AI VIET NAM
@aivietnam.edu.vn

# NLP Basic – 08
# Pre-trained Word Vectors

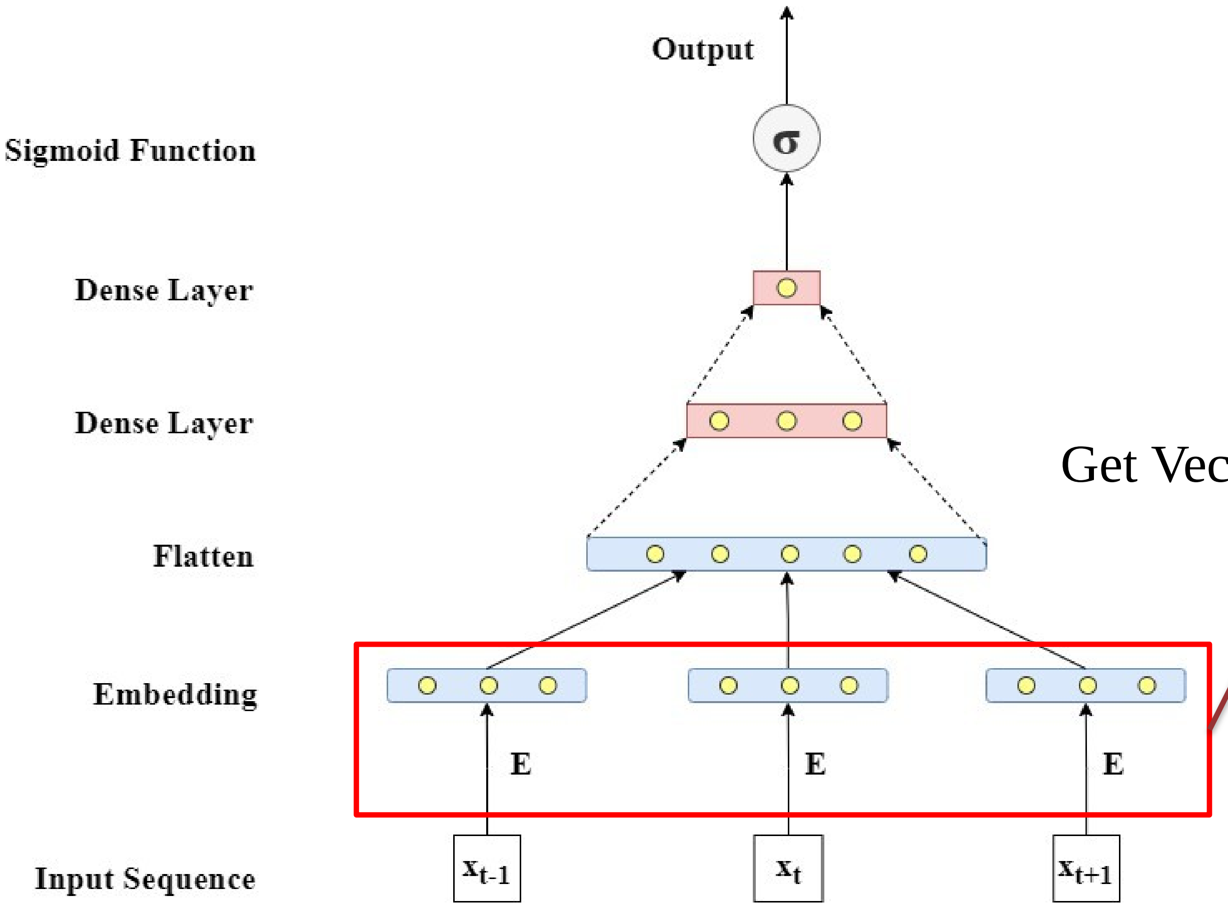
AI VIET NAM
Nguyễn Quốc Thái

AI VIET NAM
@aivietnam.edu.vn

# CONTENT

# 1 – Pretrained Word Vectors

## ❖ Pre-trained Language Models (LMs)




Get Vectors

Not Enough

Embedding Matrix

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *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 |

