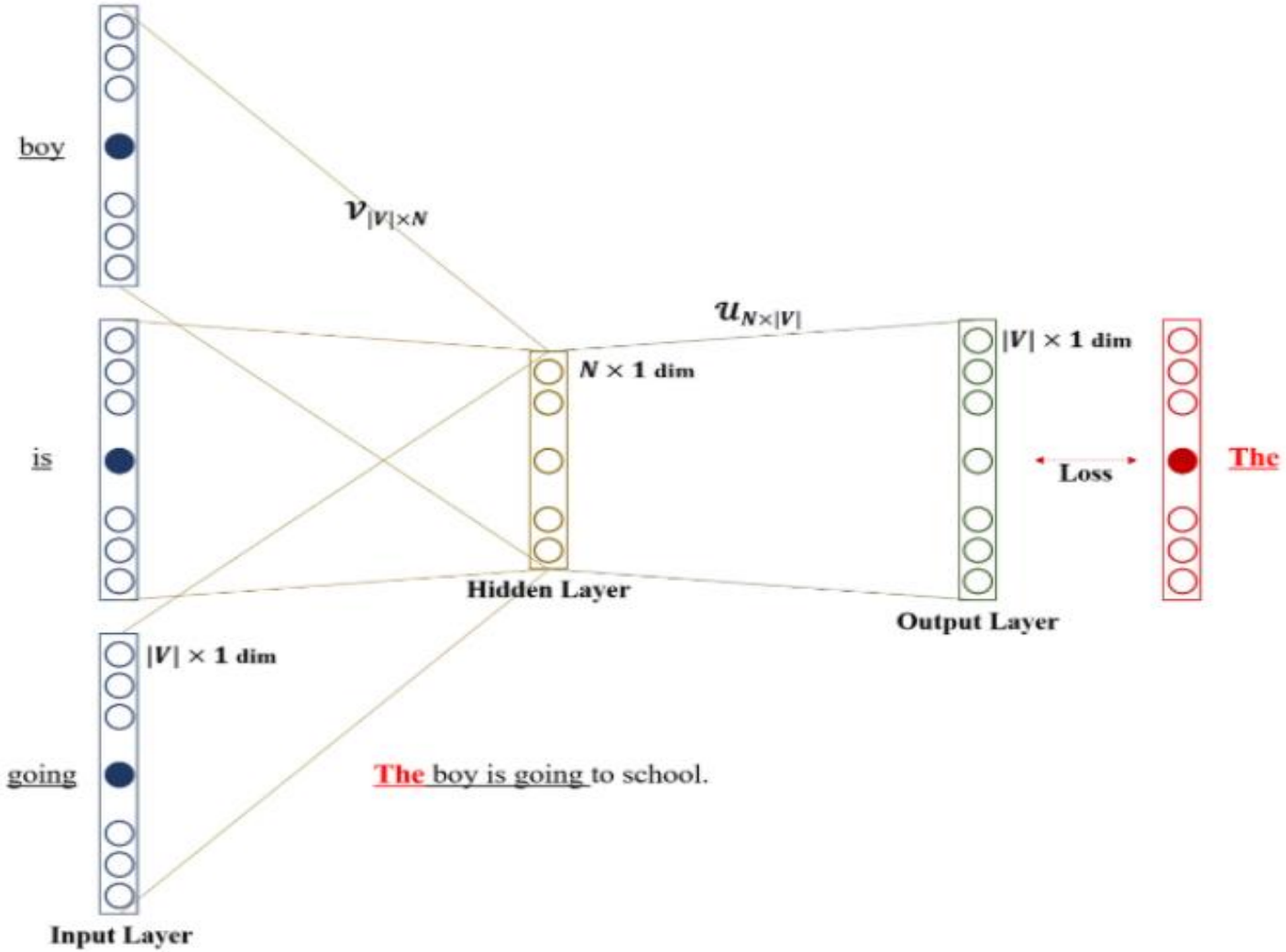
(I, like)
(I, playing)

(like, I)
(like, playing)
(like, football)

(playing, I)
(playing, like)
(playing, football)
(playing, with)
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

Word2Vec - CBOW

- **CBOW architecture**



Word2Vec - CBOW

- **CBOW Training**

1. (Input Layer) context word 각각의 one-hot vector를 입력

$$x^{(c-m)}, \cdots, x^{(c-1)}, x^{(c+1)}, \cdots, x^{(c+m)} \in \mathbb{R}^{|V|}$$

2. (Input Layer -> Hidden Layer) v = V * input (N-dim vector)

$$v_{c-m} = \mathcal{V}x^{(c-m)}, \cdots, v_{c+m} = \mathcal{V}x^{(c+m)} \in \mathbb{R}^n$$

3. (Hidden Layer) averaging v's of Step.2 -> v_hat

$$\hat{v} = \frac{v_{c-m} + \cdots + v_{c+m}}{2m} \in \mathbb{R}^n$$

4. (Hidden Layer -> Output Layer) z = U * v_hat

$$z = \mathcal{U}\hat{v} \in \mathbb{R}^{|V|}$$

5. (Output Layer) y_hat = softmax(z)

$$\hat{y} = softmax(z) \in \mathbb{R}^{|V|}$$

6. (Loss) error = y_hat - y(center word)

