NLP Basic

Distributed Representation

CONTENT

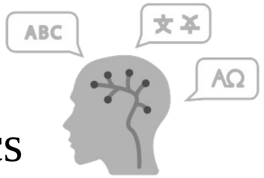
1 – Distributed Representation

❖ Text Representation

Natural Language Understanding (NLU)

a computer’s ability to understand language

- ❑ Syntax
- ❑ Semantics
- ❑ Phonology
- ❑ Pragmatics
- ❑ Morphology



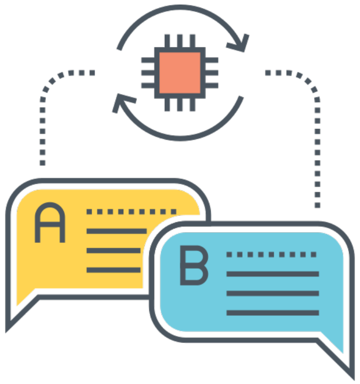
I go to school

Convert

[0 0 0 0 1 0 1 0

Natural Language Generation (NLG)

generate natural language by a computer



1 – Distributed Representation

❖ Review: Basic Representation

- One-hot encoding, BOW, TF-IDF
- Sparse Representation
- Not capture the meaning
- OOV Problem

$$
\begin{array}{ll}
 & \begin{array}{cccccccc} \text{Rome} & \text{Paris} & & & & & & \text{word V} \end{array} \\
\text{Rome} = & [1,\ 0,\ 0,\ 0,\ 0,\ 0,\ \ldots,\ 0] \\
\text{Paris} = & [0,\ 1,\ 0,\ 0,\ 0,\ 0,\ \ldots,\ 0] \\
\text{Italy} = & [0,\ 0,\ 1,\ 0,\ 0,\ 0,\ \ldots,\ 0] \\
\text{France} = & [0,\ 0,\ 0,\ 1,\ 0,\ 0,\ \ldots,\ 0]
\end{array}
$$

1 – Distributed Representation

- Distributional Similarity: meaning of a word defined by context

  Mùa **xuân** là tết trồng cây
  Làm cho đất nước càng ngày càng **xuân**

- Distributional Hypothesis: similar context => similar meaning

  Hai **cha** con bước đi trên cát
  Hai **bố** con bước đi trên cát

1 – Distributed Representation

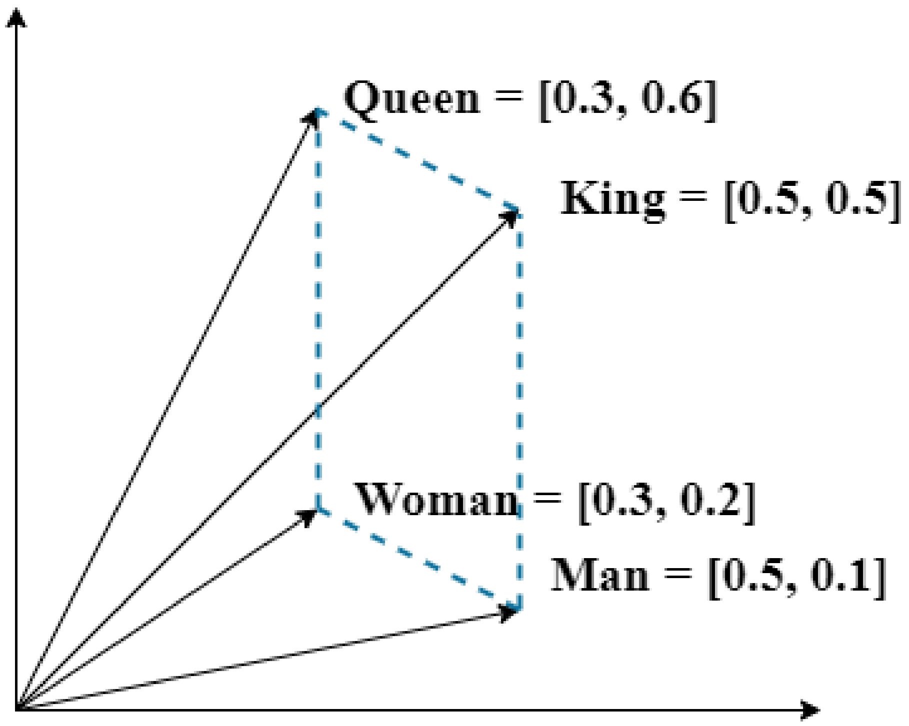
- Distributional Representation: dense vectors (low dimensional, hardly any zeros)
- Embedding: map words into a distributed representation space
- Vector Semantics: based on distributional properties of words in a large corpus

man →	0.6	−0.2	0.8	0.9	−0.1	−0.9	−0.7
woman →	0.7	0.3	0.9	−0.7	0.1	−0.5	−0.4
king →	0.5	−0.4	0.7	0.8	0.9	−0.7	−0.6
queen →	0.8	−0.1	0.8	−0.9	0.8	−0.5	−0.9

cat →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
kitten →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
dog →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8