# 1 – Pretrained Word Vectors

## ❖ Pre-trained Language Models (LMs)

Objective: Language Model

Supervised: Text Classification

Model

**LANGUAGE MODEL**

Data

0 1

**Classifier**

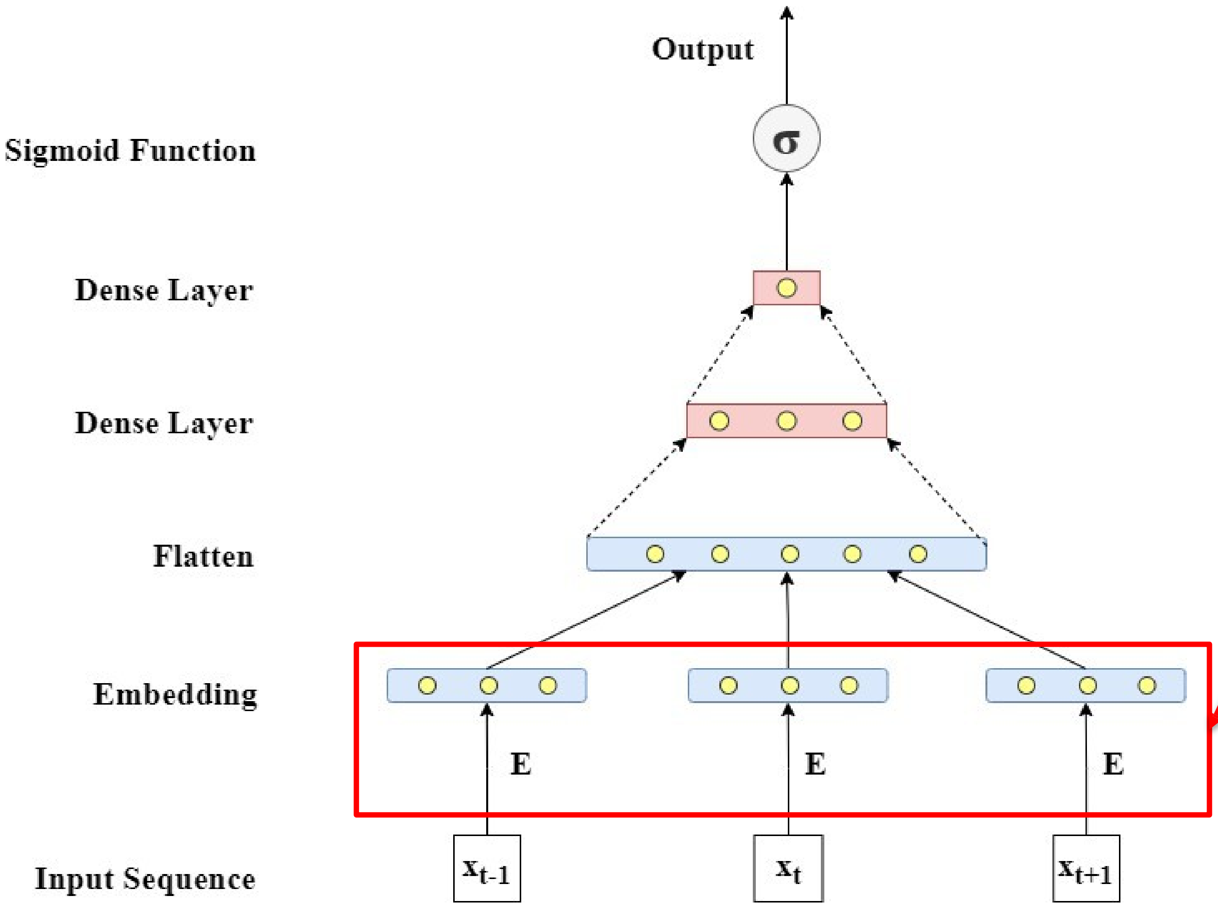
**PRE-TRAINED LMs**

| | |
|---|---|
| This movie is bad | 0 |
| This movie is good | 1 |

AI VIET NAM
@aivietnam.edu.vn

# 1 – Pretrained Word Vectors

## Pre-trained Language Models (LMs)

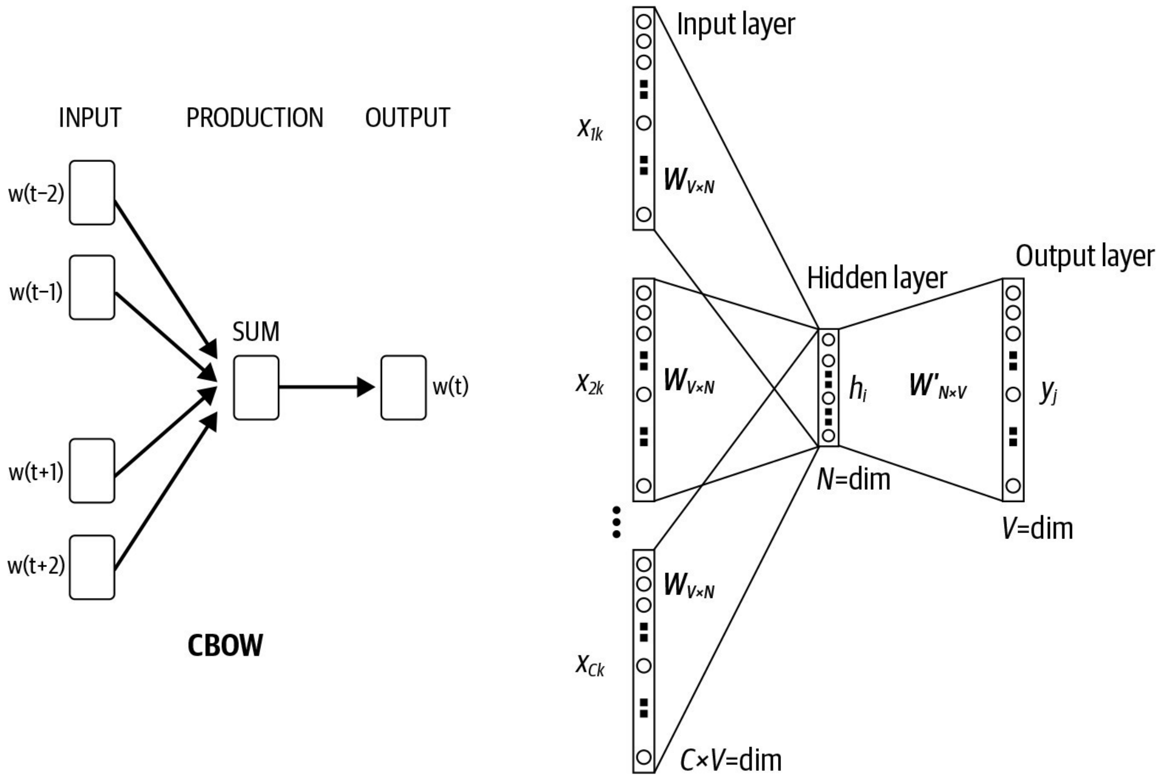


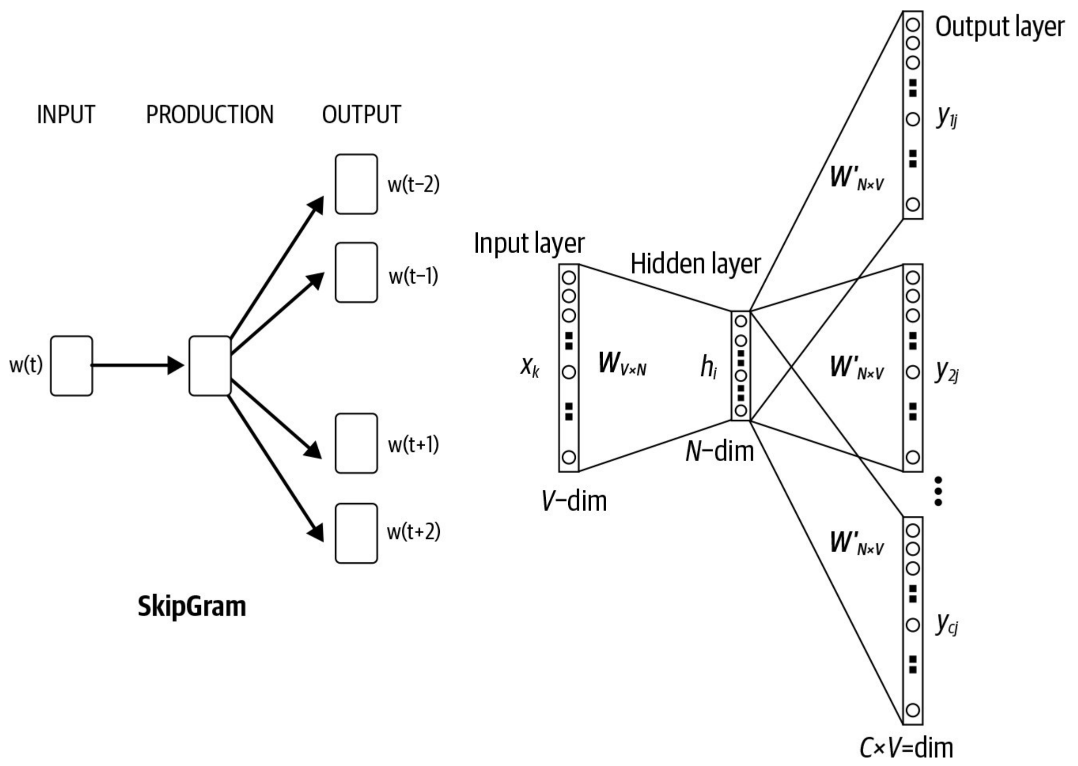

- [Word2Vec]
- [Glove]
- [Fasttext]
- ELMO

**Training Data**
Wikipedia
News
Book
Social Network
…

# 1 – Pretrained Word Vectors

## ❖ Word2Vec

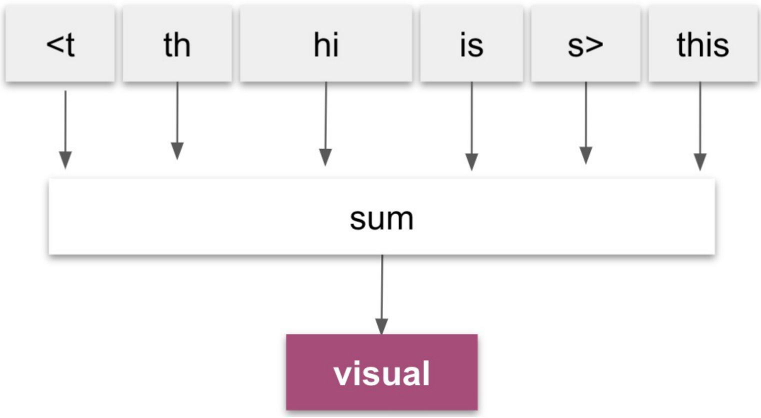


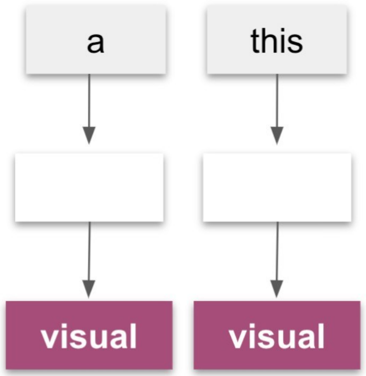
# 1 – Pretrained Word Vectors

## ❖ Fasttext

fastText




Word2Vec



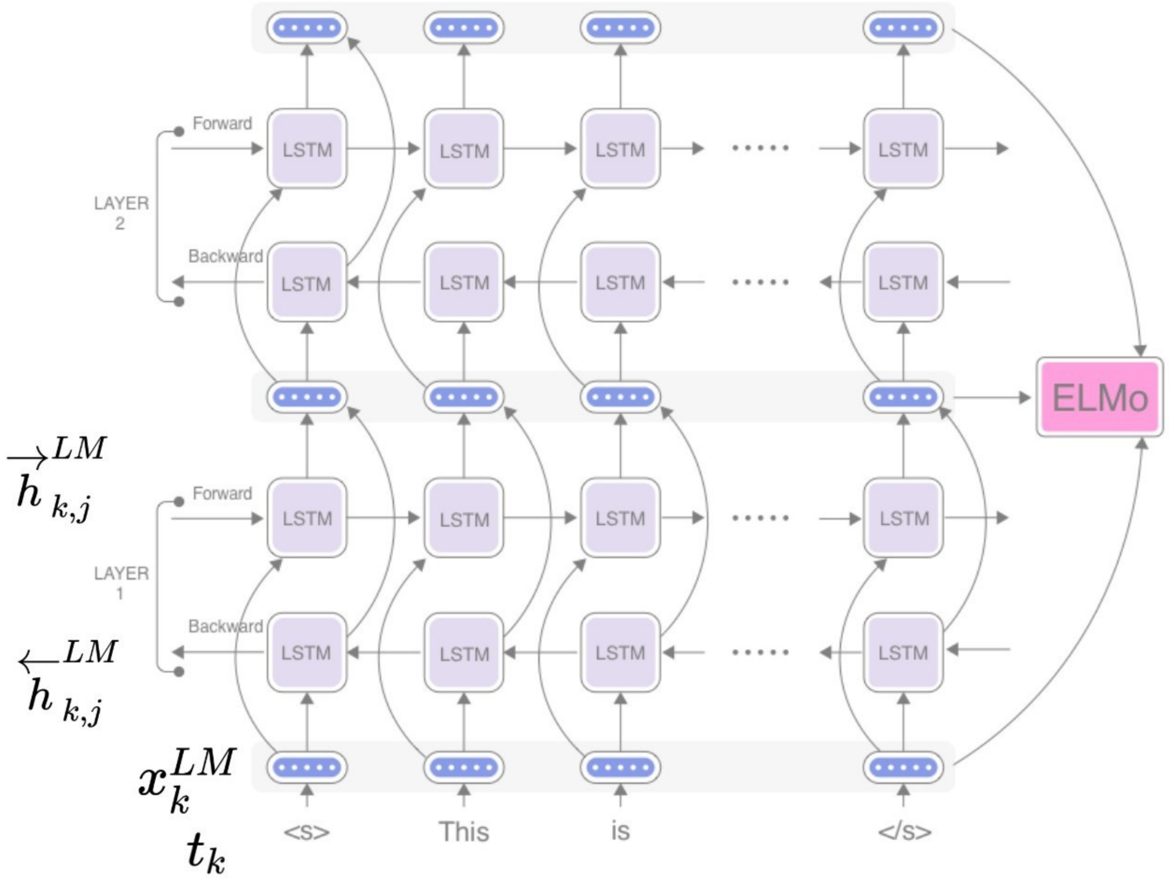

Perplexity

| | Cs | De | Es | Fr | Ru |
|---|---|---|---|---|---|
| Vocab. size | 46k | 37k | 27k | 25k | 63k |
| CLBL | 465 | 296 | 200 | 225 | 304 |
| CANLM | 371 | 239 | 165 | 184 | 261 |
| LSTM | 366 | 222 | 157 | 173 | 262 |
| sg | 339 | 216 | 150 | 162 | 237 |
| sisg | **312** | **206** | **145** | **159** | **206** |

AI VIET NAM
@aivietnam.edu.vn

# 1 – Pretrained Word Vectors

## ❖ ELMO




[Paper](#)
[Pre-trained](#)

| TASK | PREVIOUS SOTA | | OUR BASELINE | ELMO + BASELINE |
|---|---|---|---|---|
| SQuAD | Liu et al. (2017) | 84.4 | 81.1 | 85.8 |
| SNLI | Chen et al. (2017) | 88.6 | 88.0 | 88.7 ± 0.17 |
| SRL | He et al. (2017) | 81.7 | 81.4 | 84.6 |
| Coref | Lee et al. (2017) | 67.2 | 67.2 | 70.4 |
| NER | Peters et al. (2017) | 91.93 ± 0.19 | 90.15 | 92.22 ± 0.10 |
| SST-5 | McCann et al. (2017) | 53.7 | 51.4 | 54.7 ± 0.5 |

# 1 – Pretrained Word Vectors

## Glove

- The conditional probability:

$$Q_{ij} = \frac{\exp(u_j^T v_i)}{\sum_{w\in V} \exp(u_w^T v_i)}$$

- Using co-occurrence probabilities:

$$J = -\sum_{i}\sum_{j\in \text{context}(i)} X_{ij} \log Q_{ij}$$

NER Task

| Model | Dev | Test | ACE | MUC7 |
|---|---|---|---|---|
| Discrete | 91.0 | 85.4 | 77.4 | 73.4 |
| SVD | 90.8 | 85.7 | 77.3 | 73.7 |
| SVD-S | 91.0 | 85.5 | 77.6 | 74.3 |
| SVD-L | 90.5 | 84.8 | 73.6 | 71.5 |
| HPCA | 92.6 | **88.7** | 81.7 | 80.7 |
| HSMN | 90.5 | 85.7 | 78.7 | 74.7 |
| CW | 92.2 | 87.4 | 81.7 | 80.2 |
| CBOW | 93.1 | 88.2 | 82.2 | 81.1 |
| GloVe | **93.2** | 88.3 | **82.9** | **82.2** |

# 1 – Pretrained Word Vectors

## ❖ Pre-trained Glove Embedding

### Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at: http://www.opendatacommons.org/licenses/pddl/1.0/.
  - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip
- Ruby script for preprocessing Twitter data

# 1 – Pretrained Word Vectors

## ❖ Pre-trained Glove Embedding

Version:
6B
400K Vocab
50 D

```
embeddings_dict = {}
with open("/content/glove.6B.50d.txt", 'r') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], "float32")
        embeddings_dict[word] = vector
def embedding(word):
    return embeddings_dict.get(word, embeddings_dict.get('unk'))
```

# 1 – Pretrained Word Vectors

## ❖ Pre-trained Glove Embedding

Version:
6B
400K Vocab
50 D

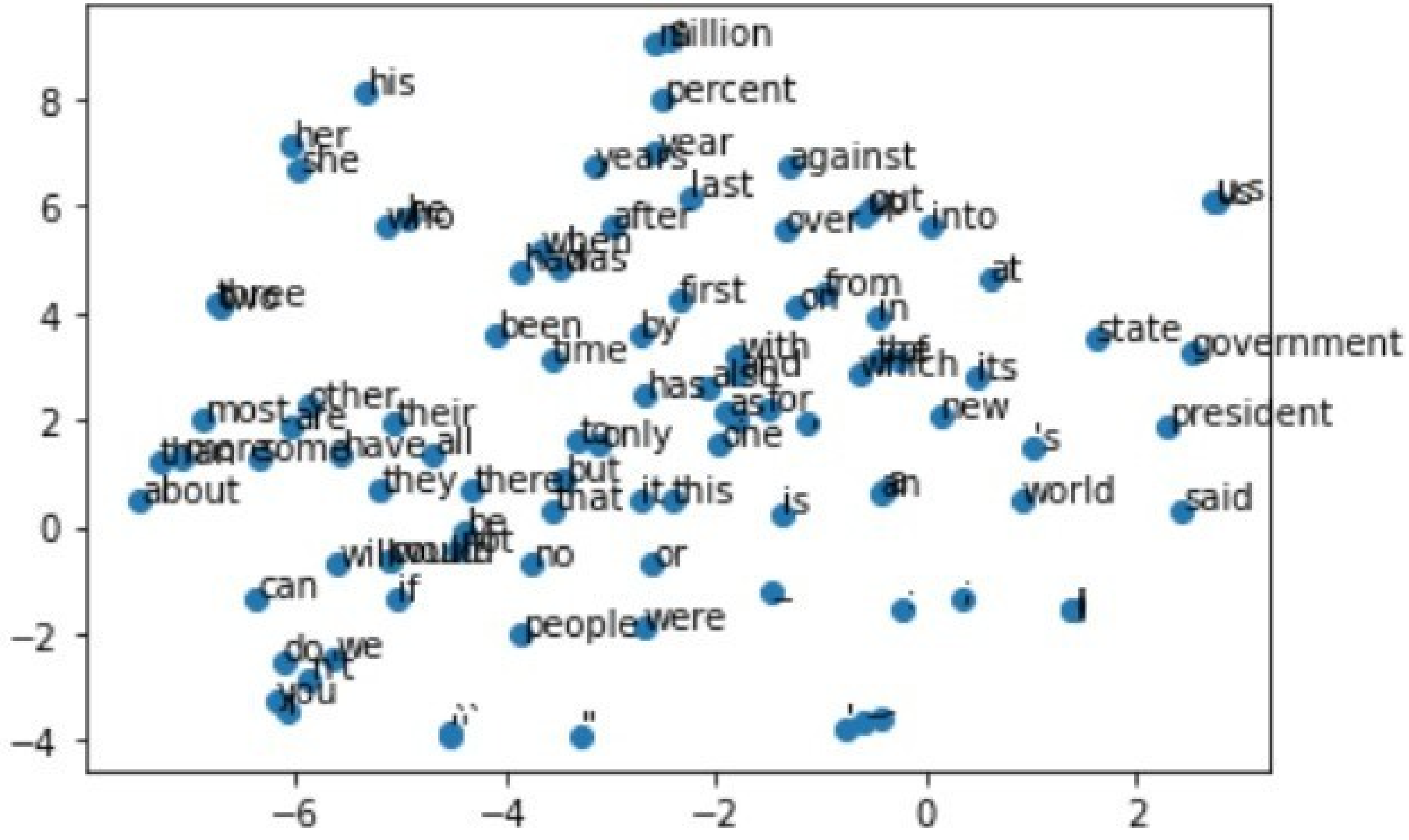
```
embedding('man')
```

```
array([-0.094386,  0.43007 , -0.17224 , -0.45529 ,  1.6447   ,  0.40335 ,
       -0.37263 ,  0.25071 , -0.10588 ,  0.10778 , -0.10848 ,  0.15181 ,
       -0.65396 ,  0.55054 ,  0.59591 , -0.46278 ,  0.11847 ,  0.64448 ,
       -0.70948 ,  0.23947 , -0.82905 ,  1.272   ,  0.033021,  0.2935  ,
        0.3911  , -2.8094  , -0.70745 ,  0.4106  ,  0.3894  , -0.2913  ,
        2.6124  , -0.34576 , -0.16832 ,  0.25154 ,  0.31216 ,  0.31639 ,
        0.12539 , -0.012646,  0.22297 , -0.56585 , -0.086264,  0.62549 ,
       -0.0576  ,  0.29375 ,  0.66005 , -0.53115 , -0.48233 , -0.97925 ,
        0.53135 , -0.11725 ], dtype=float32)
```

# 1 – Pretrained Word Vectors

## ❖ Pre-trained Glove Embedding

Version:
6B
400K Vocab
50 D

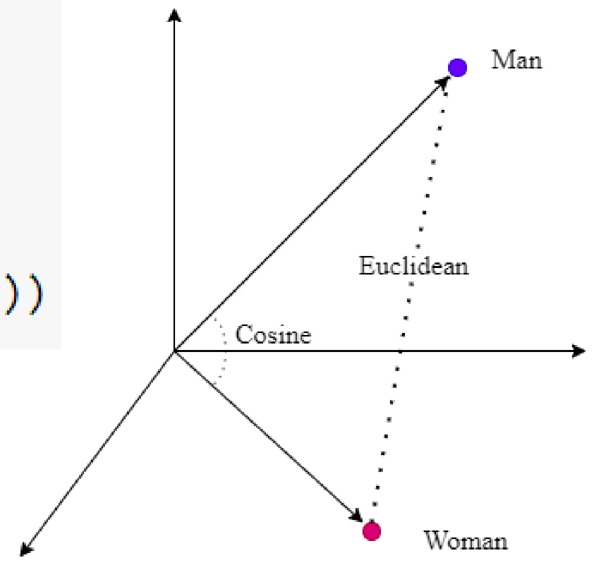


# 1 – Pretrained Word Vectors

## ❖ Pre-trained Glove Embedding

Find Synonyms
Find Analogies

```
def euclidean_distance(x, y):
    return np.sqrt(np.sum((x - y) ** 2))

def cosine_similarity(x, y):
    return np.dot(x, y) / (np.sqrt(np.dot(x, x)) * np.sqrt(np.dot(y, y)))
```