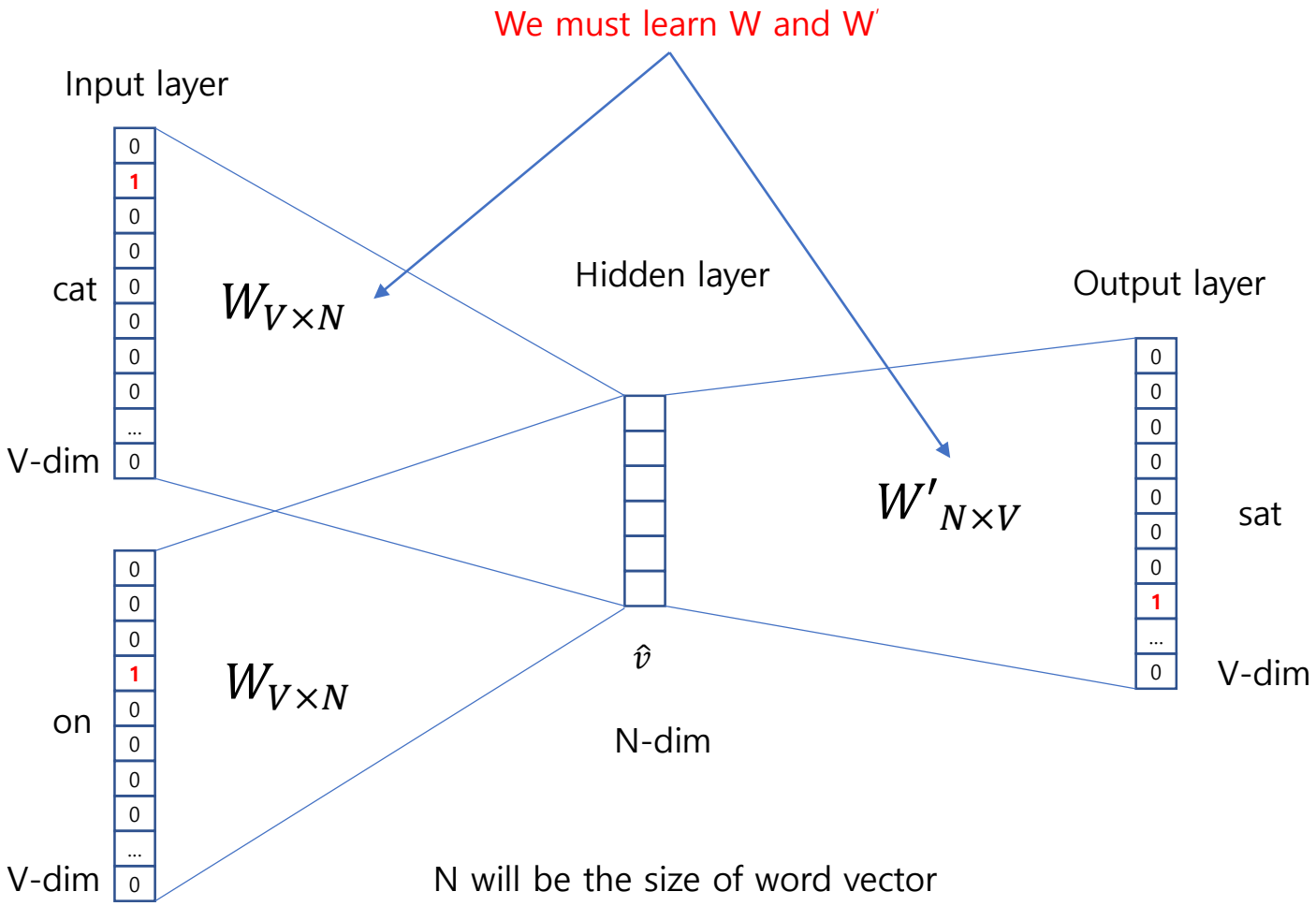
$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{|V|} y_j \log \hat{y}_j = -y_i \log \hat{y}_i$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(\hat{y}, y)$$