1 – Distributed Representation

- Based on neural network: Word2Vec, Glove, Fasttext,…
- Word2Vec: embedding dimension (50 – 100)
- Capture word analogy relationships



1 – Distributed Representation

❖ Embedding Layer

» Keras API reference / Layers API / Core layers / Embedding layer

Embedding layer

`Embedding` class

```
tf.keras.layers.Embedding(
    input_dim,
    output_dim,
    embeddings_initializer="uniform",
    embeddings_regularizer=None,
    activity_regularizer=None,
    embeddings_constraint=None,
    mask_zero=False,
    input_length=None,
    **kwargs
)
```

EMBEDDING

CLASS `torch.nn.Embedding(num_embeddings, embedding_dim, padding_idx=None, max_norm=None, norm_type=2.0, scale_grad_by_freq=False, sparse=False, _weight=None, device=None, dtype=None)` [SOURCE]

A simple lookup table that stores embeddings of a fixed dictionary and size.

This module is often used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings.

Parameters

- **num_embeddings** (*int*) – size of the dictionary of embeddings
- **embedding_dim** (*int*) – the size of each embedding vector
- **padding_idx** (*int, optional*) – If specified, the entries at `padding_idx` do not contribute to the gradient; therefore, the embedding vector at `padding_idx` is not updated during training, i.e. it remains as a fixed "pad". For a newly constructed Embedding, the embedding vector at `padding_idx` will default to all zeros, but can be updated to another value to be used as the padding vector.
- **max_norm** (*float, optional*) – If given, each embedding vector with norm larger than `max_norm` is renormalized to have norm `max_norm`.
- **norm_type** (*float, optional*) – The p of the p-norm to compute for the `max_norm` option. Default `2`.
- **scale_grad_by_freq** (*boolean, optional*) – If given, this will scale gradients by the inverse of frequency of the words in the mini-batch. Default `False`.
- **sparse** (*bool, optional*) – If `True`, gradient w.r.t. `weight` matrix will be a sparse tensor. See Notes for more details regarding sparse gradients.

1 – Distributed Representation

❖ Embedding Layer

Embedding matrix

I
n
p
u
t

m
a
t
r
i
x
I

0	0.1	3.1
1	0.5	2.5
2	1.3	0.6
3	0.4	0.1
4	0.7	1.4
5	2.3	1.7
6	2.5	2.5

[ ]

Vocab size = 7

1 – Distributed Representation

❖ Embedding Layer

Input matrix
Index-based Representation

$$\begin{bmatrix} 0 & 4 & 2 & 3 & 1 \\ 5 & 6 & 1 & 2 & 3 \end{bmatrix}$$

Input shape: 2x5

Embedding matrix

0	0.1	3.1
1	0.5	2.5
2	1.3	0.6
3	0.4	0.1
4	0.7	1.4
5	2.3	1.7
6	2.5	2.5

Vocab size = 7

$\xrightarrow{\text{Select Operation}}$

$$\begin{bmatrix} w[0] & w[4] & w[2] & w[3] & w[1] \\ w[5] & w[6] & w[1] & w[2] & w[3] \end{bmatrix}$$

1 – Distributed Representation

❖ Embedding Layer

Input matrix
Index-based Representation

$$\begin{bmatrix} 0 & 4 & 2 & 3 & 1 \\ 5 & 6 & 1 & 2 & 3 \end{bmatrix}$$

Input shape: 2x5

Embedding matrix

0	0.1	3.1
1	0.5	2.5
2	1.3	0.6
3	0.4	0.1
4	0.7	1.4
5	2.3	1.7
6	2.5	2.5

Vocab size = 7

Select Operation

$$\begin{bmatrix} w[0] & w[4] & w[2] & w[3] & w[1] \\ w[5] & w[6] & w[1] & w[2] & w[3] \end{bmatrix}$$

Output matrix

	3.1	1.4	0.6	0.1	2.5
0.1	0.7	1.3	0.4	0.5	
2.3	2.5	0.5	1.3	0.4	

Input shape: 2x5x2

2 – Word2Vec

- **Paper - Word2Vec**:
  - 2 algorithms:
    - Continuous bag-of-words (CBOW)
    - Skip-gram
  - 2 training methods:
    - Negative Sampling
    - Hierarchical SoftMax

