Summer semester 2023

344.063 KV Special Topic:

Natural Language Processing with Deep Learning

N-gram Embeddings with Convolutional Neural Networks

Navid Rekab-saz

navid.rekabsaz@jku.at

Institute of Computational Perception

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computational Perception

Agenda

- *N*-Gram Embeddings with CNN
- CNN in practice
  - Document classification
  - From characters to word embedding
  - CNN in information retrieval models

Notation – recap

- $a \longrightarrow$ scalar
- $\boldsymbol{b} \longrightarrow$ vector
  - i^{th} element of $\boldsymbol{b}$ is the scalar b_i
- $\boldsymbol{C} \longrightarrow$ matrix
  - i^{th} vector of $\boldsymbol{C}$ is $\boldsymbol{c}_i$
  - j^{th} element of the i^{th} vector of $\boldsymbol{C}$ is the scalar $c_{i,j}$
- Tensor: generalization of scalar, vector, matrix to any arbitrary dimension

Linear Algebra – Dot product

- $\boldsymbol{a} \cdot \boldsymbol{b}^T = c$
  - dimensions: $1 \times d \cdot d \times 1 = 1$

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

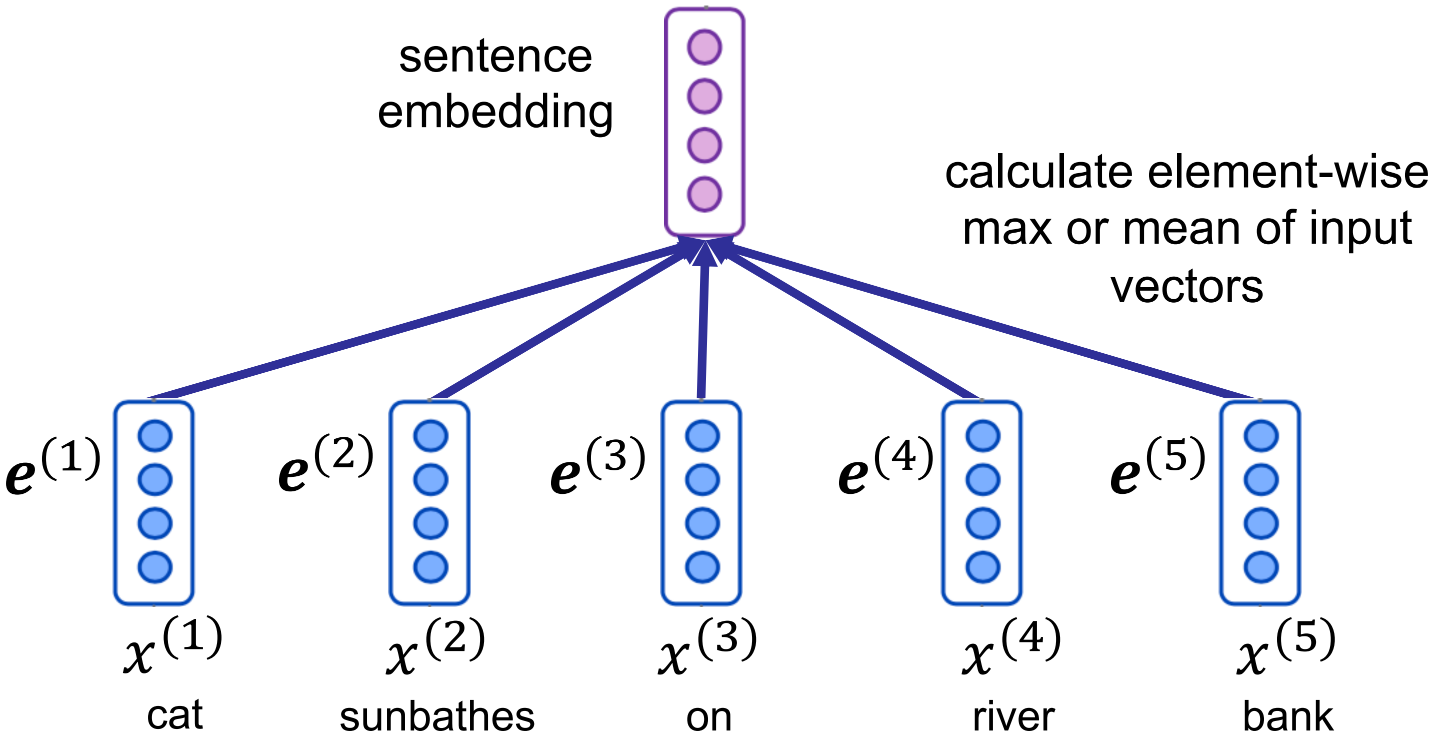
- $\boldsymbol{a} \cdot \boldsymbol{B} = \boldsymbol{c}$
  - dimensions: $1 \times d \cdot d \times e = 1 \times e$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 2 \end{bmatrix}$$