https://wooono.tistory.com/244

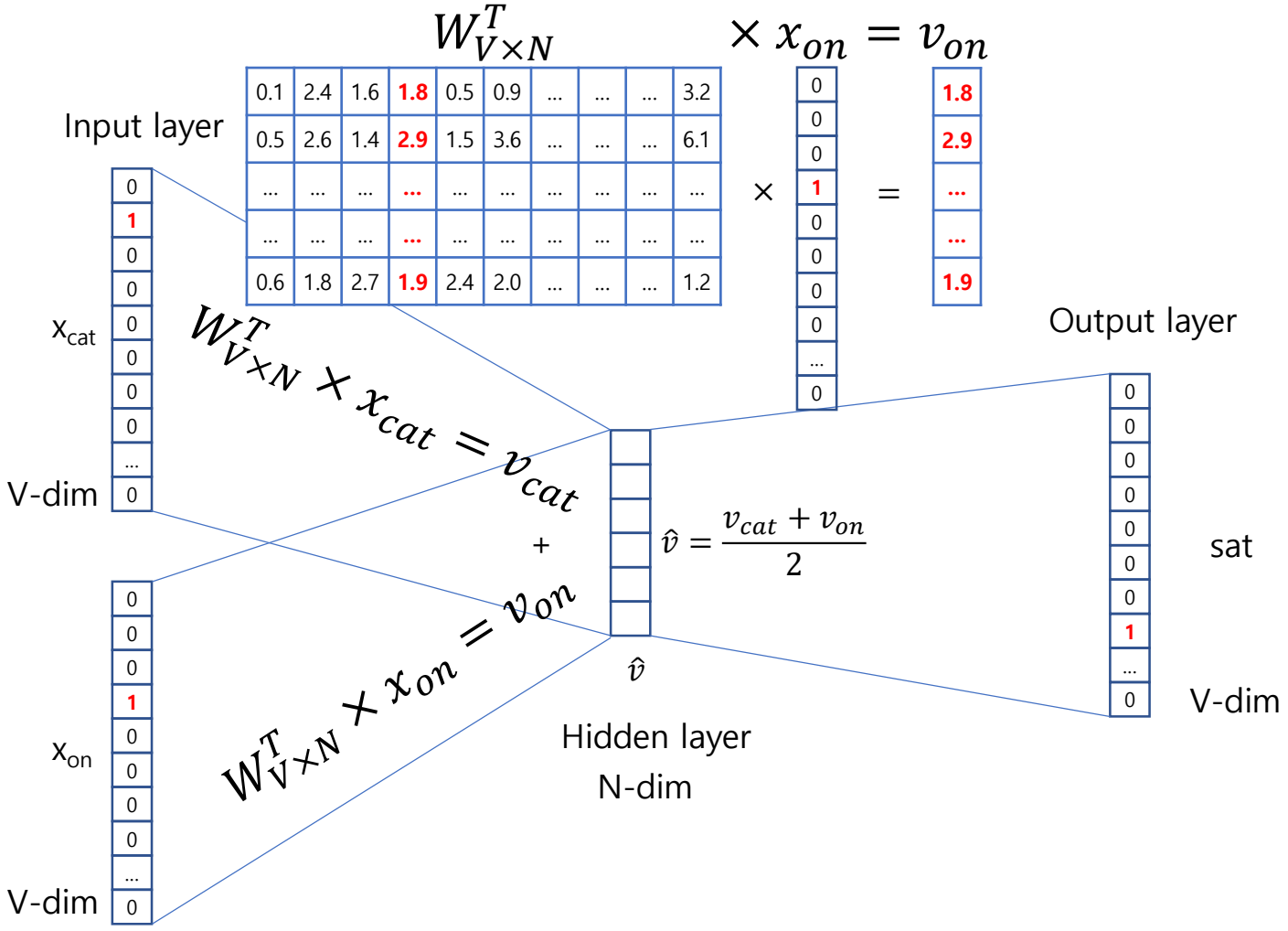
Word2Vec - CBOW

- **(ex) "The cat sat on floor" with window_size = 1**



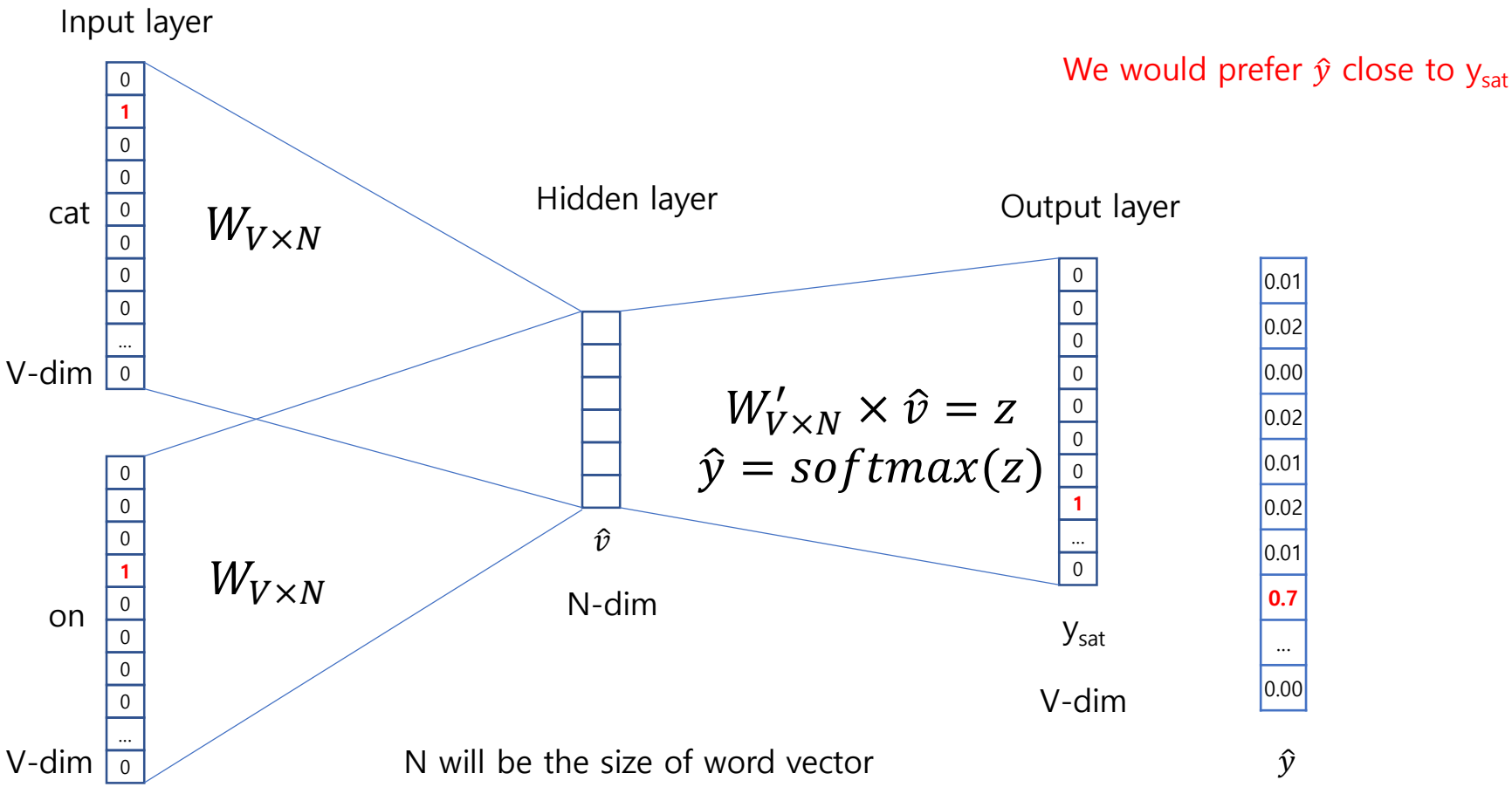
Word2Vec - CBOW

- **(ex) "The cat sat on floor" with window_size = 1**



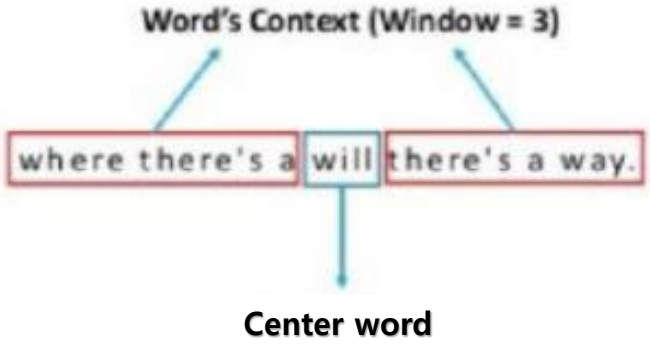
Word2Vec - CBOW

- **(ex) "The cat sat on floor" with window_size = 1**

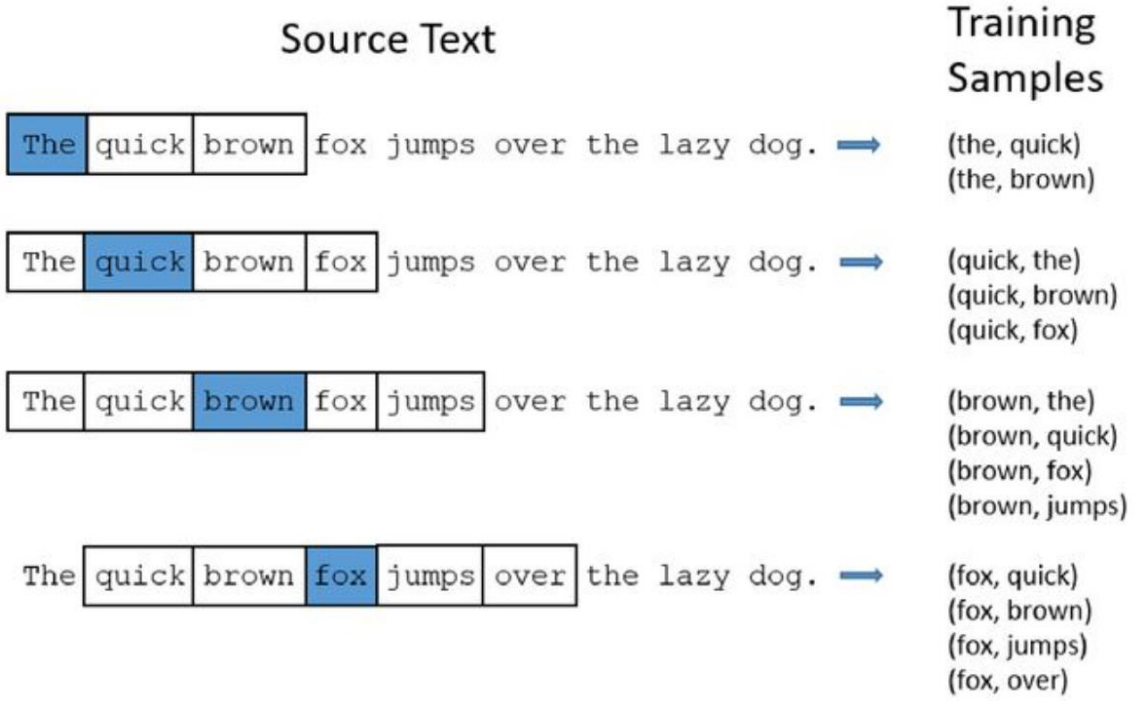


Word2Vec – Skip Gram

- **Predict neighbors** of a center word using Skip-gram model

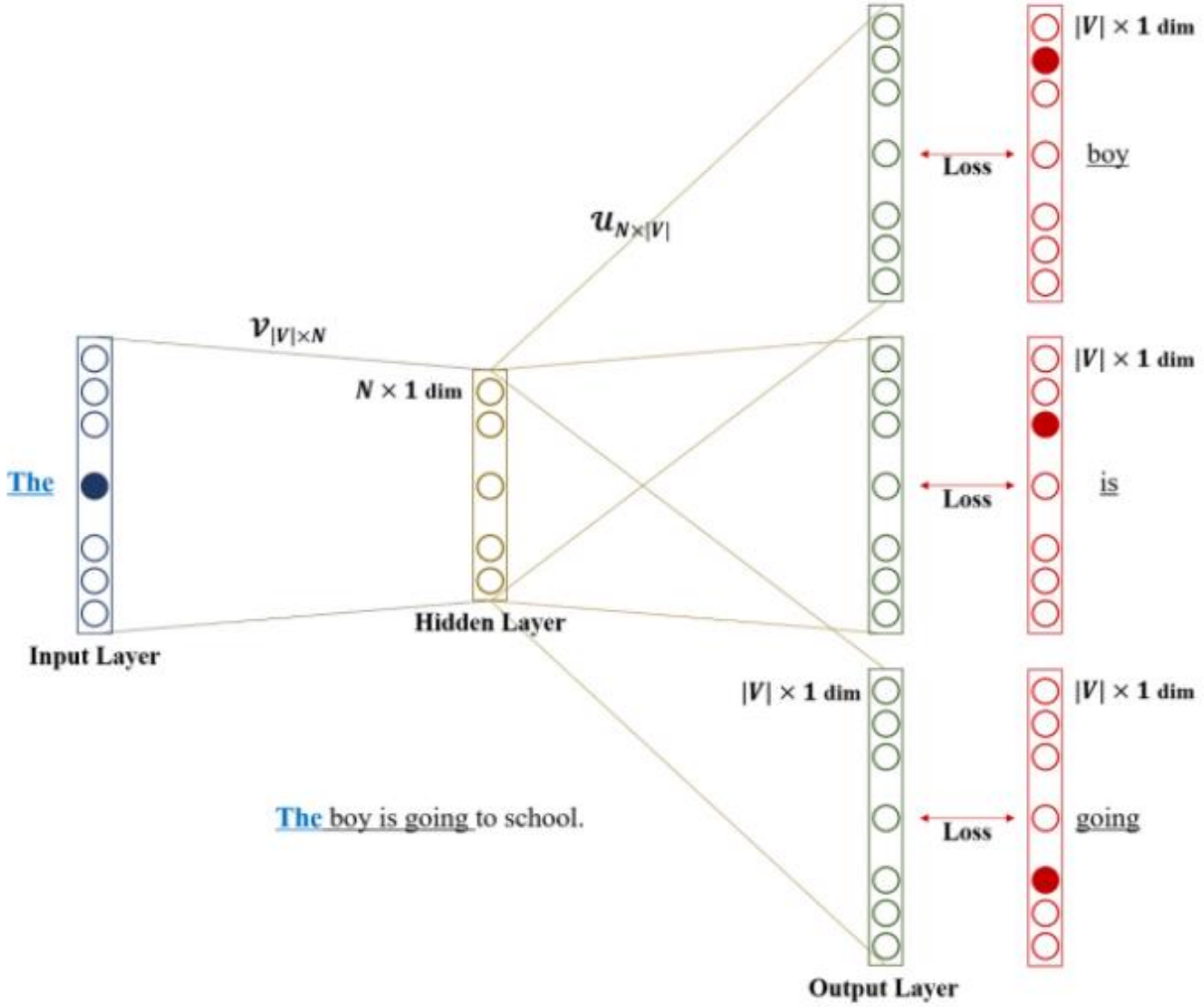


- Collection of Training samples
  - with a window of size 2



Word2Vec – Skip Gram

- **Skip-Gram architecture**



Word2Vec - Skip-Gram

- **Skip-Gram Training**

1. (Input Layer) center word 각각의 one-hot vector를 입력

$$x \in \mathbb{R}^{|V|}$$

2. (Input Layer -> Hidden Layer) v_c = V * input (N-dim vector)

$$v_c = \mathcal{V}x \in \mathbb{R}^n$$

3. (Hidden Layer -> Output Layer) z = U * v_c

$$z = \mathcal{U}v_c \in \mathbb{R}^{|V|}$$

4. (Output Layer) y_hat = softmax(z)

$$\hat{y} = softmax(z) \in \mathbb{R}^{|V|}$$

5. Step 3-4 를 모든 context word (주변단어) 에 대해 실행

6. (Loss) error = $\sum_{2m}$ (y_hat − y(context word))