# 1 – Pretrained Word Vectors

## ❖ Pre-trained Glove Embedding

Find Synonyms

Given a word

Find top k synonym words

```
def find_closest_embeddings(target_embedding, top_word, score="cosine"):
    score_dict = {}
    if score == "euclidean":
        for word in embeddings_dict.keys():
            score_dict[word] = euclidean_distance(embedding(word), target_embedding)
        score_dict = sorted(score_dict.items(), key=lambda kv: kv[1])
    if score == "cosine":
        for word in embeddings_dict.keys():
            score_dict[word]= cosine_similarity(embedding(word), target_embedding)
        score_dict = sorted(score_dict.items(), key=lambda kv: kv[1], reverse=True)

    return score_dict[:top_word]
```

# 1 – Pretrained Word Vectors

## ❖ Pre-trained Glove Embedding

Find Synonyms

Given a word

Find top k synonym words

```
#Use euclidean distance
find_closest_embeddings(embedding("man"), 5, score="euclidean")
```

```
[('man', 0.0),
 ('woman', 2.6026237),
 ('another', 2.8089325),
 ('boy', 2.8093922),
 ('one', 2.9732437)]
```

```
#Use cosine similarity
find_closest_embeddings(embedding("man"), 5)
```

```
[('man', 1.0),
 ('woman', 0.8860338),
 ('boy', 0.8564431),
 ('another', 0.84528404),
 ('old', 0.8372183)]
```

# 1 – Pretrained Word Vectors

## ❖ Pre-trained Glove Embedding

Find Analogies

Given 3 words

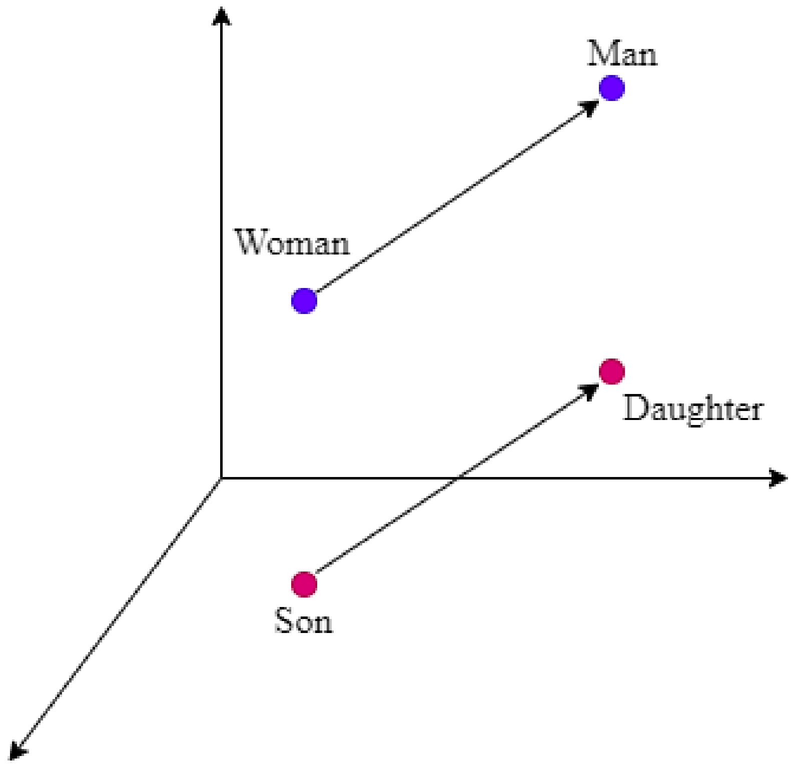
Find a word with analogies relationship

Example:

"man" : "woman" :: "son" : "daughter"

a : b :: c : d

Vec(a) + Vec(d) = Vec(b) + Vec(c)



AI VIET NAM
@aivietnam.edu.vn

# 1 – Pretrained Word Vectors

## ❖ Pre-trained Glove Embedding

Find Analogies

Given 3 words

Find a word with analogies relationship

Example:

“man” : “woman” :: “son” : “daughter”

a : b :: c : d

Vec(a) + Vec(d) = Vec(b) + Vec(c)

```
def find_analogy(word_a, word_b, word_c, score="cosine"):
    x = embedding(word_c) + embedding(word_b) - embedding(word_a)
    return find_closest_embeddings(x, 5, score=score)
```

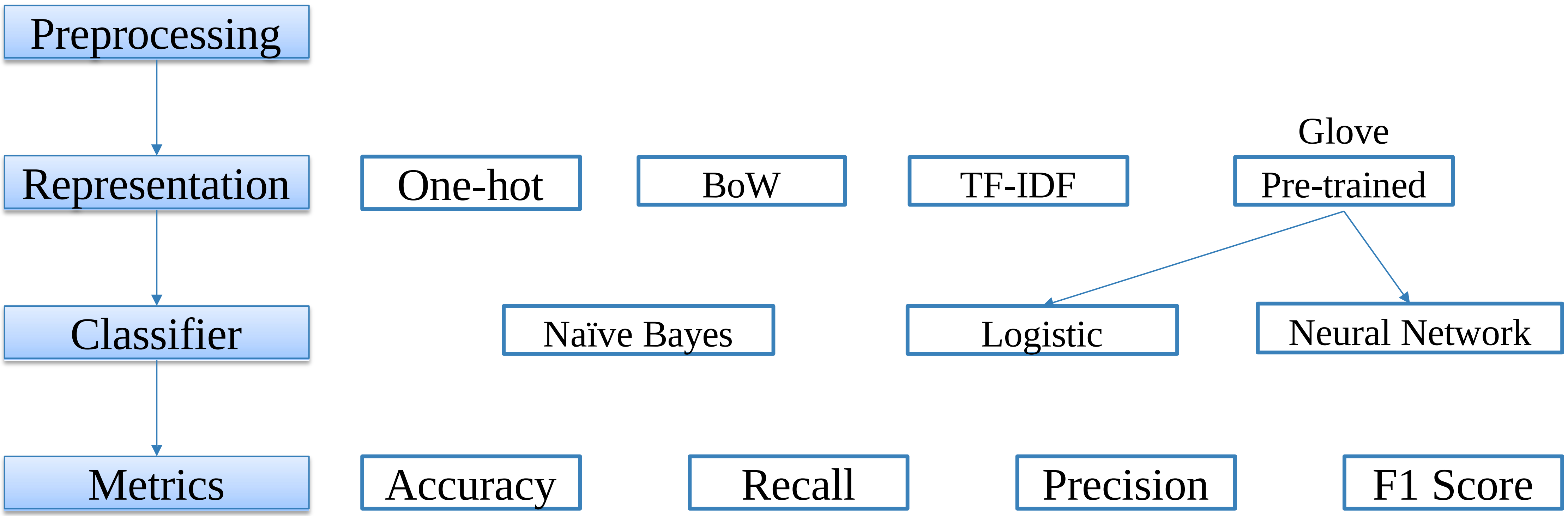
```
find_analogy("man", "woman", "son", score="euclidean")
```

```
[('daughter', 1.5812494),
 ('mother', 2.448467),
 ('wife', 2.4532523),
 ('son', 2.6026237),
 ('father', 2.7388973)]
```

```
find_analogy("man", "woman", "son")
```

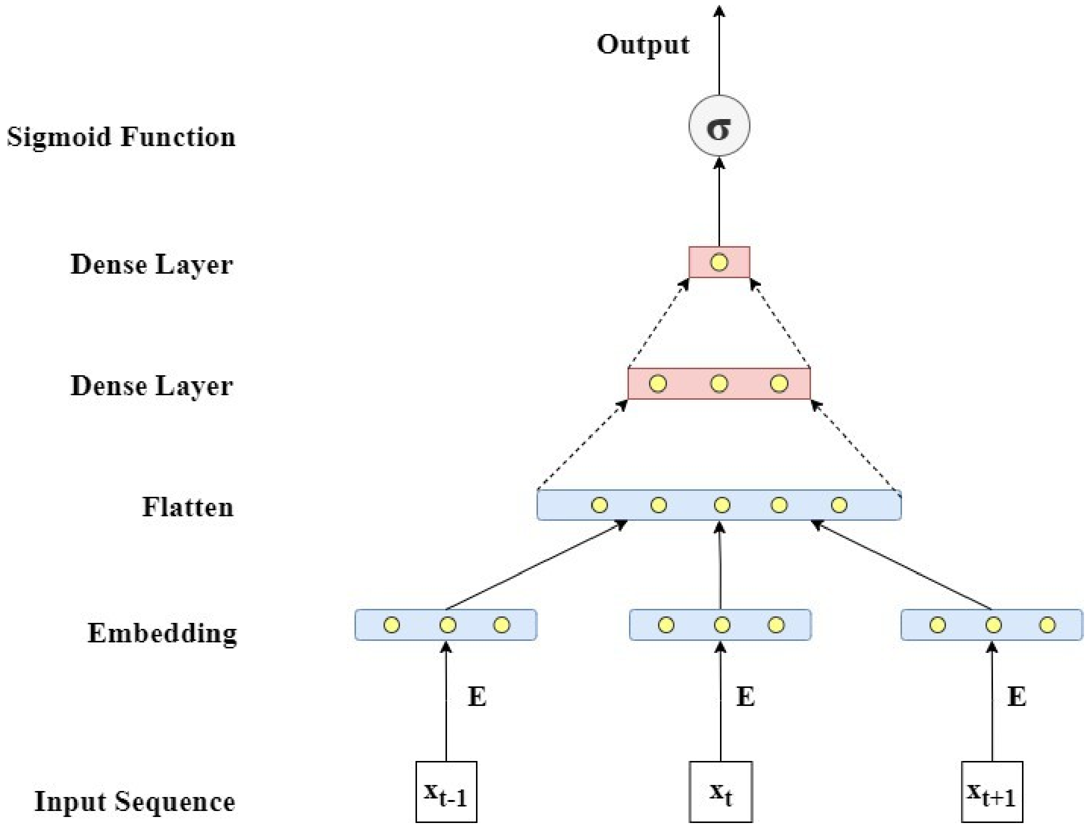
```
[('daughter', 0.9658342),
 ('mother', 0.91536117),
 ('wife', 0.9149919),
 ('son', 0.903869),
 ('niece', 0.8937417)]
```

# 2 – Text Classification



# 2 – Text Classification

## Neural Network




```python
model_nn = Sequential()
model_nn.add(Embedding(vocab_size, embedding_dim, input_length=max_length))
model_nn.add(Flatten())
model_nn.add(Dense(10, activation='relu'))
model_nn.add(Dense(1, activation='sigmoid'))
model_nn.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
model_nn.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 100, 200)          3000000
_________________________________________________________________
flatten_2 (Flatten)          (None, 20000)             0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                200010
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 11
=================================================================
Total params: 3,200,021
Trainable params: 3,200,021
Non-trainable params: 0
```

# 2 – Text Classification

## Review: 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

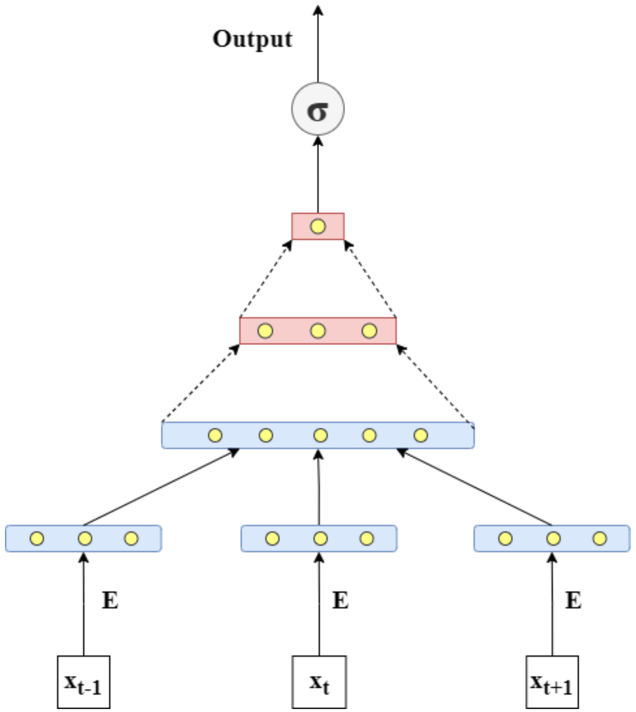
| | | | | |
|---|---|---|---|---|
| | 3.1 | 1.4 | 0.6 | 0.1 | 2.5 |
| 0.1 | 0.7 | 1.3 | 0.4 | 0.5 | |
| 2.3 | 2.5 | 0.5 | 1.3 | 0.4 | |

Input shape: 2x5x2

AI VIET NAM
@aivietnam.edu.vn

# 2 – Text Classification

## ❖ Pre-trained Glove Embedding




Embedding matrix

| | | | |
|---|---|---|---|
| 0 | <oov> | 0.1 | 3.1 |
| 1 | <pad> | 0.5 | 2.5 |
| 2 | <unk> | 1.3 | 0.6 |
| 3 | neural | 0.4 | 0.1 |
| 4 | language | 0.7 | 1.4 |