- $\boldsymbol{A} \cdot \boldsymbol{B} = \boldsymbol{C}$
  - dimensions: $l \times m \cdot m \times n = l \times n$

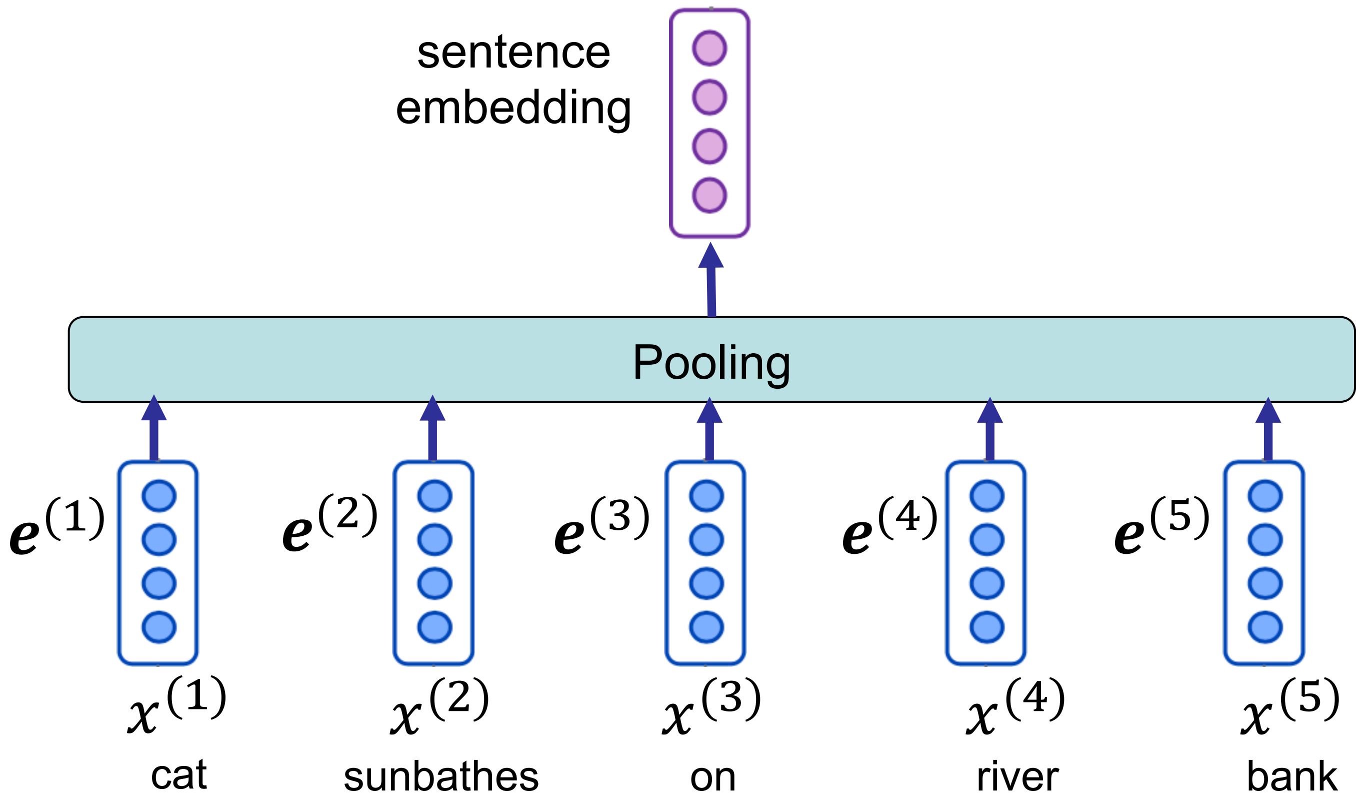
$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 0 & 1 \\ 0 & 0 & 5 \\ 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 2 \\ 3 & 2 \\ 5 & -5 \\ 8 & 13 \end{bmatrix}$$