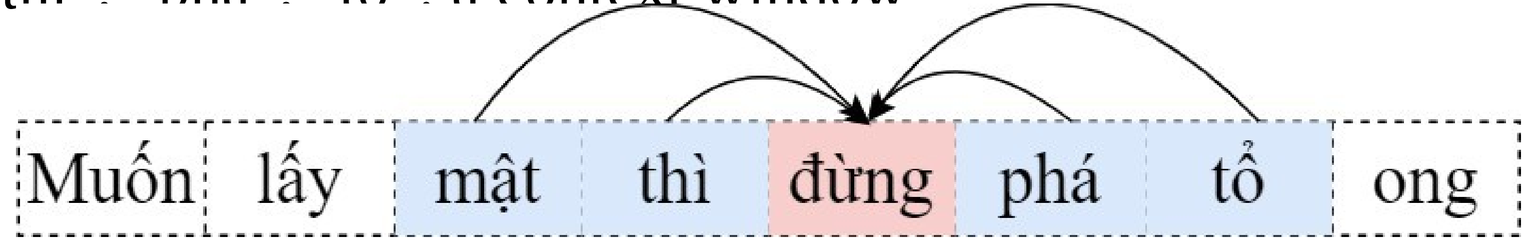
2 – Word2Vec

❖ Word2Vec: CBOW

- Predict a center word from the surrounding context in term of word vectors
- Example: “**Muốn lấy mật thì đừng phá tổ ong**”

“đừng”: a central target word

“mật”, “thì”, “phá”, “tổ”: a context window size = 2

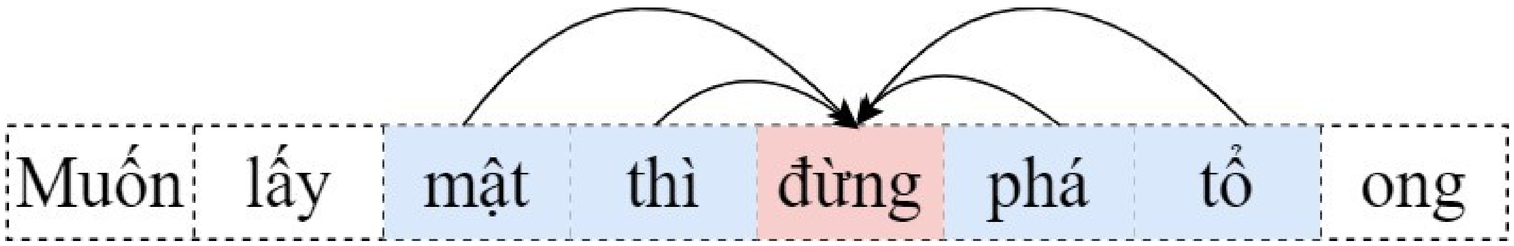


2 – Word2Vec

❖ Word2Vec: CBOW

- CBOW model: concerned with the conditional probability of generating the target word “đừng” based on the context words “mật”, “thì”, “phá”, “tổ”

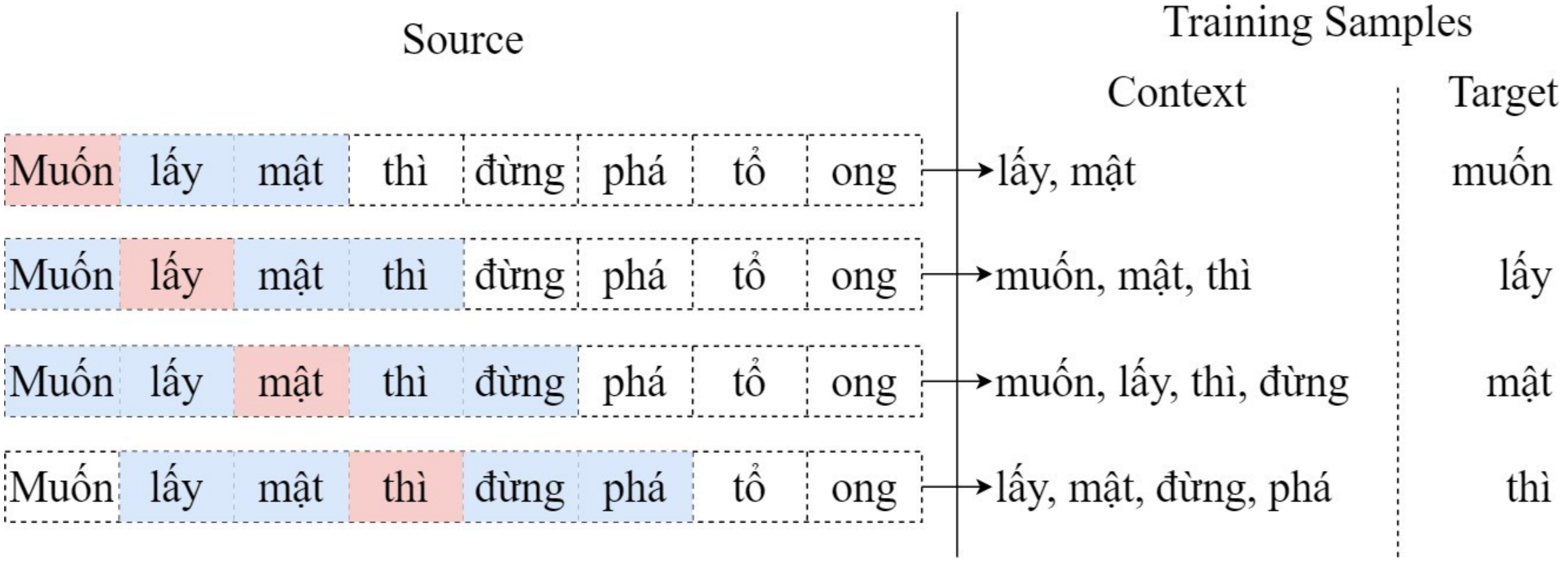
$P(\text{đừng}|\text{mật, thì, phá, tổ})$