Glove embedding matrix

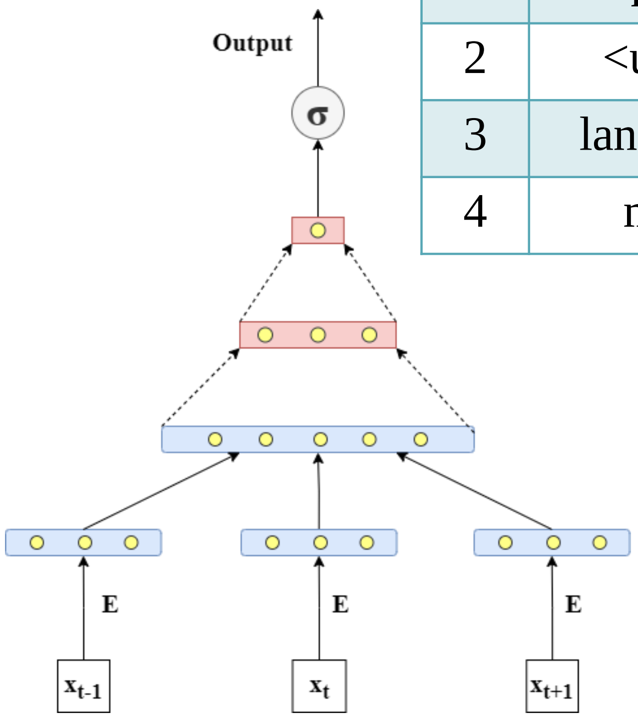
| | | | |
|---|---|---|---|
| 0 | <oov> | 0.1 | 0.1 |
| 1 | <pad> | 0.5 | 0.5 |
| 2 | <unk> | 0.3 | 0.6 |
| 3 | language | 0.7 | 0.7 |
| 4 | mưa | 0.7 | 0.4 |

Final embedding matrix

| | | | |
|---|---|---|---|
| 0 | <oov> | 0.1 | 0.1 |
| 1 | <pad> | 0.5 | 0.5 |
| 2 | <unk> | 0.3 | 0.6 |
| 3 | neural | 0.0 | 0.0 |
| 4 | language | 0.7 | 0.7 |

# 2 – Text Classification

## ❖ Pre-trained Glove Embedding




Glove embedding matrix

| | | | |
|---|---|---|---|
| 0 | <oov> | 0.1 | 0.1 |
| 1 | <pad> | 0.5 | 0.5 |
| 2 | <unk> | 0.3 | 0.6 |
| 3 | language | 0.7 | 0.7 |
| 4 | mưa | 0.7 | 0.4 |

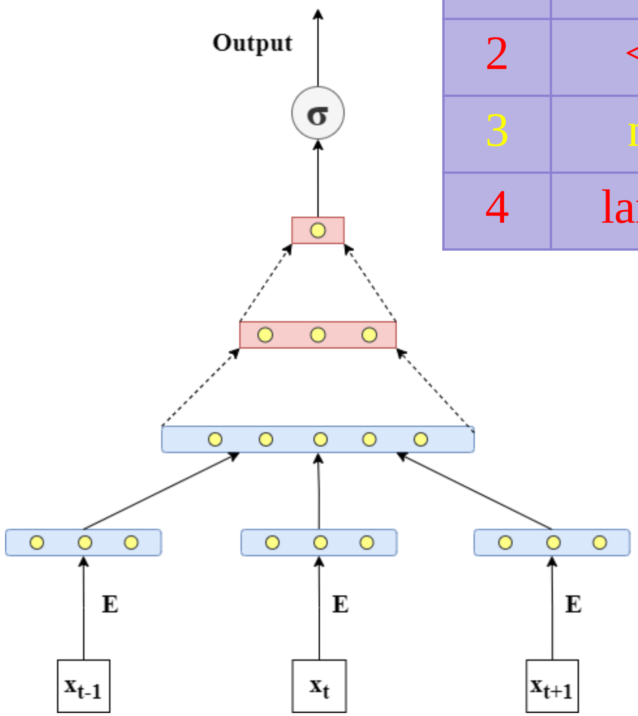
```
count = 0
embedding_matrix = np.zeros((vocab_size, embedding_dim))

for word, i in tokenizer.word_index.items():
    if i < vocab_size:
        embedding_vector = embeddings_dict.get(word)

        # Words not found in the mebedding index will all l
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

# 2 – Text Classification

## ❖ Pre-trained Glove Embedding

Final embedding matrix

| | | | |
|---|---|---|---|
| 0 | <oov> | 0.1 | 0.1 |
| 1 | <pad> | 0.5 | 0.5 |
| 2 | <unk> | 0.3 | 0.6 |
| 3 | neural | 0.0 | 0.0 |
| 4 | language | 0.7 | 0.7 |

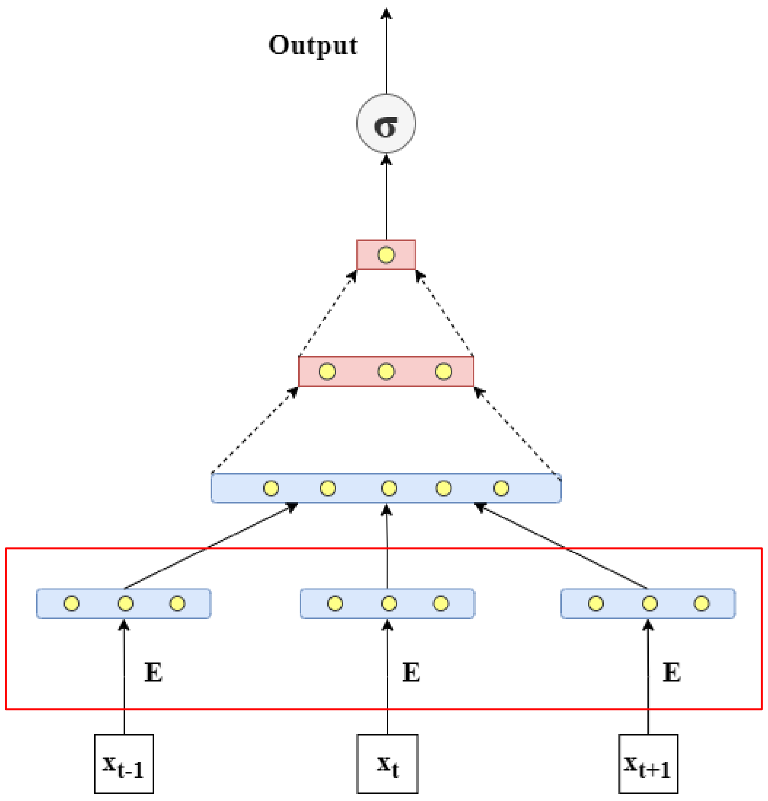



```python
vocab_size = 15000
max_length = 100
embedding_dim = 200
```

```python
embeddings_dict = {}
with open("/content/glove.6B.200d.txt", 'r') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], "float32")
        embeddings_dict[word] = vector
def embedding(word):
    return embeddings_dict.get(word, embeddings_dict.get('unk'))
```

# 2 – Text Classification

## ❖ Pre-trained Glove Embedding




Final embedding matrix

| | | | |
|---|---|---|---|
| 0 | <oov> | 0.1 | 0.1 |
| 1 | <pad> | 0.5 | 0.5 |
| 2 | <unk> | 0.3 | 0.6 |
| 3 | neural | 0.0 | 0.0 |
| 4 | language | 0.7 | 0.7 |

~~Update weight~~

Update weight

```
model_nn_pre.layers[0].set_weights([embedding_matrix])
model_nn_pre.layers[0].trainable = False
```