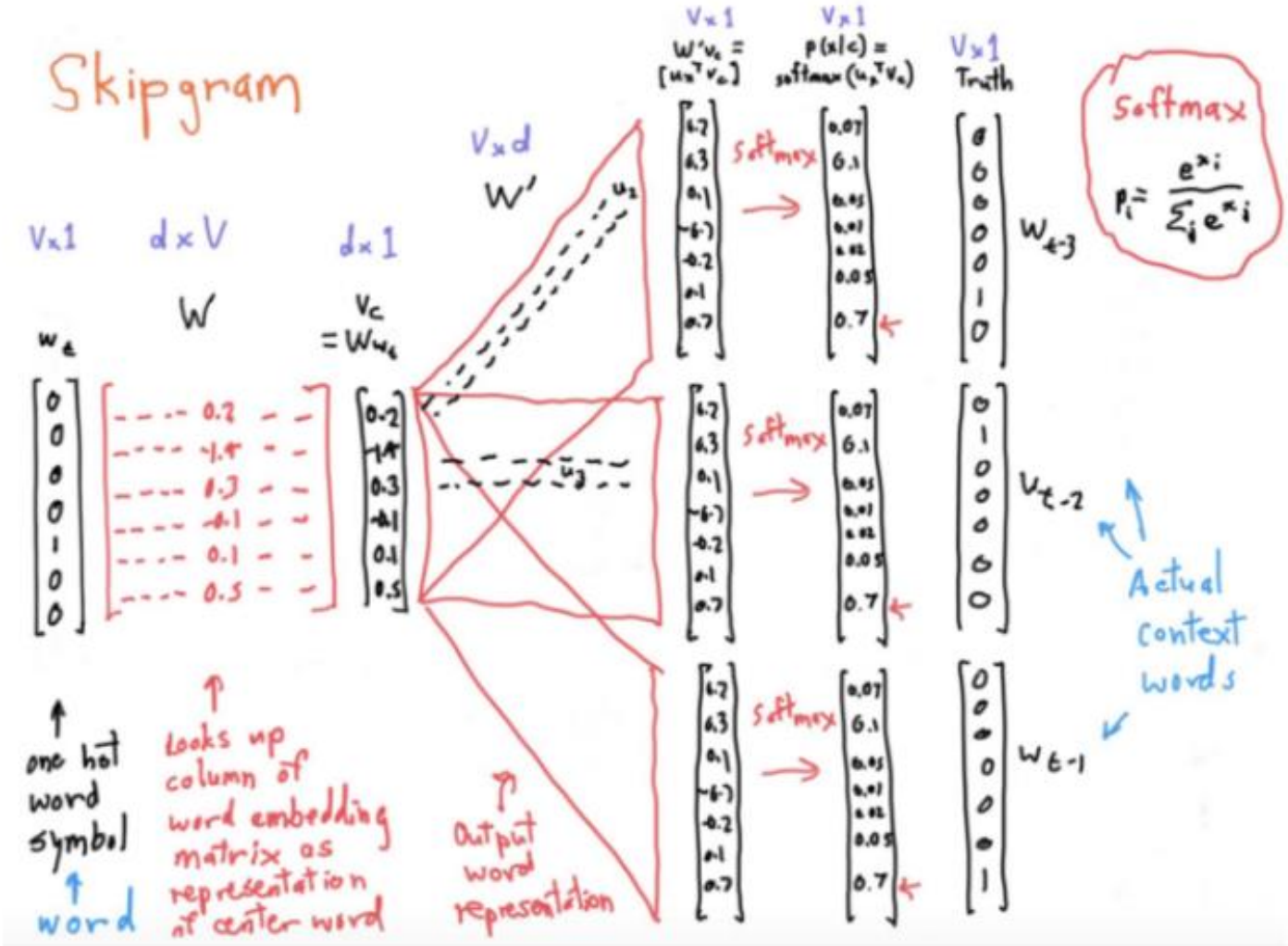
$$\mathcal{L}(\hat{y}, y) = -\sum_{j=0, j\neq m}^{2m}\sum_{k=1}^{|V|} y_k^{(c-j)} \log \hat{y}_k^{(c-j)} = -\sum_{j=0, j\neq m}^{2m} y^{(c-j)} \log \hat{y}^{(c-j)}$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(\hat{y}, y)$$

https://wooono.tistory.com/244

Word2Vec – Skip Gram

- **Summary**

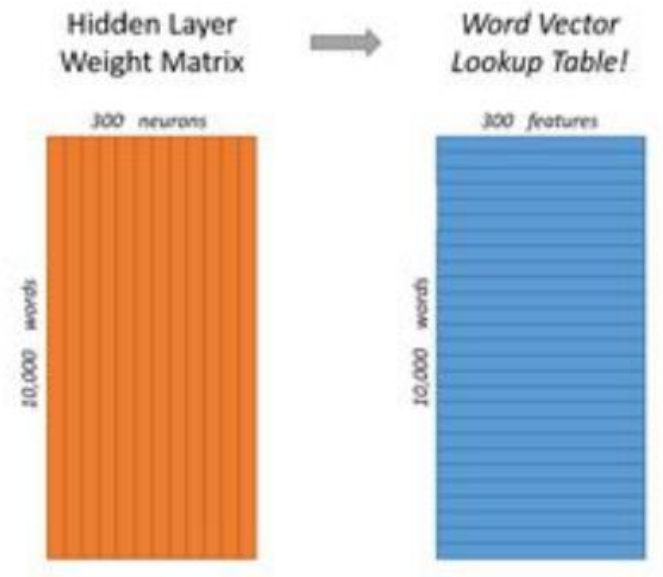


Word Embedding

- **Build a vocabulary of words from training documents**
  - (e.g.) a vocabulary of 10,000 unique words
- **Represent an input word as a one-hot vector**
  - This vector will have 10,000 components (one for every word in our vocabulary)
  - This vector will have "1" in the position corresponding to the word, say "ants", and 0s in all of the other positions.
- **No activation function for hidden layer neurons, but output neurons use softmax.**
- **When *training* the network with word pairs, the input is a one-hot vector and output is also a one-hot vector representing the output word.**
- **When evaluating the network on an input word, the output vector will actually be a probability distribution (i.e., a bunch of floating point values, not a one-hot vector).**

Word Embedding

- **Hidden layer is represented by a weight matrix with 10,000 rows (one for every word in our vocabulary) and 300 columns (one for every hidden neuron)**
  - 300 features
- **Rows of this weight matrix are the word vectors!**



Word Embedding

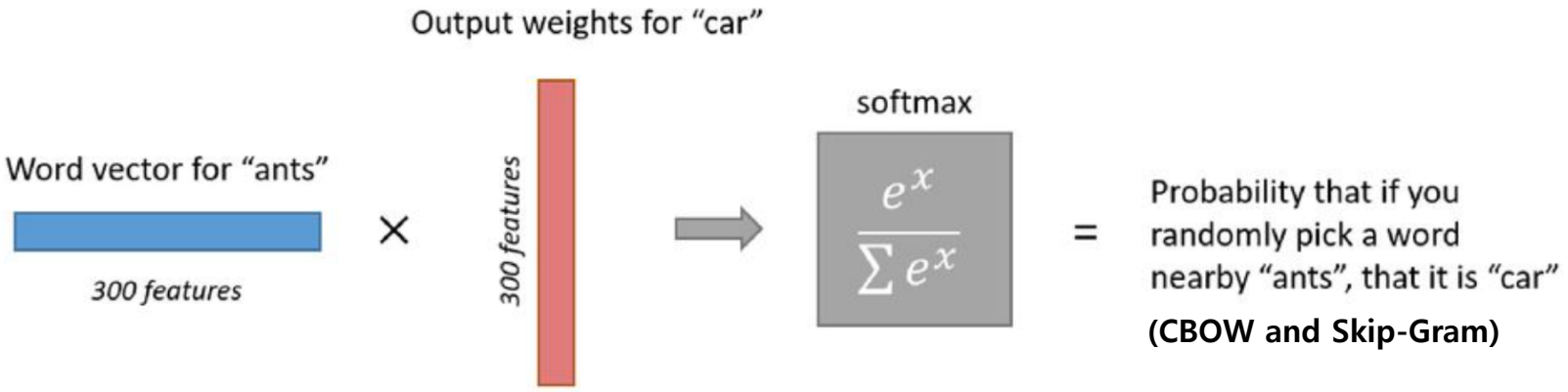
- **If we multiply a 1x10,000 one-hot vector by a 10,000x300 matrix, it will effectively just *select* the matrix row corresponding to the "1"**

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

- **Hidden layer is really just operating as a lookup table !!**
- **The output of the hidden layer is just the "word vector" for the input word.**