2 – Word2Vec

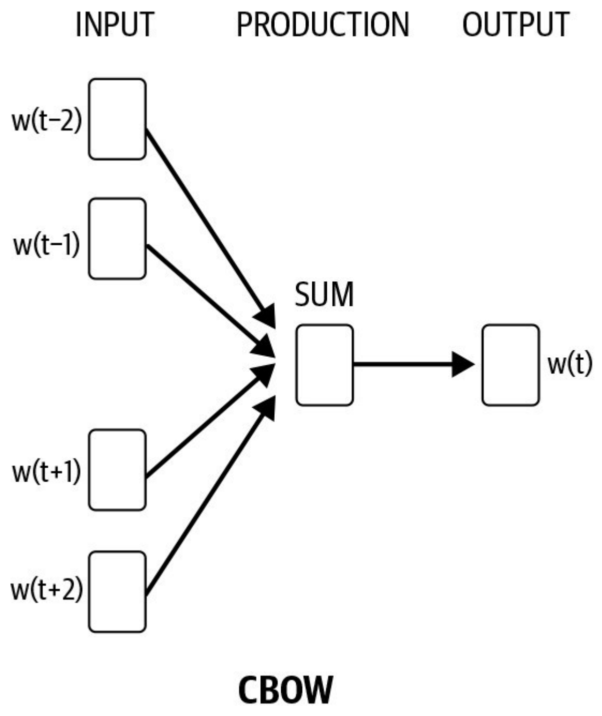
❖ Word2Vec: CBOW

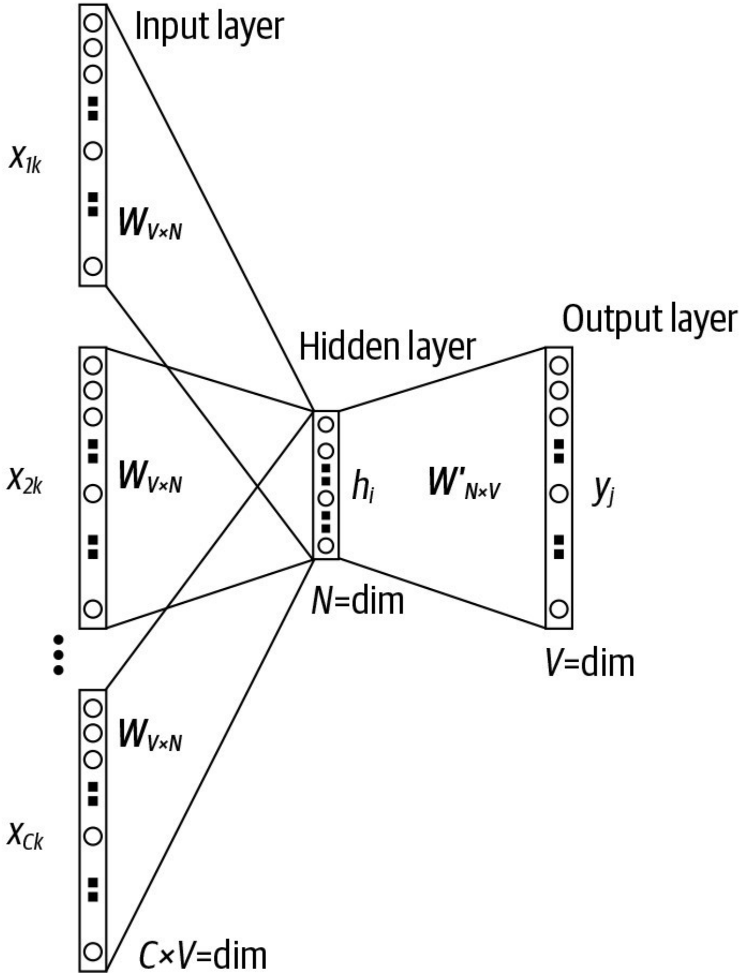
➢ Example: “Muốn lấy mật thì đừng phá tổ ong”