```
model_nn_pre.layers[0].set_weights([embedding_matrix])
model_nn_pre.layers[0].trainable = True
```

AI VIET NAM
@aivietnam.edu.vn

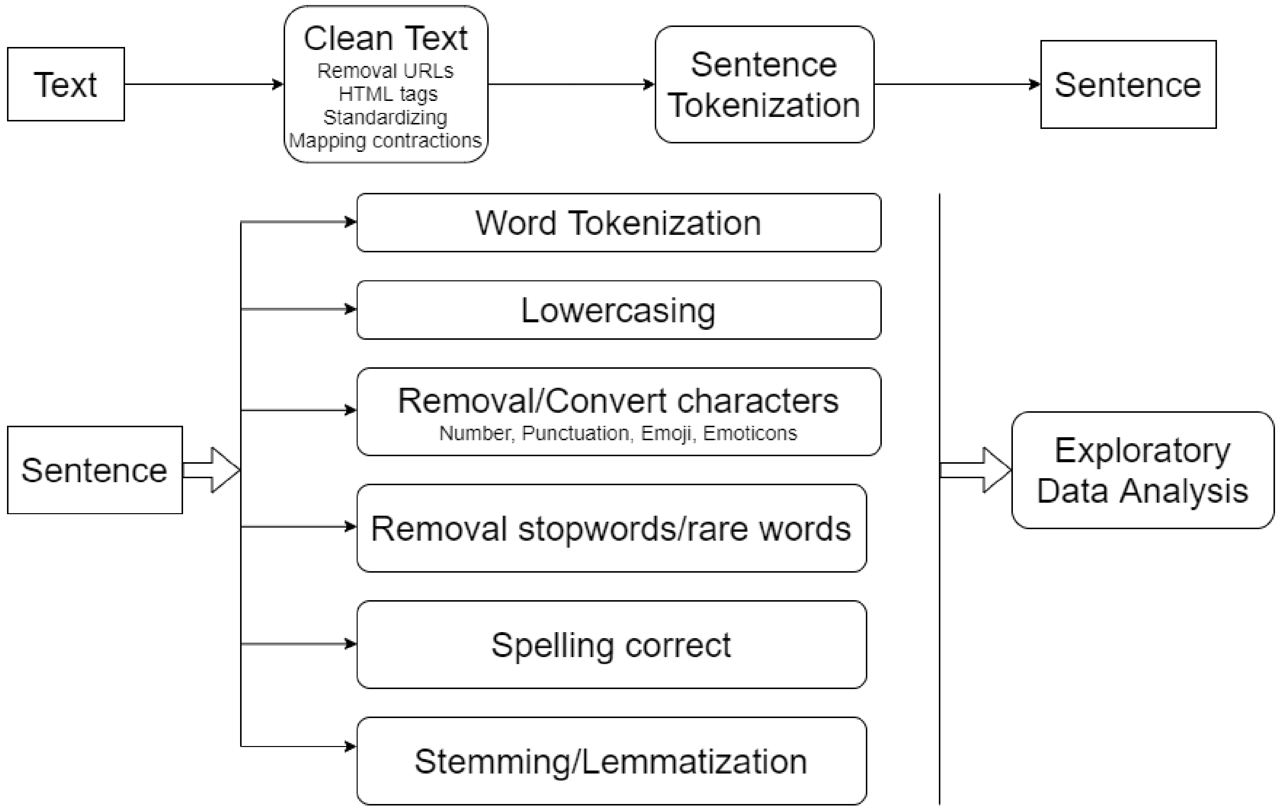
# 3 - Summary

## ❖ Basic NLP Course

01 Introduction
02 Preprocessing
03 Language Modeling
04 Part Of Speech (POS)
05 Constituency Parsing
06 Basic Vectorization
07 Word2Vec
08 Pretrained Model

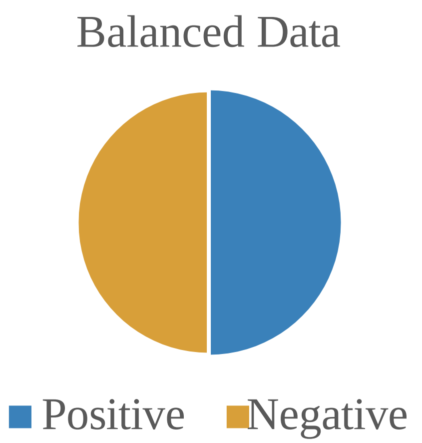
# 3 - Summary

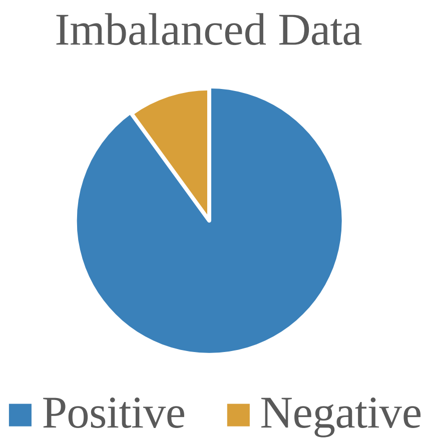
## 2 - Preprocessing

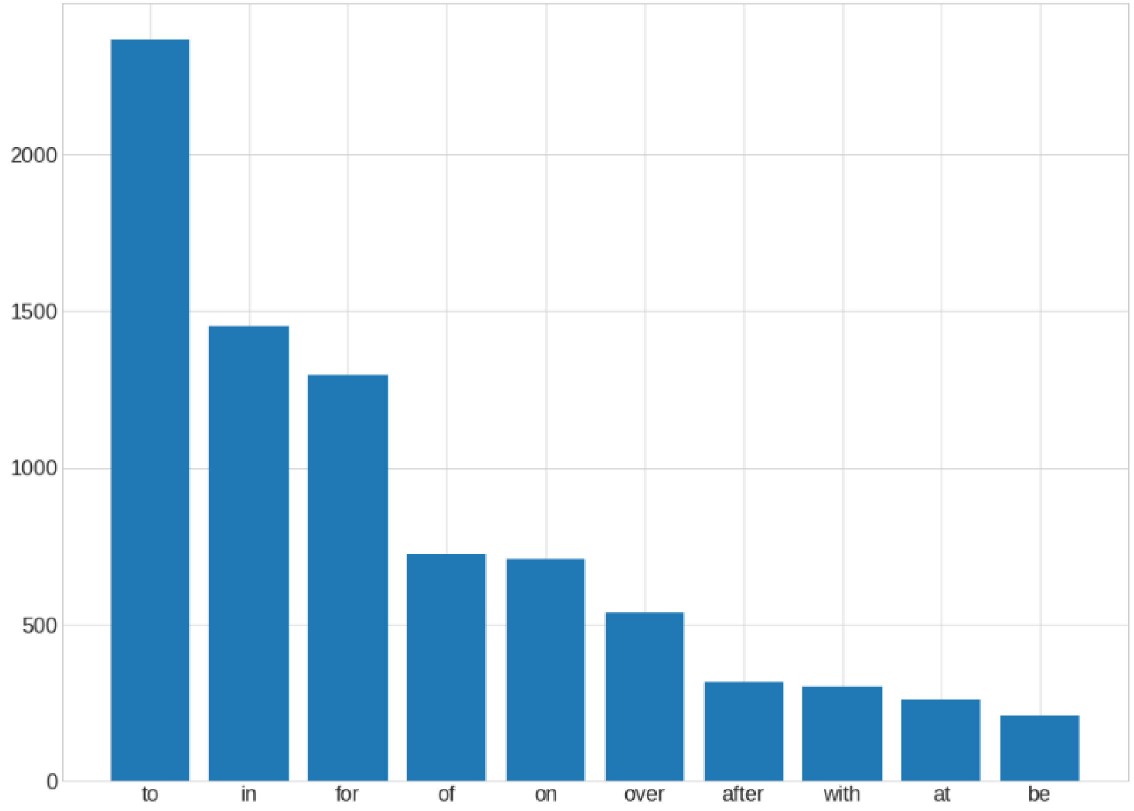


# 3 - Summary

## ❖ 2 - Preprocessing



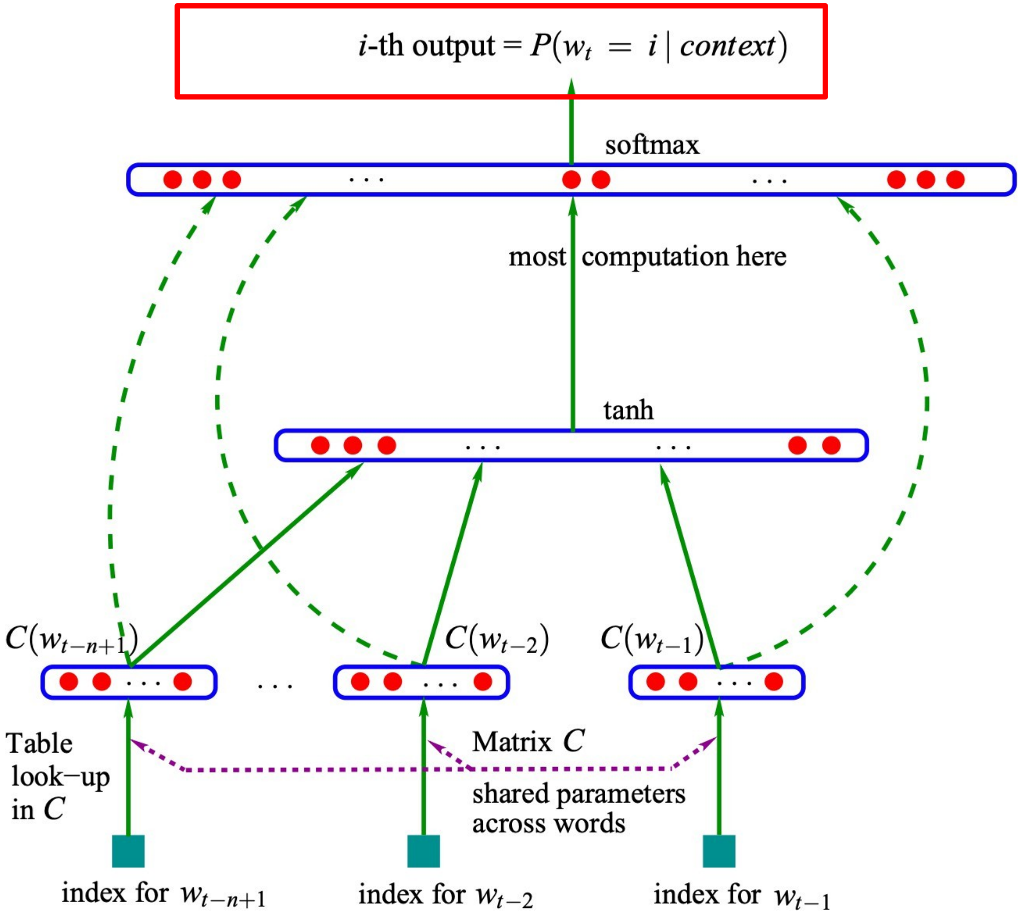



AI VIET NAM
@aivietnam.edu.vn

# 3 - Summary

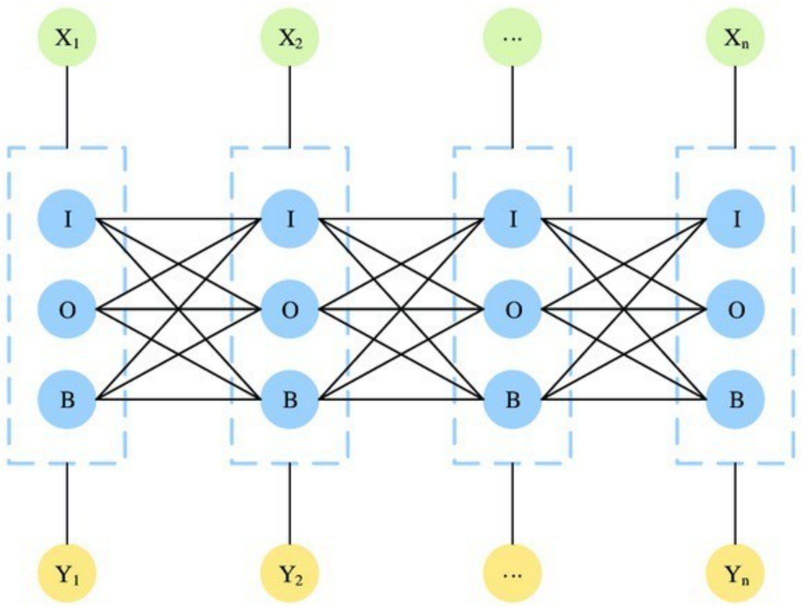
## ❖ 3 – Language Model



# 3 - Summary

## ❖ 4 – POS Tagging - NER

Model: Hidden Markov Model (HMM)





# 3 - Summary

## 5 – Constituency Parsing (CFG)

**G = (T, N, P, S, R)**

T: a set of terminal symbols

N: a set of non-terminal symbols

P(P N ): a set of pre-terminal symbols

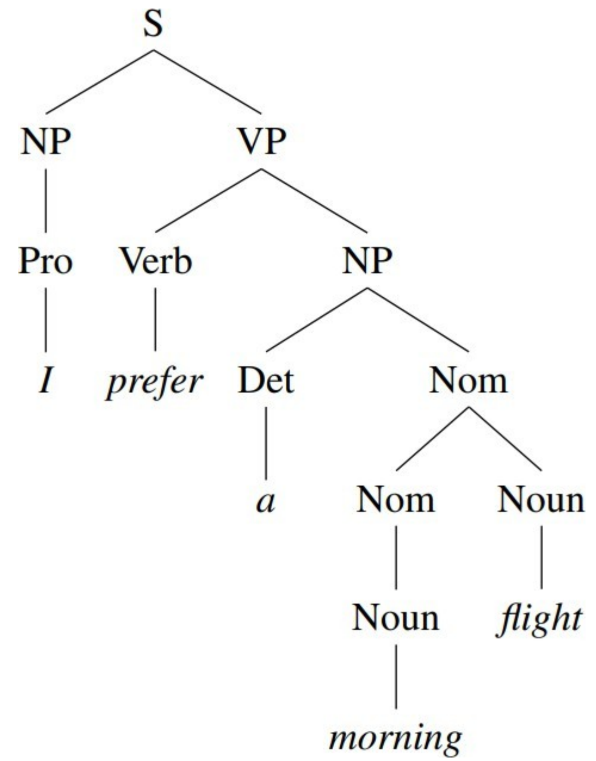
S: a start symbol

R: a set of rules or productions
R = { | N, (T N)}

**Grammar in CNF**

1. Start S
2. S NP VP
3. NP Det Noun
4. NP NN PP
5. PP Prep NP
6. VP V NP
7. a. VP V Args
   b. Args NP PP
8. V ate
9. NP John
10. NP ice-cream, snow
11. Noun ice-cream, pizza
12. Noun table, guy, campus
13. Det the
14. Prep on

CKY



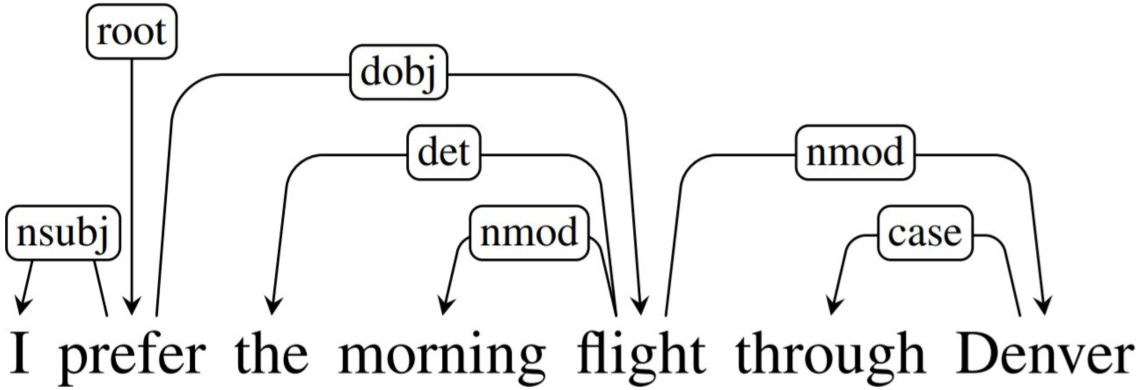
# 3 - Summary

## 5 – Dependency Parsing

**A graph G = (V, A)**

**V vertices**
**{$w_0$=root, $w_1$,…, $w_n$)**
usually one per word in sentence