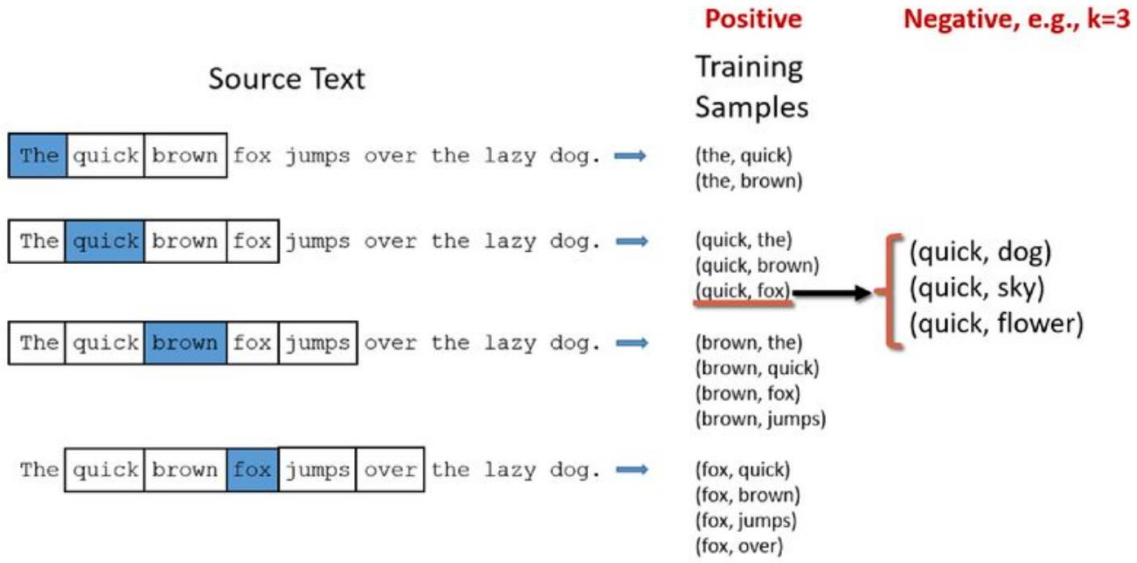
Word Embedding

- The output layer is a softmax regression classifier
- Each output neuron has a weight vector which it multiplies against the word vector from the hidden layer, then it applies the function exp(x) to the result. Finally, in order to get the outputs to sum up to 1, we divide this result by the sum of the results from *all* 10,000 output nodes.



Negative Sampling

- **Softmax function requires too much computation for its denominator (say, summation of 10,000 terms)**
- **When training, negative sampling considers only a small number of negative words (let's say 5) including the positive word for this summation (normally choose more than 20 based on some probability distribution)**
  - "negative" word is one for which network outputs "0" and "positive" word is one for "1"
  - 즉, 파라미터 조정할 때 전체 단어 집합이 아닌 일부 단어만 조정 (주변 단어(positive)들은 모두 조정하지만 그렇지 않은 단어들(negative)은 일부만 조정)



Word Analogies

- **Interesting result**

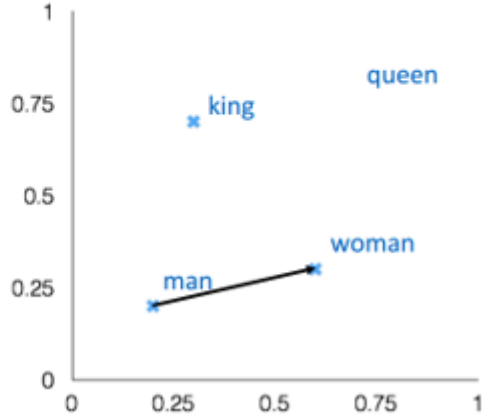
Test for linear relationships, examined by Mikolov et al. (2014)

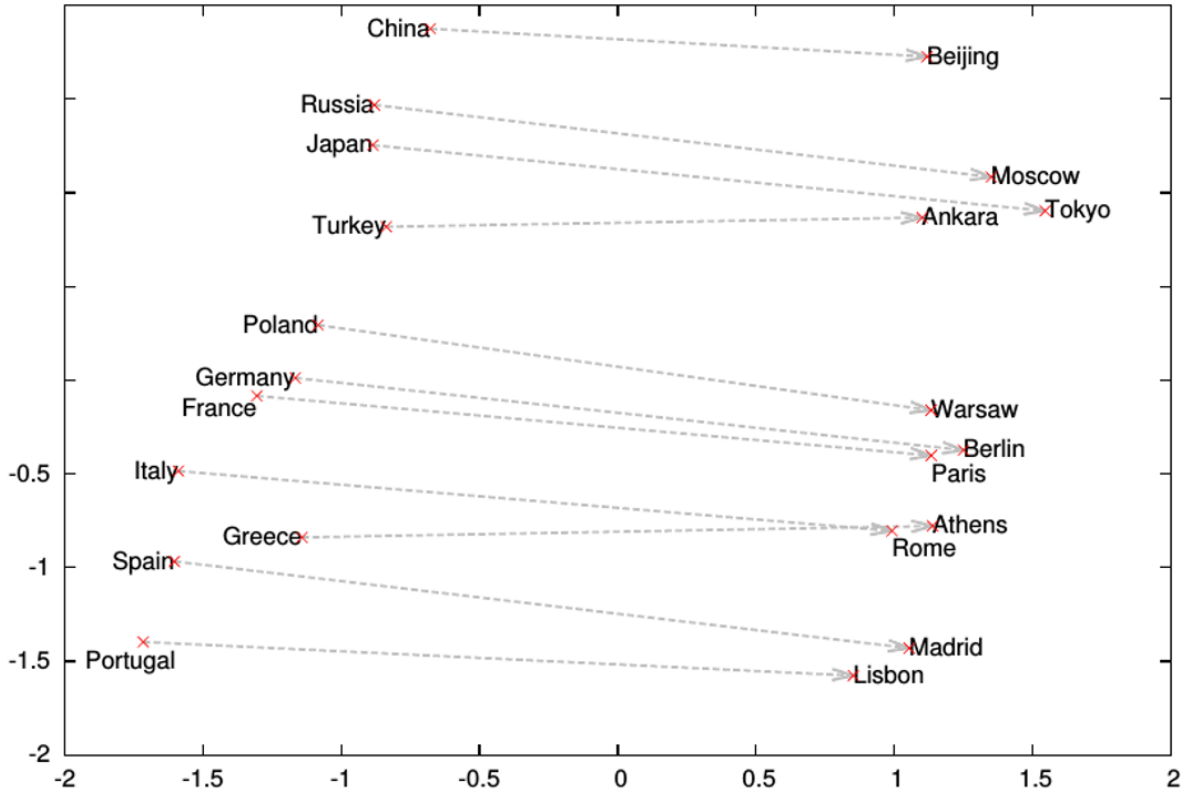
a:b :: c:?

$$d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

man:woman :: king:?