2 – Word2Vec

❖ Word2Vec: CBOW





2 – Word2Vec

❖ Word2Vec: CBOW

- w_i: word i from vocabulary V
- each word w_i is represented as two d-dimension vectors:
  - $v_i \in \mathbb{R}^d$: the context word vector
  - $u_i \in \mathbb{R}^d$: the central target word vector

2 – Word2Vec

❖ Word2Vec: CBOW

- w_c: central target word, indexed as c
- $w_{o1},\ldots,w_{o2m}$: context words, indexed as $o_1,\ldots, o_{2m}$ in the vocabulary with m: window size
- The conditional probability:

$$P(w_c|W_o) = \frac{\exp(u_c^T\bar{v}_o)}{\sum_{i\in V}\exp(u_i^T\bar{v}_o)}$$

$$W_o = \{w_{o1}, \ldots, w_{o2m}\} \text{ and } \bar{v} = (v_{o1} + \cdots + v_{o2m})/(2m)$$

2 – Word2Vec

❖ Word2Vec: CBOW

CBOW Model Training

- Optimization objective function:

$$-\log P(w_c|W_o) = -u_c^T \overline{v_o} + \log \sum_{i=1}^{|V|} \exp\left(u_i^T \overline{v_o}\right)$$

- Use SGD to update all relevant word vectors *v* and *u*
- **NOTE:** CBOW model use the context word vector as the representation vector for a word

2 – Word2Vec

❖ Word2Vec: CBOW

Cross entropy

- Loss function:

$$H(\hat{y}, y) = -\sum_{i=1}^{|V|} y_i \log(\hat{y}_i)$$

- If y is a one-hot vector:

$$H(\hat{y}, y) = -y_i \log(\hat{y}_i)$$

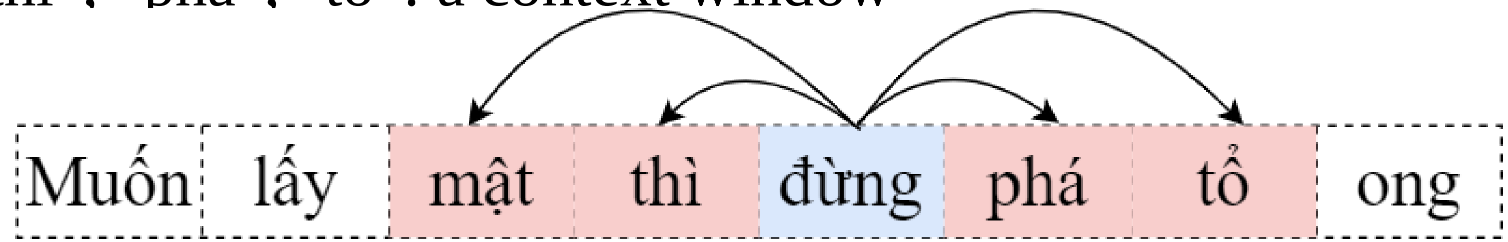
2 – Word2Vec

❖ Word2Vec: Skip-gram

- Predict the distribution (probability) of context words from a center word
- Example: “**Muốn lấy mật thì đừng phá tổ ong**”

“đừng”: a central target word

“mật”, “thì”, “phá”, “tổ”: a context window size of



2 – Word2Vec

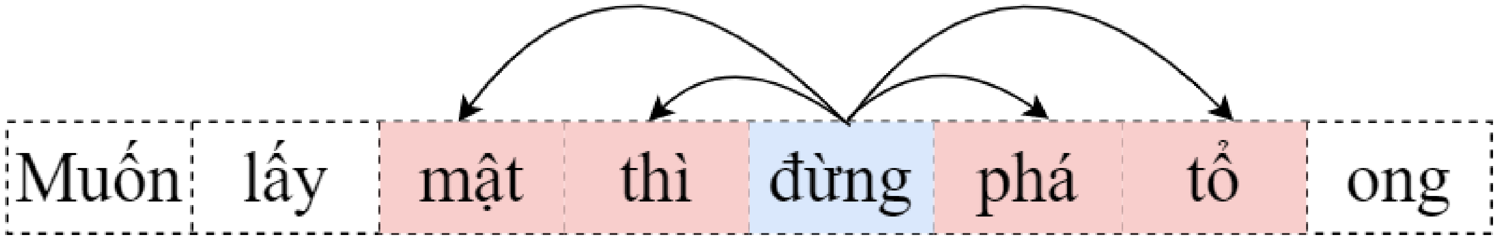
❖ Word2Vec: Skip-gram

- Skip-gram model: The conditional probability for generating the context words "mật", "thì", "phá", "tổ" based on the central word "đừng"

$$P(\text{mật}, \text{thì}, \text{phá}, \text{tổ} | \text{đừng})$$

- Assume: the context words are generated independently:

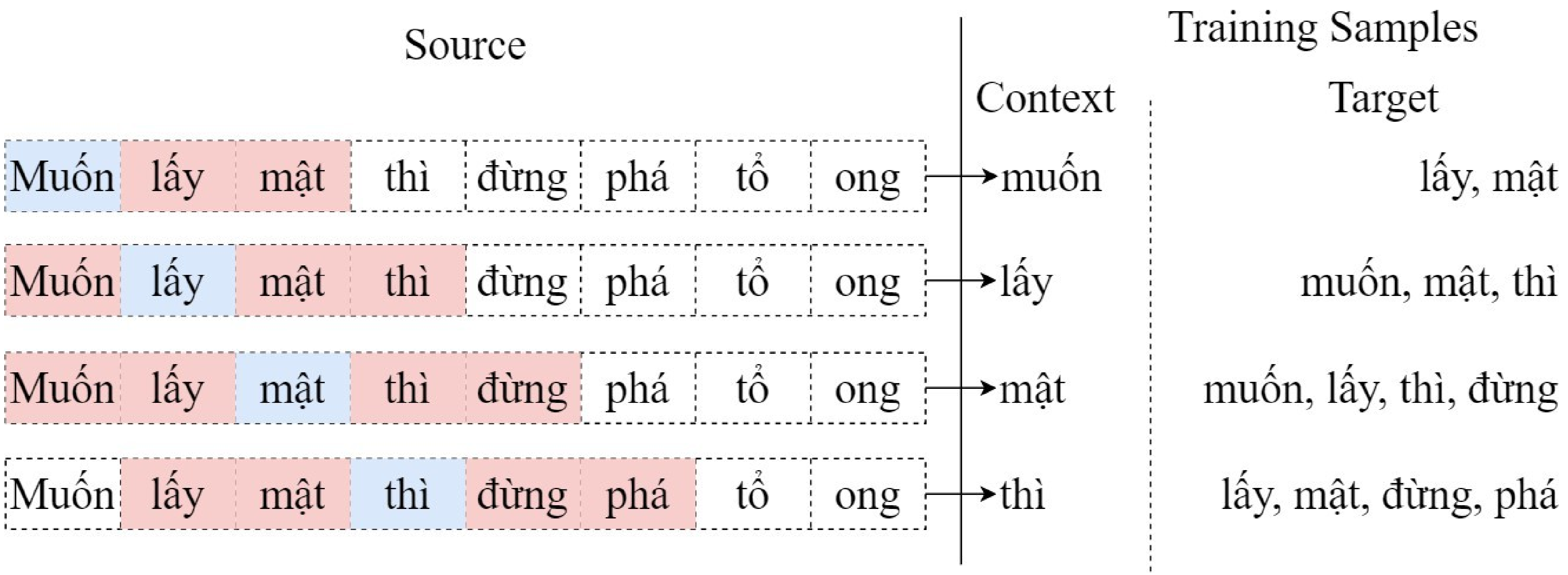
$$P(\text{mật}|\text{đừng}).\,P(\text{thì}|\text{đừng}).\,P(\text{phá}|\text{đừng}).\,P(\text{tổ}|\text{đừng})$$