**A arcs**
**{($w_i$,r,$w_j$):$w_i$≠$w_j$) | $w_i$∈V, $w_j$∈V-$w_0$, r ∈$R_x$}**
$R_x$: a set of all possible dependency relations in x



# 3 - Summary

## ❖ 5 – Dependency Parsing

**Dependency Tree**

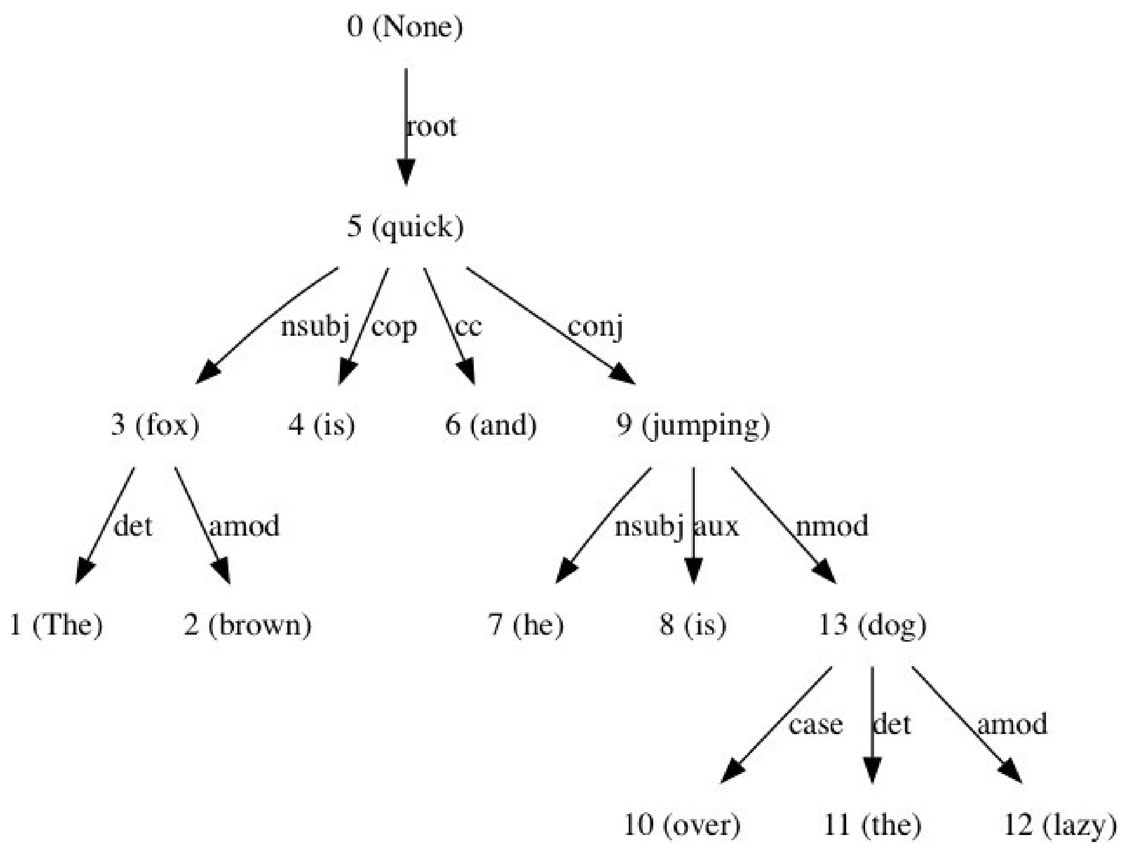
**a ROOT**

Each word has a **single head**

Dependency structure is **connected**

**Projective**

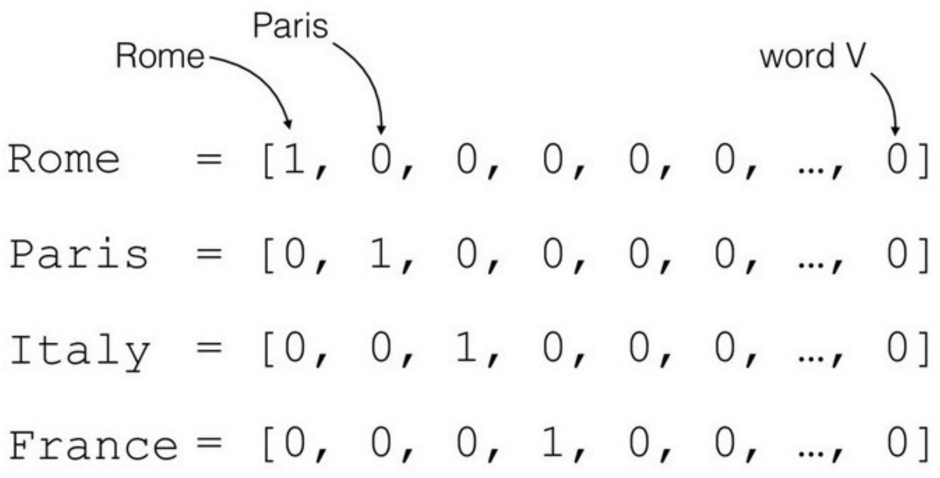
**Acyclic**
Unique path from ROOT to each word

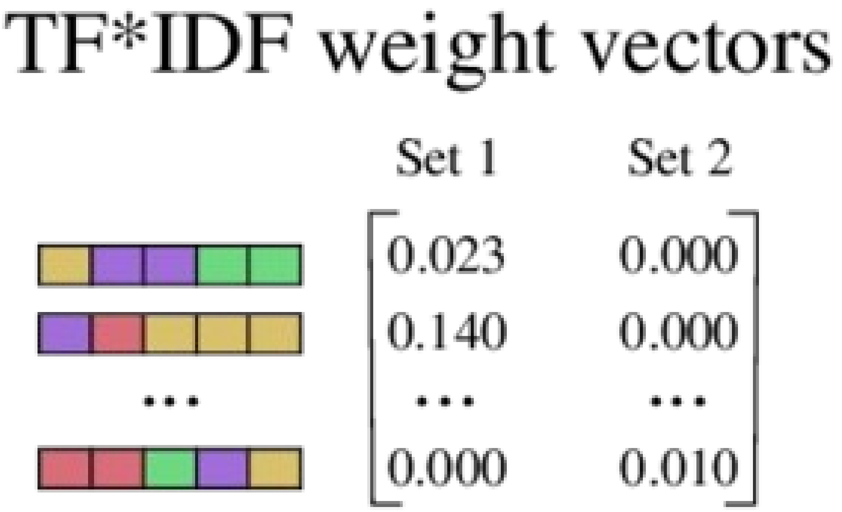


# 3 - Summary

## ❖ 6 – Basic Vectorization

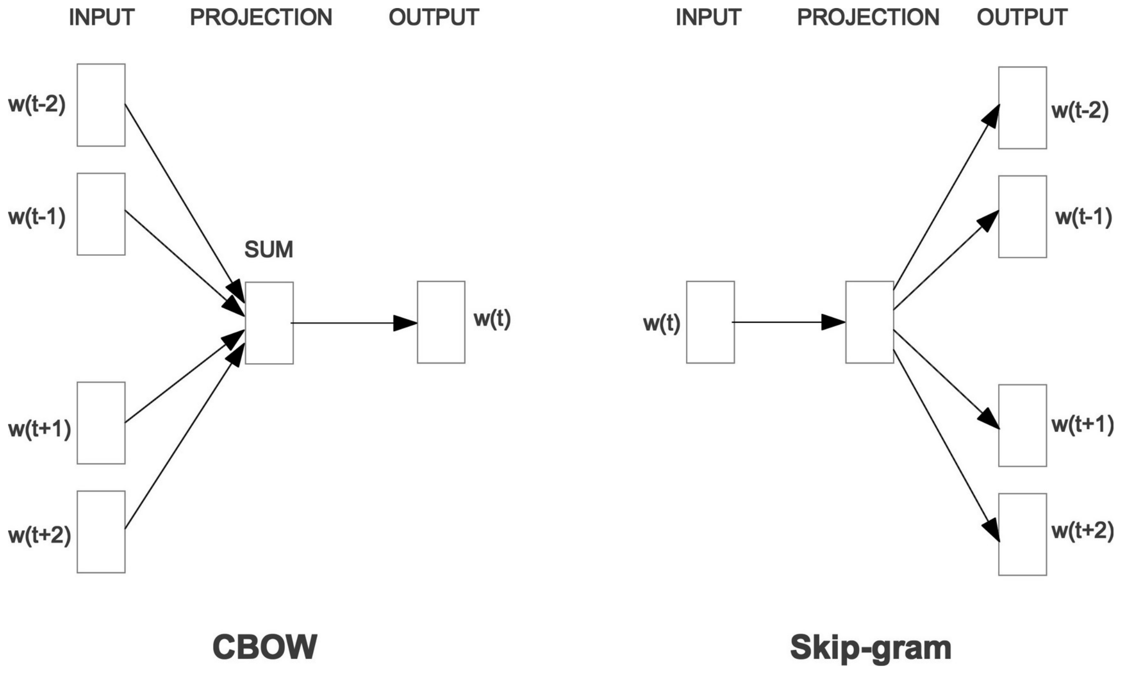
- ❑ One-hot encoding
- ❑ Bag-of-words (BoW)
- ❑ Bag-of-N-gram



# 3 - Summary

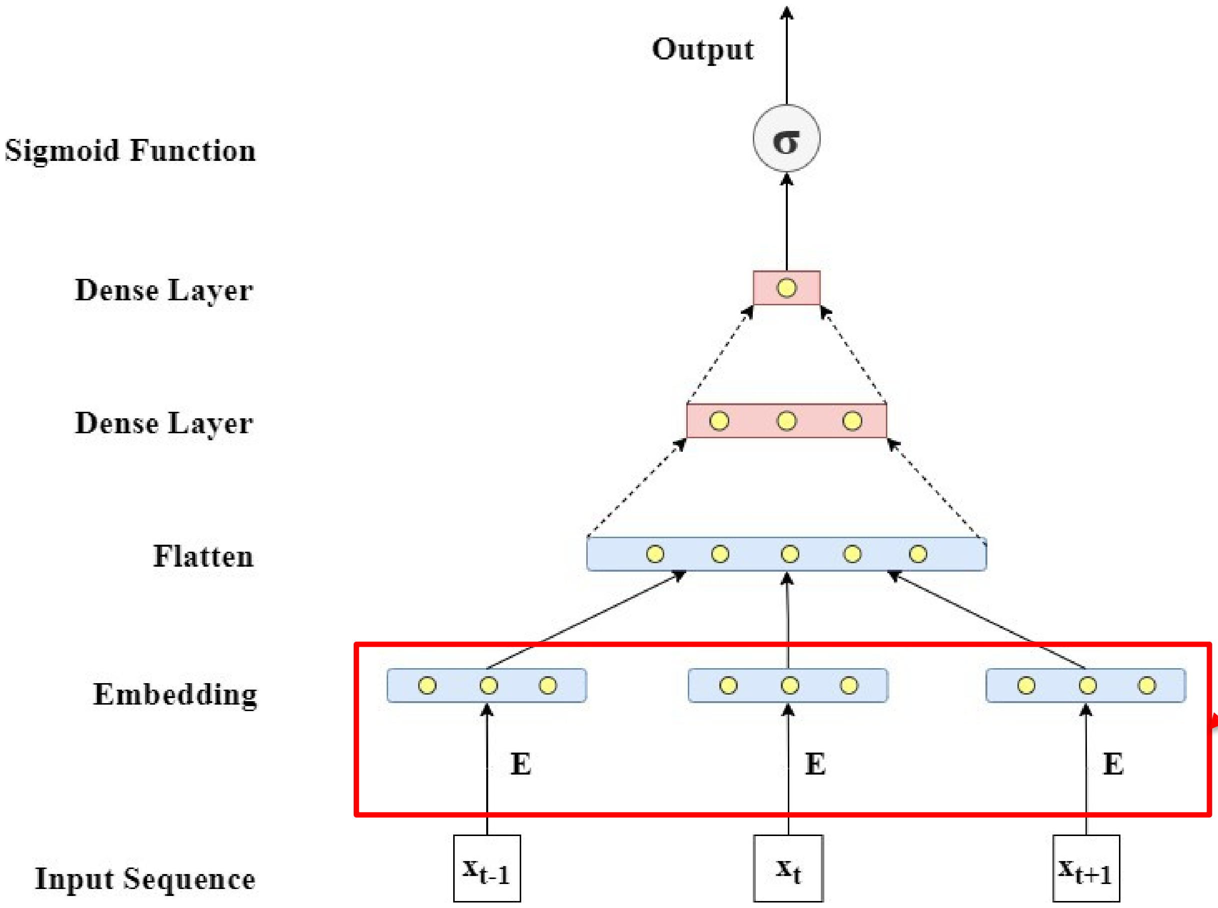
## ❖ 7 – Word2Vec




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *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 |

# 3 - Summary

## ❖ 7 – Pre-trained Embedding

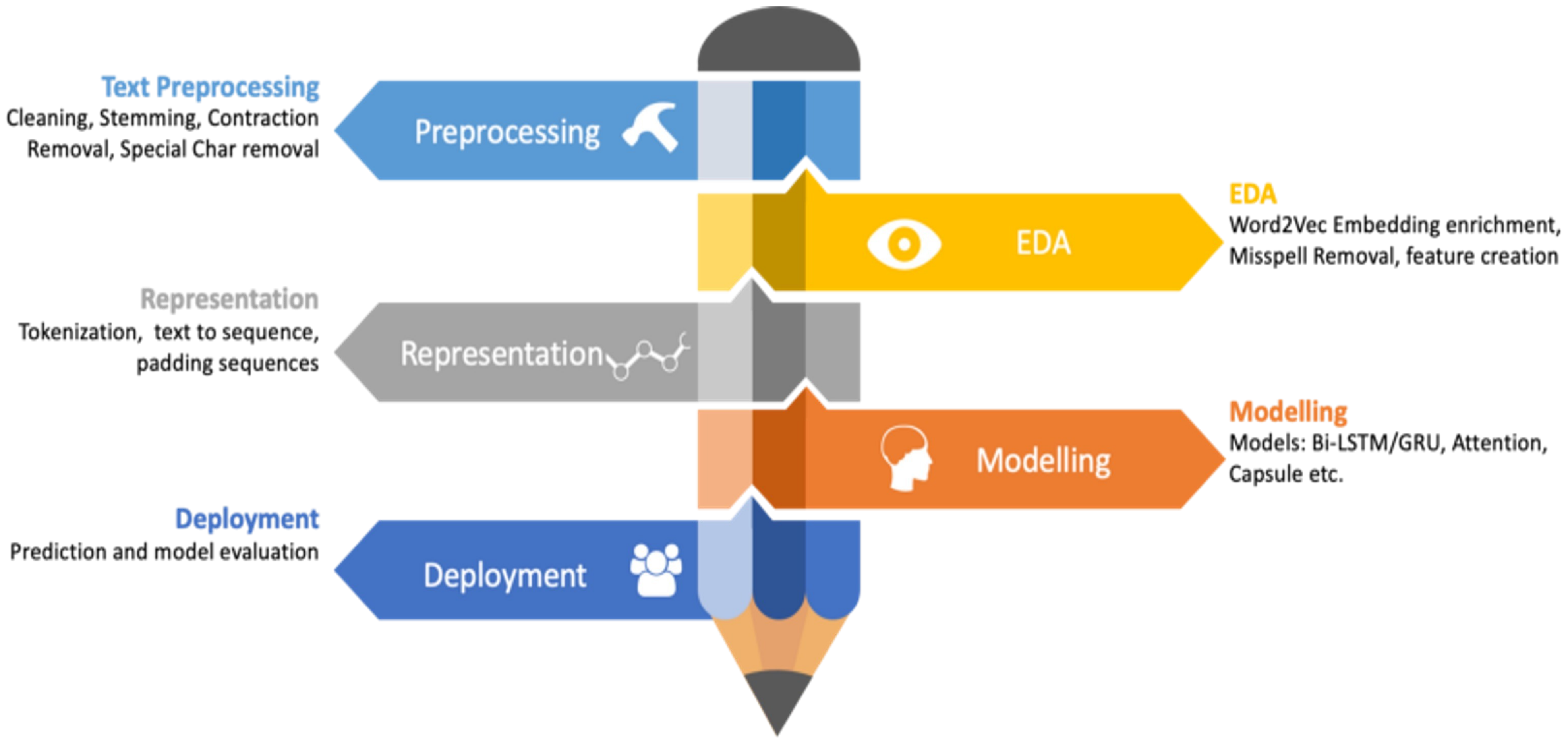



- ❑ Word2Vec
- ❑ Glove
- ❑ Fasttext
- ❑ ELMO

AI VIET NAM
@aivietnam.edu.vn

# 3 - Summary

## NLP Pipeline



# 3 - Summary

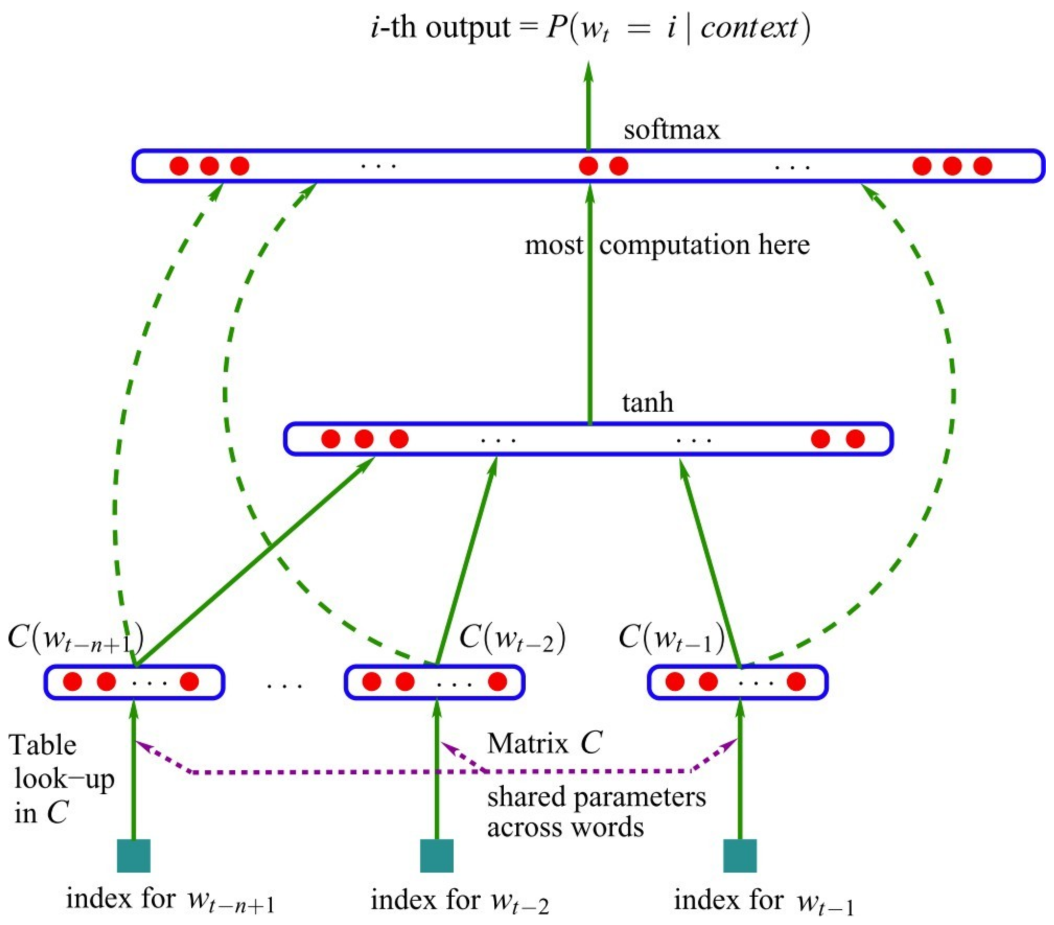
## ❖ The neural history of NLP

2001 — Neural Language
Models
2008 — Multi-task Learning
2013 — Word Embedding
Neural Network for NLP
2014 — Sequence-to-sequence Models
2015 — Attention - Transformer
2018 — Pretrained Language Models

# 3 - Summary

## ❖ The neural history of NLP

| | |
|---|---|
| 2001 | Neural Language Models |
| 2008 | Multi-task Learning |
| 2013 | Word Embedding<br>Neural Network for NLP |
| 2014 | Sequence-to-sequence Models |
| 2015 | Attention - Transformer |
| 2018 | Pretrained Language Models |



# 3 - Summary

## ❖ The neural history of NLP

2001 Neural Language Models

2008