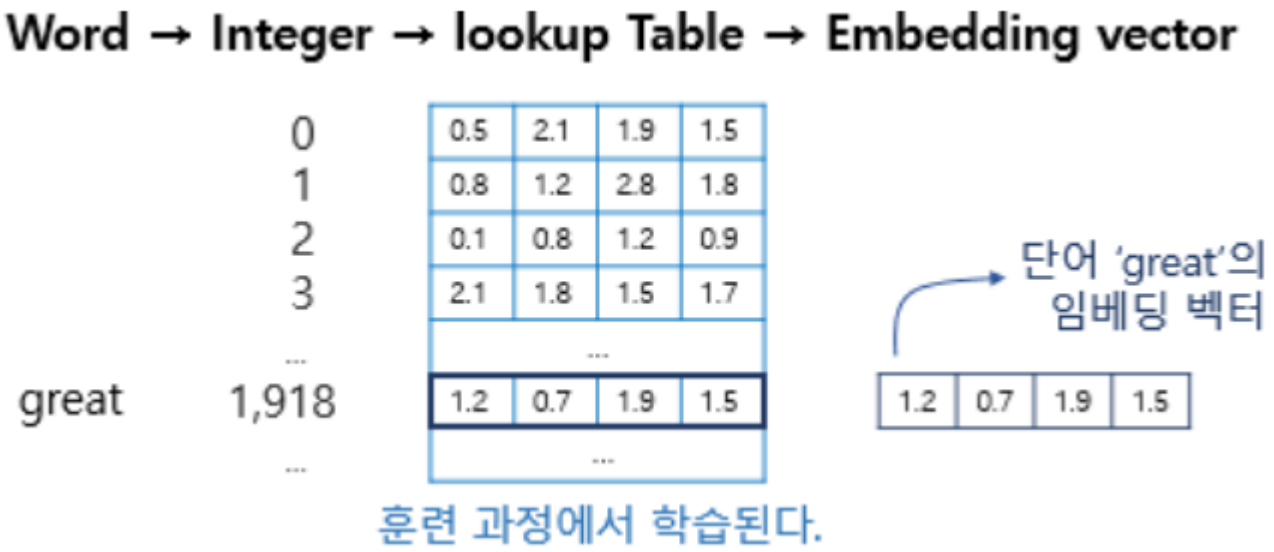
+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]





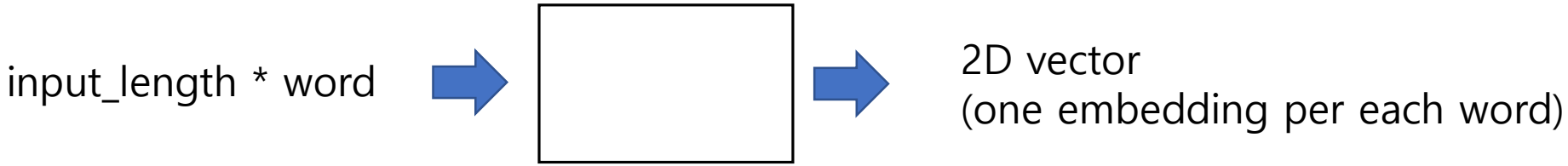
Keras Embedding() Layer

- **A word embedding can be learned as part of a deep learning model. This can be a slower approach, but tailors the model to a specific training dataset.**
- **Embedding layer is just a Look-up Table.**
  - All the words should be encoded as integers to be used as input.



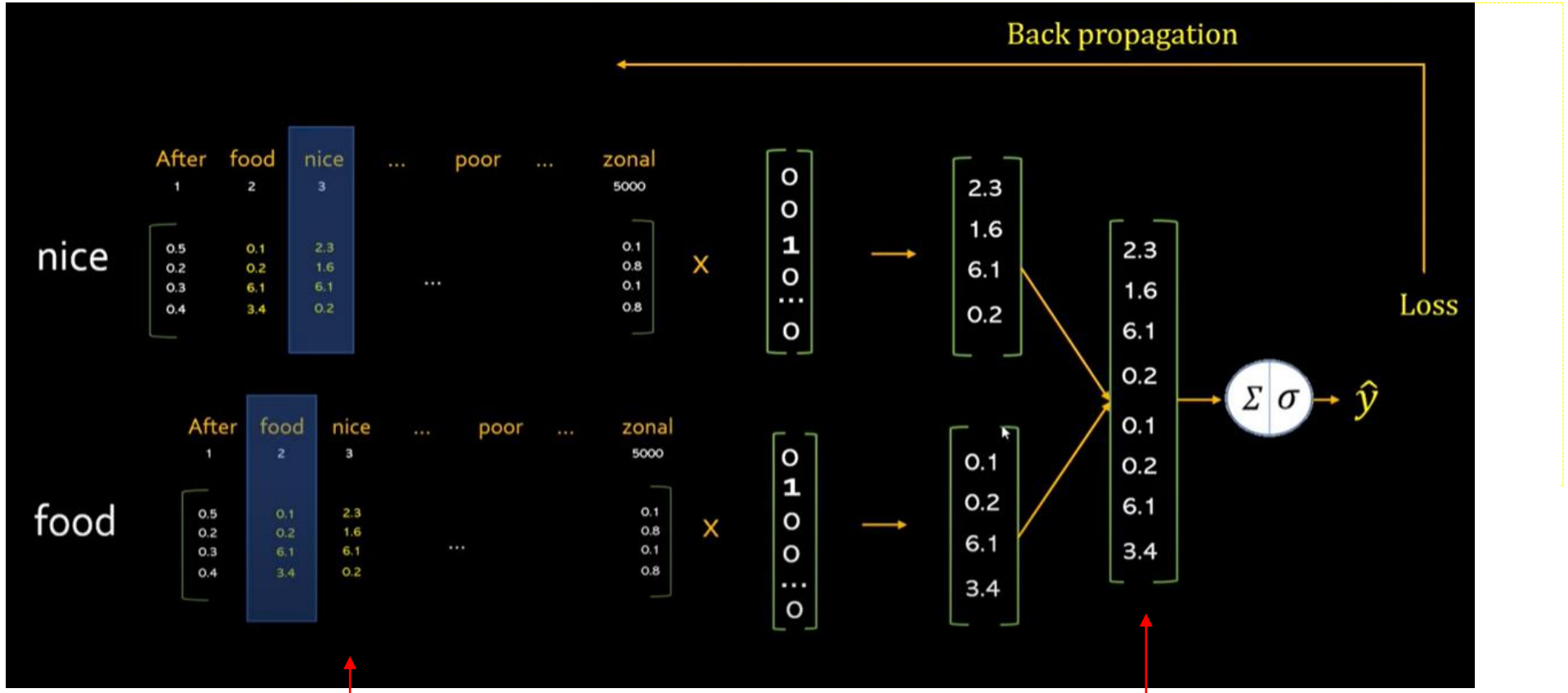
Keras Embedding() Layer

- **Flexible layer**
  - **Can be used alone, or can be used as part of deep learning model**
  - **Can also be used to load a pre-trained word embedding model**
- **Embedding layer is defined as the first layer of a network.**
- **To be connected directly to Dense layer, the 2D output must be flattened to 1D.**
- **3 arguments**
  - **input_dim: size of the vocabulary**
  - **output_dim: size of the vector space**
  - **input_length: length of input sequences (e.g. if all of your input documents are comprised of 1000 words, this would be 1000.)**