2 – Word2Vec

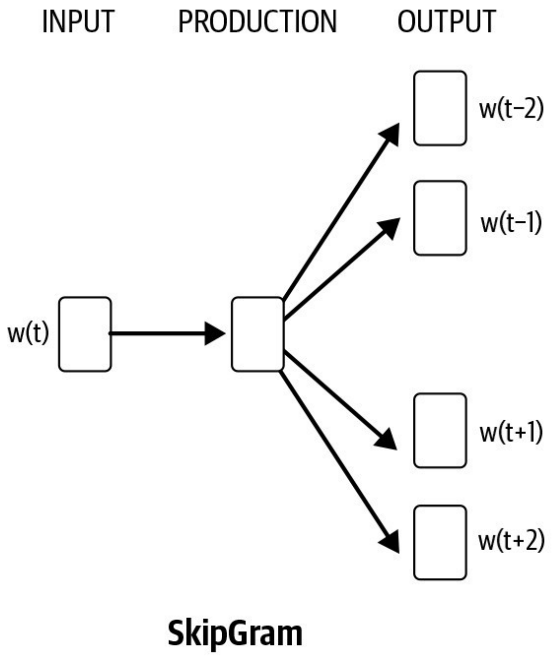
❖ Word2Vec: Skip-gram

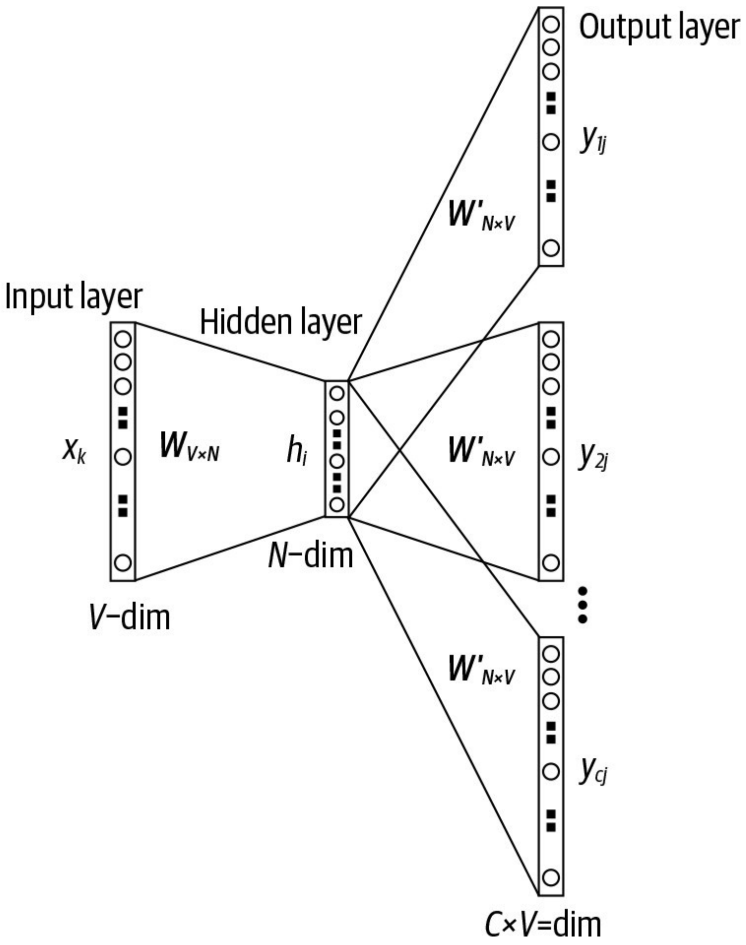
- Example: “Muốn lấy mật thì đừng phá tổ ong”



2 – Word2Vec

❖ Word2Vec: Skip-gram





2 – Word2Vec

❖ Word2Vec: Skip-gram

- w_i: word i from vocabulary V
- each word w_i is represented as two d-dimension vectors:

  - $v_i \in \mathbb{R}^d$: the central target word

  - $u_i \in \mathbb{R}^d$: the context words

2 – Word2Vec

❖ Word2Vec: Skip-gram

- w_c: central target word, indexed as c in the dictionary
- w_o: context word, indexed as o in the dictionary
- The conditional probability:

$$P(w_c|w_o) = \frac{\exp(u_o^T v_c)}{\sum_{i \in V} \exp(u_i^T v_c)}$$

2 – Word2Vec

Word2Vec: Skip-gram

- Optimization objective function:

$$\text{minimize } J = -\sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log P(w^{(t+j)} | w^{(t)})$$

$$-\log P(w_o | w_c) = -u_o^T v_c + \log \sum_{i=1}^{|V|} \exp(u_i^T v_c)$$

- Use SGD to update all relevant word vectors *v* and *u*
- **NOTE: Skip-gram** model use the central target word vector as the representation vector for a word

3 – CBOW Training

- 1. Load Dataset
- 2. Preprocessing
- 3. Build Data
- 4. Representation
- 5. CBOW Model
- 6. Predict