2013

Word Embedding

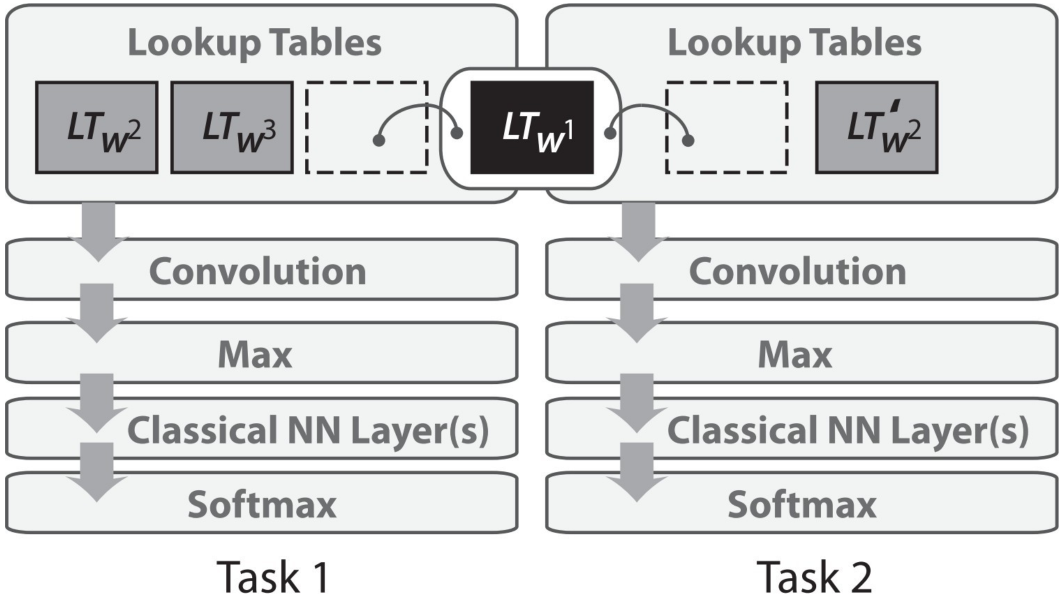
Multi-task Learning

Neural Network for NLP

2014 Sequence-to-sequence Models

2015 Attention - Transformer

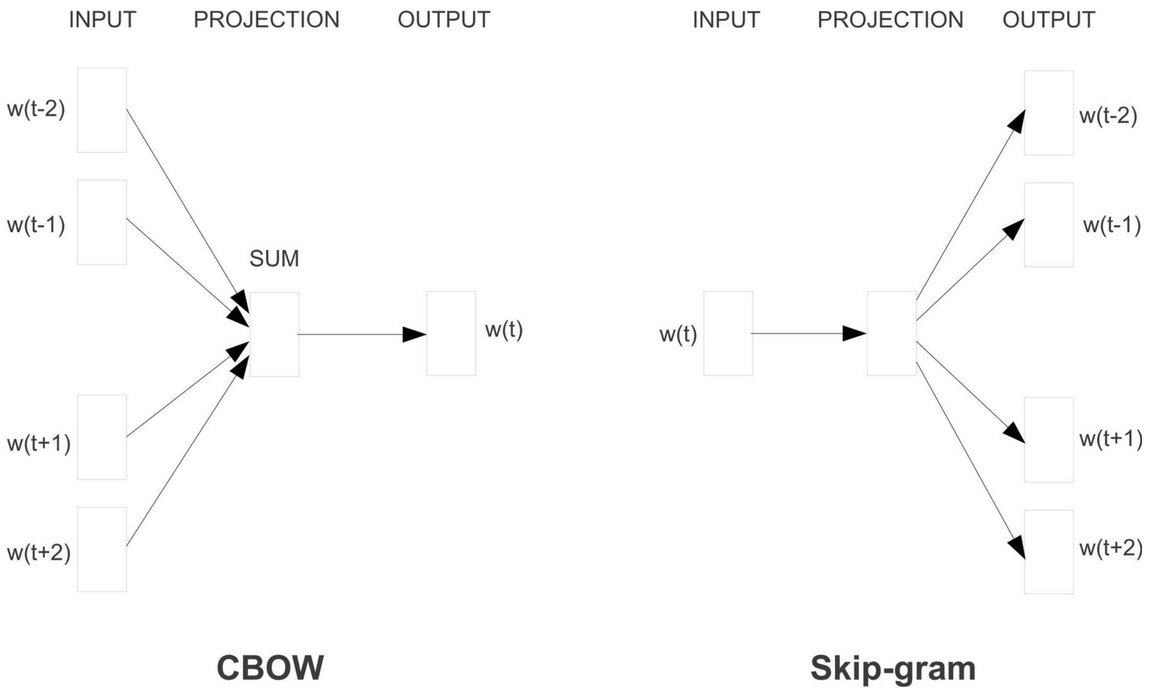
2018 Pretrained Language Models



# 3 - Summary

## ❖ The neural history of NLP

| | |
|---|---|
| 2001 | Neural Language Models |
| 2013 | Word Embedding<br>Neural Network for NLP |
| 2014 | Sequence-to-sequence Models |
| 2015 | Attention - Transformer |
| 2018 | Pretrained Language Models |

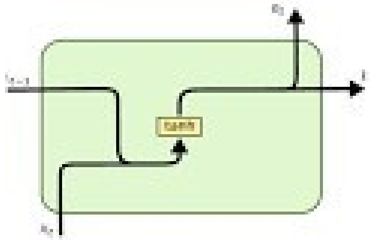


# 3 - Summary

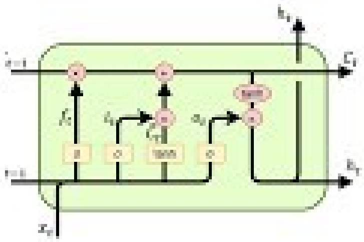
## ❖ The neural history of NLP

| | |
|---|---|
| 2001 Models | Neural Language |
| 2013 | Word Embedding<br>Neural Network for NLP |
| 2014 | Sequence-to-sequence Models |
| 2015 | Attention - Transformer |
| 2018 | Pretrained Language Models |

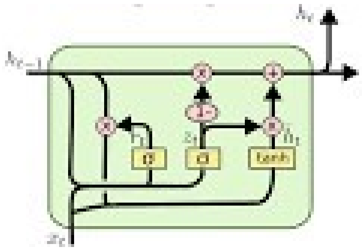
RNN



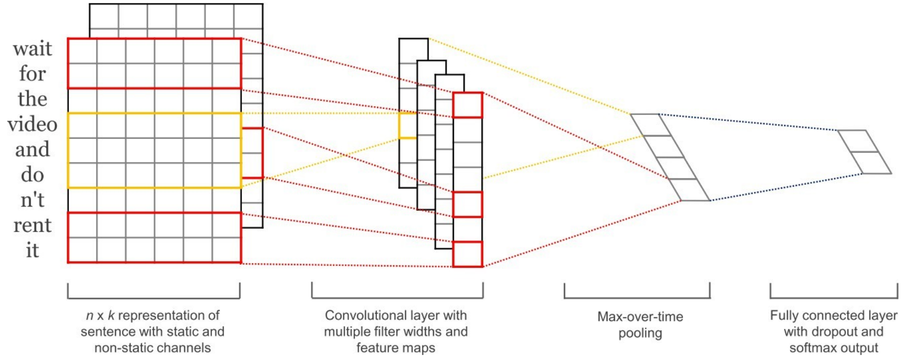
LSTM



GRU



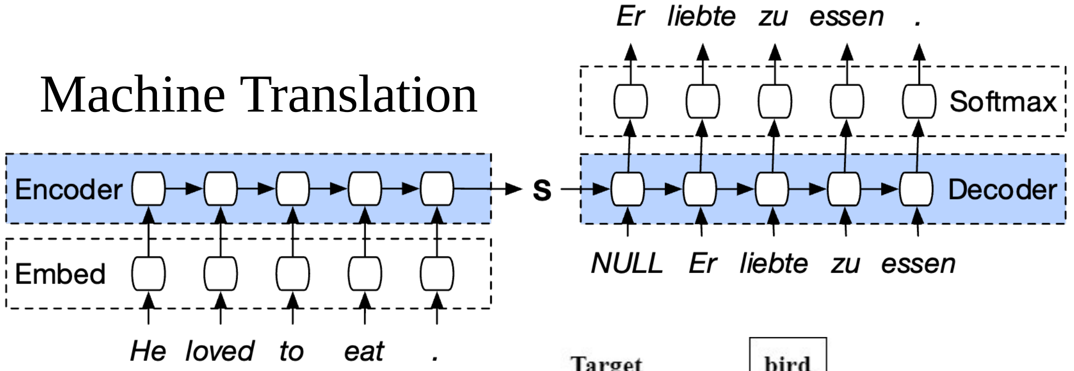
CNN

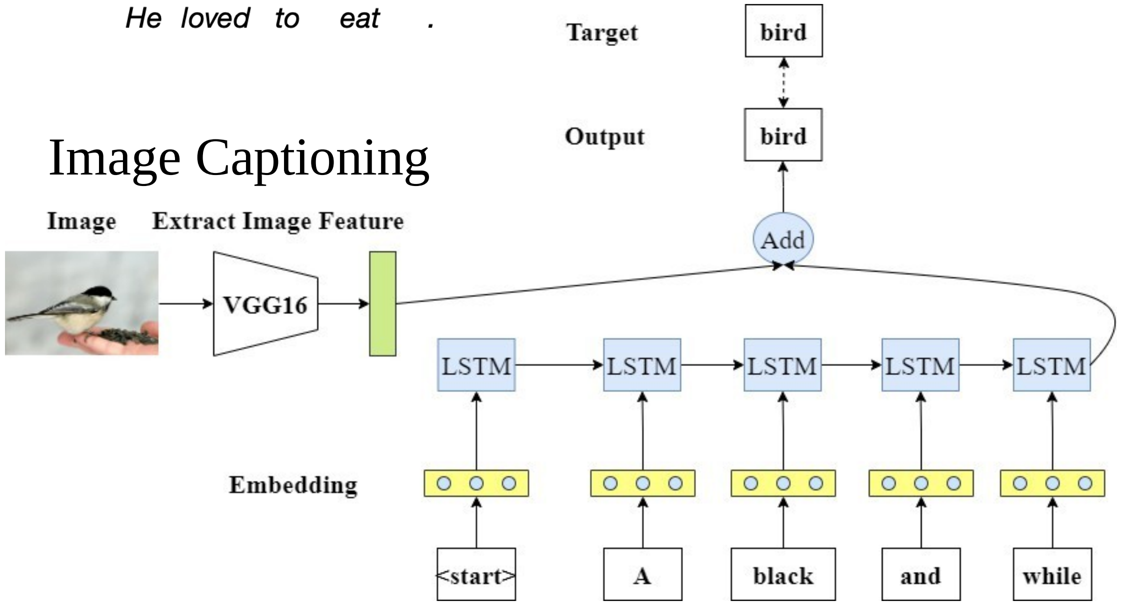


# 3 - Summary

## ❖ The neural history of NLP

| | |
|---|---|
| 2001 | Neural Language |
| Models | |
| 2008 | Multi-task Learning |
| 2013 | Word Embedding<br>Neural Network for NLP |
| **2014** | **Sequence-to-sequence Models** |
| 2015 | Attention - Transformer |
| 2018 | Pretrained Language Models |



# 3 - Summary

## ❖ The neural history of NLP

2001 Neural Language
Models

2008 Multi-task Learning

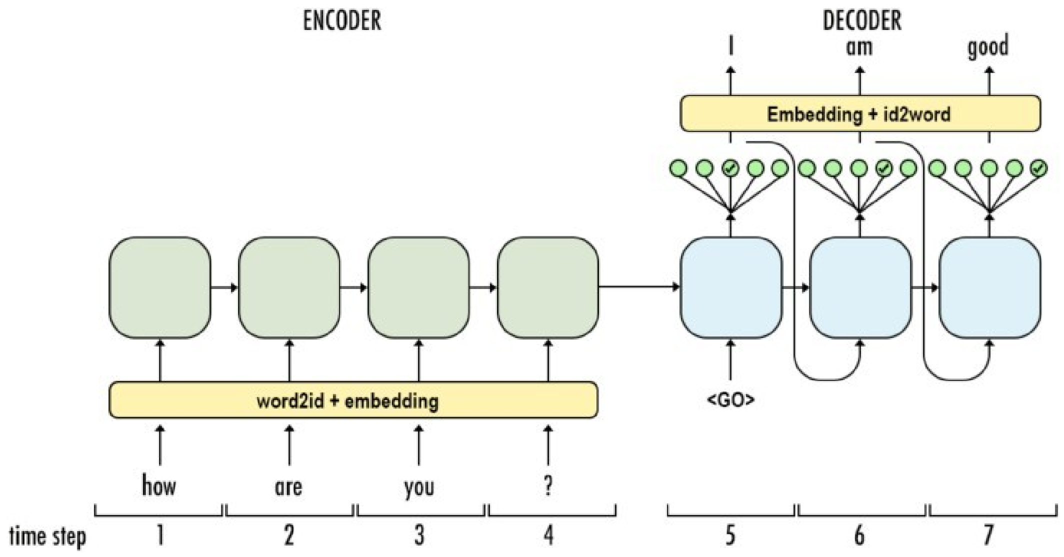
2013 Word Embedding
Neural Network for NLP

2014 Sequence-to-sequence Models

2015 Attention - Transformer