(*) The result of Embedding is just a permutation of inner trainable weights.
Where the permutation is denoted by "indices" in your input array. (see the next slide)

Inside of Keras Embedding() Layer

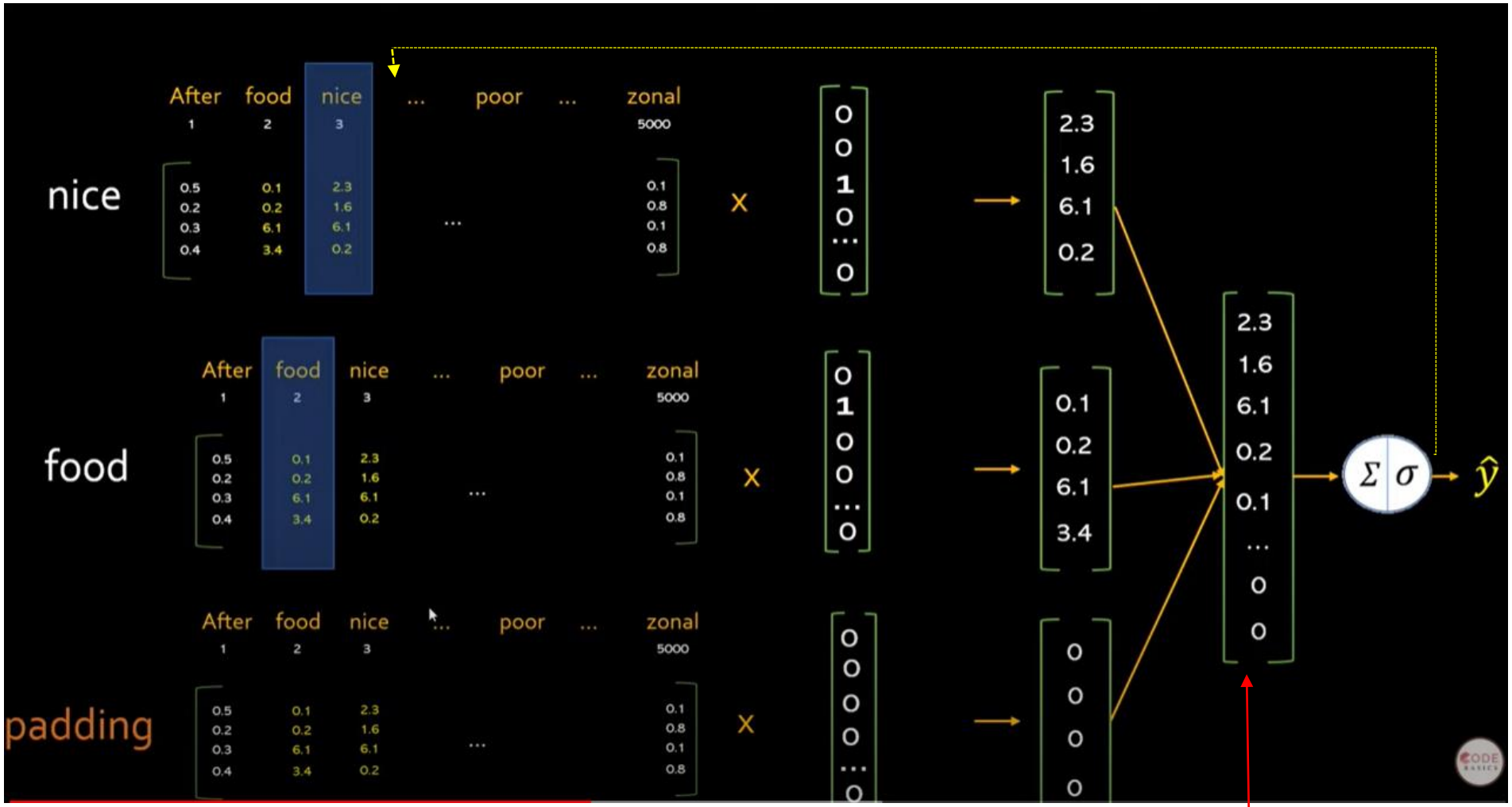


Values of the matrix are initially randomly selected, and will keep changing during the training stage

number of neurons is fixed

https://www.youtube.com/watch?v=Fuw0wv3X-0o

Inside of Keras Embedding() Layer



https://www.youtube.com/watch?v=Fuw0wv3X-0o

Inside of Keras Embedding() Layer

- **The matrix after training**

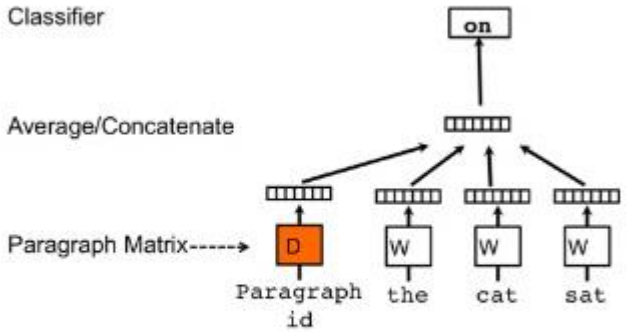
similar words are embedded similarly

Doc2Vec

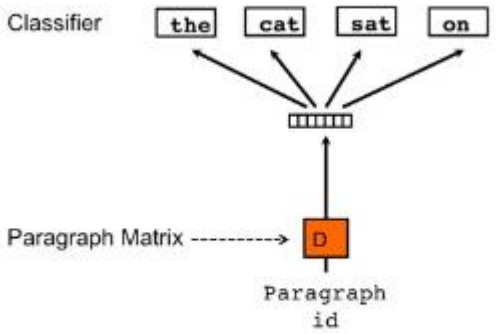
- **Embedding the paragraphs (documents) to vectors**
- **Extension of Word2Vec**
- **Basic Idea**
  - act as if a document has another floating word-like vector, which contributes to all training predictions, and is updated like other word-vectors, but we will call it a doc-vector.
- **Two implementations**
  - Paragraph Vector – Distributed Memory (PV-DM)
  - Paragraph Vector – Distributed bag of Words (PV-DBOW)

Doc2Vec

- **Example: “The cat sat on the mat.” in paragrah_1**
  - (Training data)
  - window size k = 3
  - [paragraph_1, the cat, sat] – on
  - [paragraph_1, cat, sat, on] – the
  - [paragraph_1, sat, on, the] – mat
- **PY-D2V and PV-DBOW**



Paragraph vector with distributed memory (PV-DM)



Distributed bag of words version (PVDBOW)