➤ Truyện Kiều - Nguyễn Du

```
[ ] !gdown --id 1Cm5iZJwcC-rrnUpSP1LWj2LMpytaT3gj

Downloading...
From: https://drive.google.com/uc?id=1Cm5iZJwcC-rrnUpSP1LWj2LMpytaT3gj
To: /content/truyen_kieu.txt
100% 140k/140k [00:00<00:00, 41.2MB/s]
```

```
[ ] import string as pystring
```

```
[ ] PUNCT_TO_REMOVE = pystring.punctuation + pystring.digits + "\n"
    def clean_text(text):
        """custom function to removal: punctuations and digits"""
        text = text.translate(str.maketrans(' ', ' ', PUNCT_TO_REMOVE))
        text = text.lower()
        return text
    clean_text(lines[0])

'trăm năm trong cõi người ta'
```

```
'9,,Rằng năm Gia Tĩnh triều Minh,\n',
'10.. Bốn phương phẳng lặng, hai kinh vững vàng.\n',
'11..Có nhà viên ngoại họ Vương,\n',
'12..Gia tư nghĩ cũng thường thường bực trung.\n',
'13..Một trai con thứ rốt lòng,\n',
'14..Vương Quan là chữ, nối dòng nho gia.\n',
```

3 – CBOW Training

1. Load Dataset

2. Preprocessing

3. Build Data

4. Representation

5. CBOW Model

6. Predict

- Removal punctuation, digits, tab,…
- Lowercasing

```
[ ] import string as pystring
```

```
[ ] PUNCT_TO_REMOVE = pystring.punctuation + pystring.digits + "\n"
    def clean_text(text):
        """custom function to removal: punctuations and digits"""
        text = text.translate(str.maketrans(' ', ' ', PUNCT_TO_REMOVE))
        text = text.lower()
        return text
    clean_text(lines[0])
```

```
'trăm năm trong cõi người ta'
```

3 – CBOW Training

1. Load Dataset

2. Preprocessing

3. Build Data

4. Representation

5. CBOW Model

6. Predict

- Get centers and contexts

```
[ ] def get_centers_and_contexts(corpus, max_window_size=2):

        centers, contexts = [], []

        for line in corpus:
            line = line.split()

            if len(line) <= 2*max_window_size:
                continue

            for i in range(max_window_size, len(line)-max_window_size):
                centers.append(line[i])
                idxs = list(range(i-max_window_size, i+max_window_size+1))
                idxs.remove(i)
                contexts.append(" ".join([line[idx] for idx in idxs]))

        return centers, contexts
```

```
[ ] centers, contexts = get_centers_and_contexts(corpus)
    len(centers), len(contexts)

    (9778, 9778)
```

```
[ ] centers[:2], contexts[:2]

    (['trong', 'cõi'], ['trăm năm cõi người', 'năm trong người ta'])
```

3 – CBOW Training

1. Load Dataset

2. Preprocessing

3. Build Data

4. Representation

5. CBOW Model

6. Predict

➢ Convert text into feature

```
[ ] max_length = 4
    embedding_size = 200
```

```
[ ] tokenizer = Tokenizer(oov_token='<OOV>')
    tokenizer.fit_on_texts(corpus)
```

```
[ ] vocab_size = len(tokenizer.index_word) + 1
```

```
[ ] train_seq = tokenizer.texts_to_sequences(contexts)
    train_seq_pad = pad_sequences(train_seq, maxlen=max_length, truncating='post', padding="post")
```

```
[ ] train_labels = [to_categorical( tokenizer.word_index[label], len(tokenizer.word_index) + 1) for label in centers]
```

```
[ ] train_labels = np.array(train_labels)
```

3 – CBOW Training

1. Load Dataset
2. Preprocessing
3. Build Data
4. Representation
5. CBOW Model
6. Predict

```
[ ]  cbow = Sequential()
     cbow.add(Embedding(input_dim=vocab_size, output_dim=embedding_size, input_length=4))
     cbow.add(Lambda(lambda x: K.mean(x, axis=1), output_shape=(embedding_size,)))
     cbow.add(Dense(vocab_size, activation='softmax'))
     cbow.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 4, 200)            482400

 lambda_4 (Lambda)           (None, 200)               0

 dense_4 (Dense)             (None, 2412)              484812

=================================================================
Total params: 967,212
Trainable params: 967,212
Non-trainable params: 0
_________________________________________________________________
```

```
[ ]  cbow.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
     cbow.fit(train_seq_pad, train_labels, epochs=30, verbose=1)
```

3 – CBOW Training

1. Load Dataset

2. Preprocessing

3. Build Data

4. Representation

5. CBOW Model

6. Predict

➢ Predict Center Word from Context

```
[ ]  sample_text = 'trăm năm cõi người'
     sample_seq = tokenizer.texts_to_sequences([sample_text])
     sample_seq_pad = pad_sequences(sample_seq, maxlen=max_length, truncating='post', padding="post")
     cbow.predict(sample_seq_pad)

     array([[5.1977589e-11, 5.5702581e-11, 9.7601995e-04, ..., 2.7690243e-08,
             5.7275875e-11, 1.2584069e-06]], dtype=float32)
```

```
[ ]  tokenizer.index_word[np.argmax(cbow.predict(sample_seq_pad))]

     'trong'
```