2018 Pretrained Language Models

Question Answer

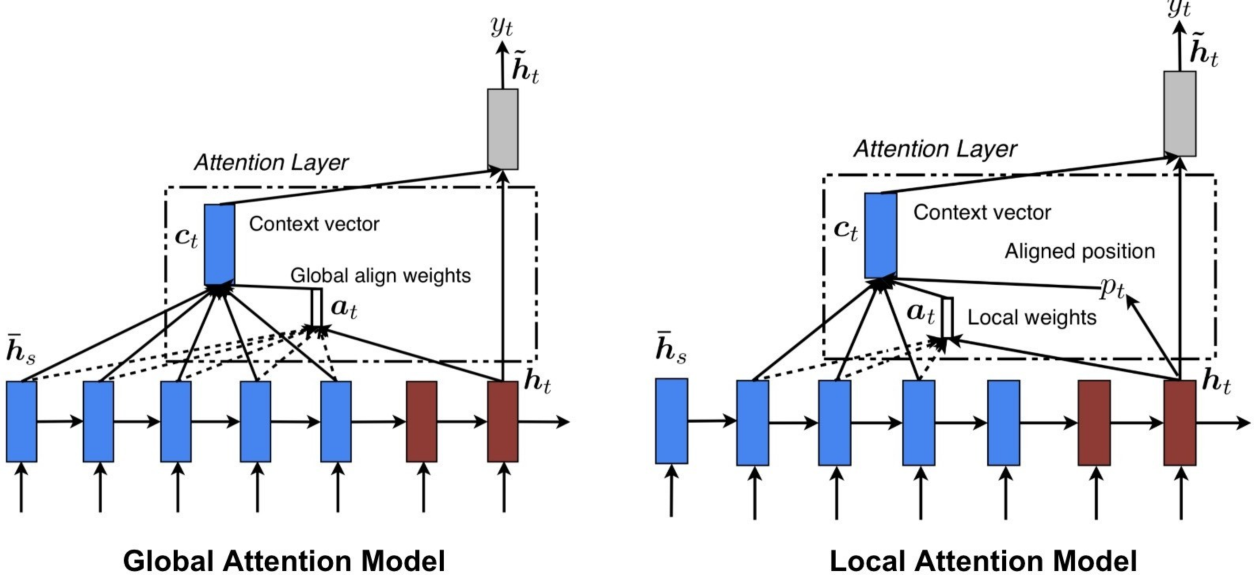


# 3 - Summary

## ❖ The neural history of NLP

| | |
|---|---|
| 2001 | Neural Language Models |
| 2008 | Multi-task Learning |
| 2013 | Word Embedding<br>Neural Network for NLP |
| 2014 | Sequence-to-sequence Models |
| 2015 | Attention - Transformer |
| 2018 | Pretrained Language Models |

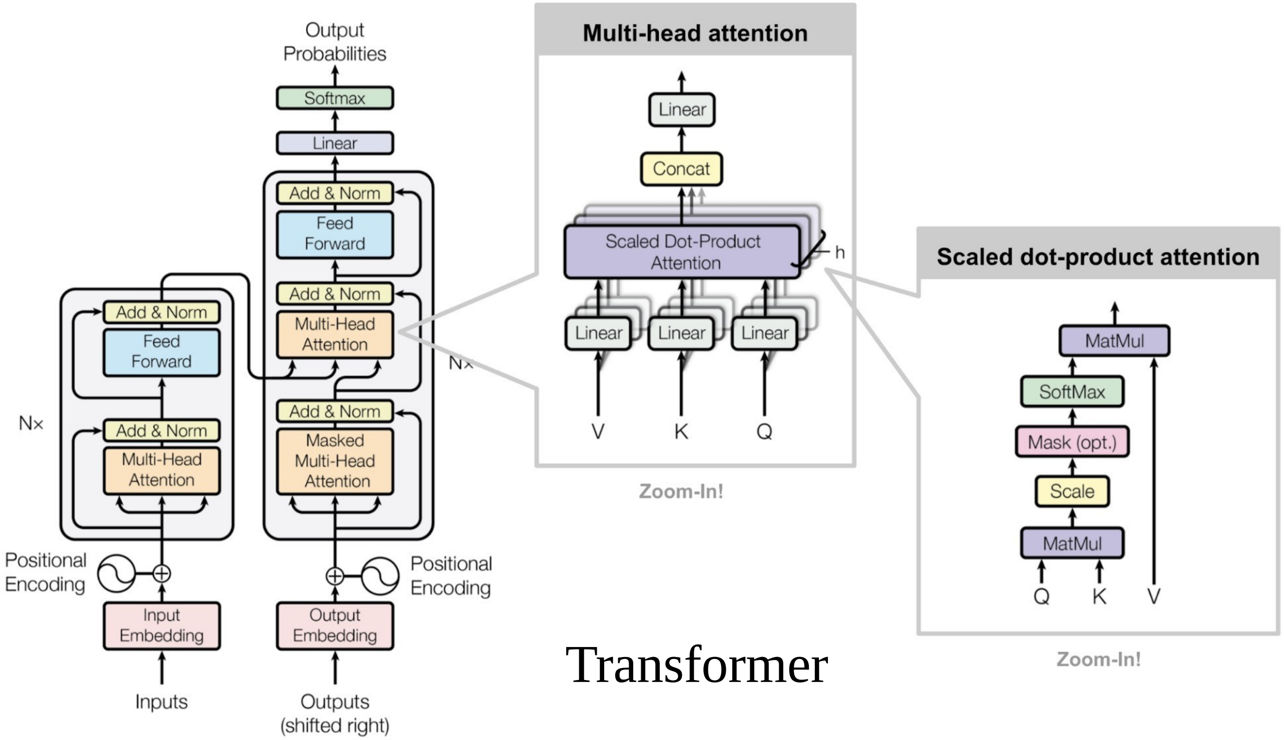
Attention



# 3 - Summary

## The neural history of NLP

- 2001 — Neural Language Models
- 2008 — Multi-task Learning
- 2013 — Word Embedding; Neural Network for NLP
- 2014 — Sequence-to-sequence Models
- 2015 — Attention - Transformer
- 2018 — Pretrained Language Models

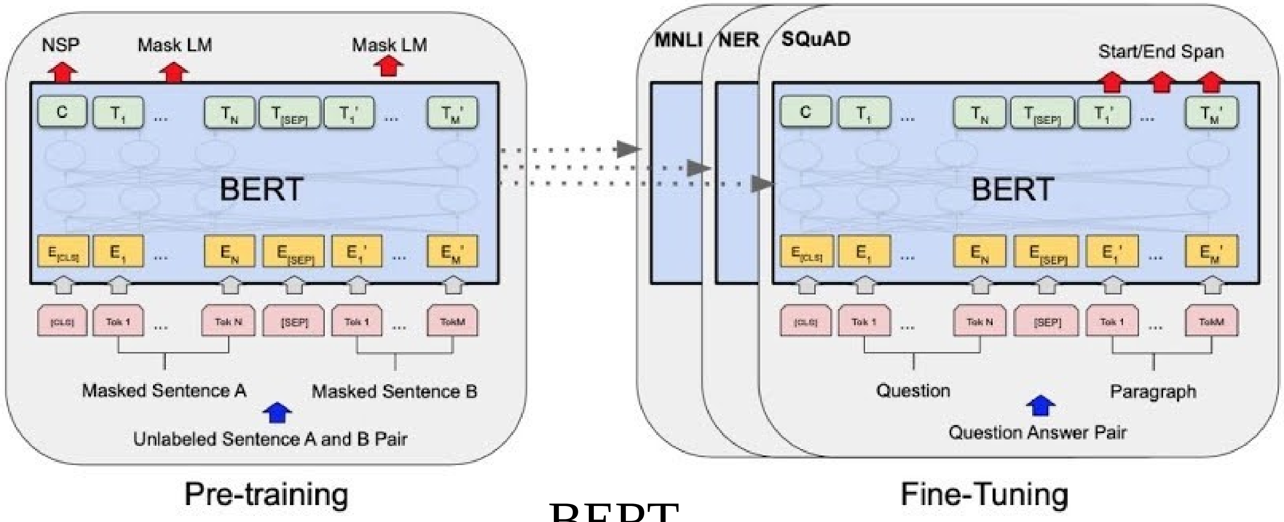



Transformer

# 3 - Summary

## ❖ The neural history of NLP

| | |
|---|---|
| 2001 | Neural Language Models |
| 2008 | Multi-task Learning |
| 2013 | Word Embedding<br>Neural Network for NLP |
| 2014 | Sequence-to-sequence Models |
| 2015 | Attention - Transformer |
| **2018** | **Pretrained Language Models** |




BERT

# 3 - Summary

## ❖ The neural history of NLP

2001 — Neural Language Models

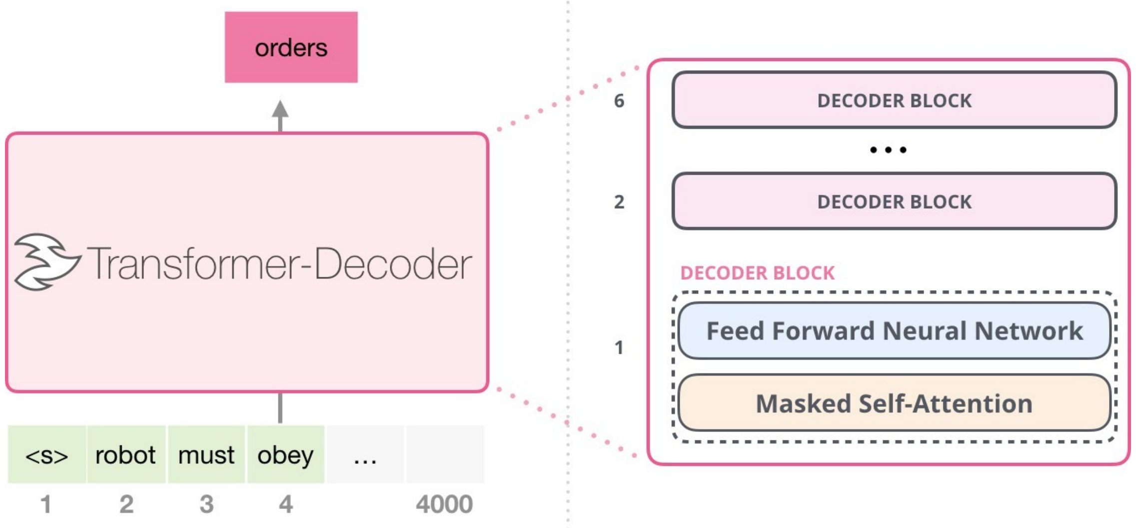
2008 — Multi-task Learning

2013 — Word Embedding, Neural Network for NLP

2014 — Sequence-to-sequence Models

2015 — Attention - Transformer

2018 — Pretrained Language Models




GPT

# 3 - Summary

## ❖ Reference

1. https://web.stanford.edu/~jurafsky/slp3/
2. http://web.stanford.edu/class/cs224n/
3. https://d2l.ai/
4. http://nlpprogress.com/
5. https://github.com/undertheseanlp/NLP-Vietnamese-progress

# Thanks!

## Any questions?