Sentence embedding from word embeddings



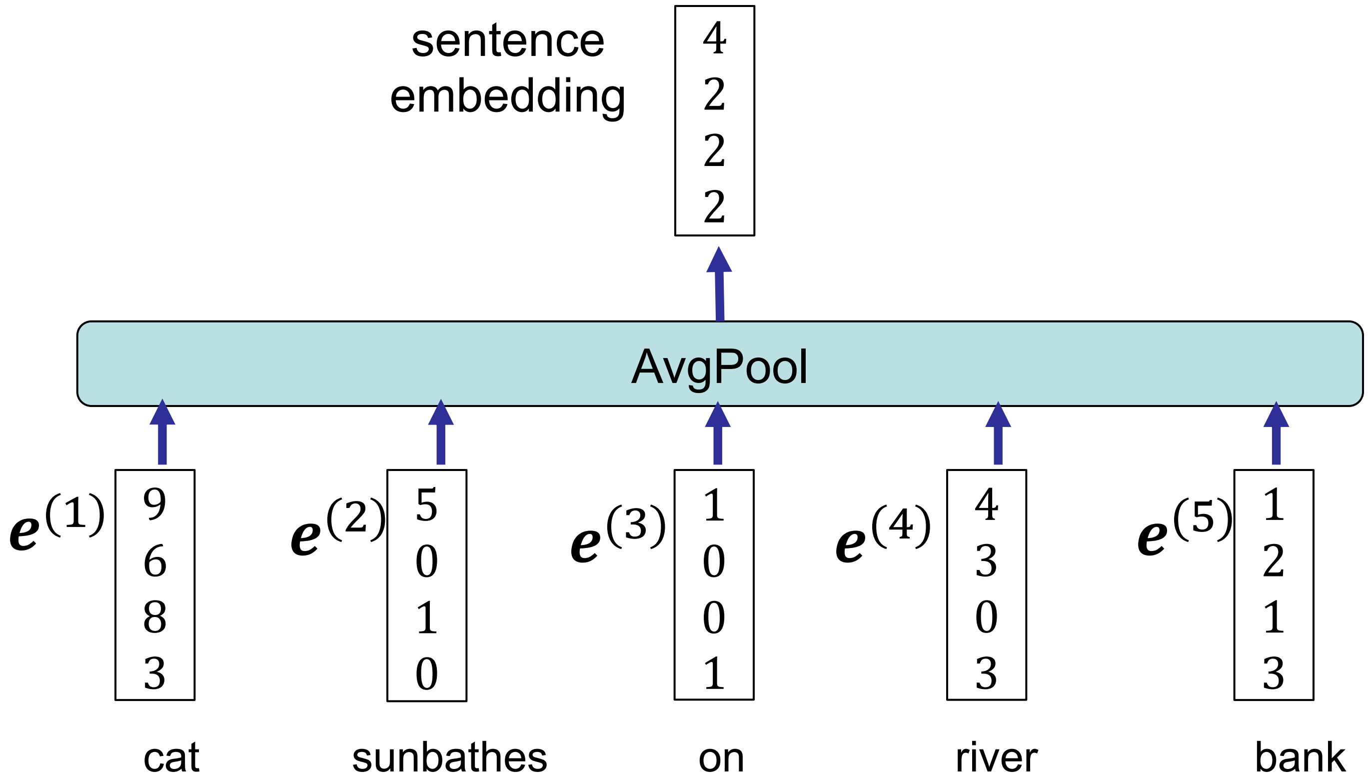
Sentence embedding from word embeddings

- Pooling: element-wise operation on input vectors resulting to an output vector



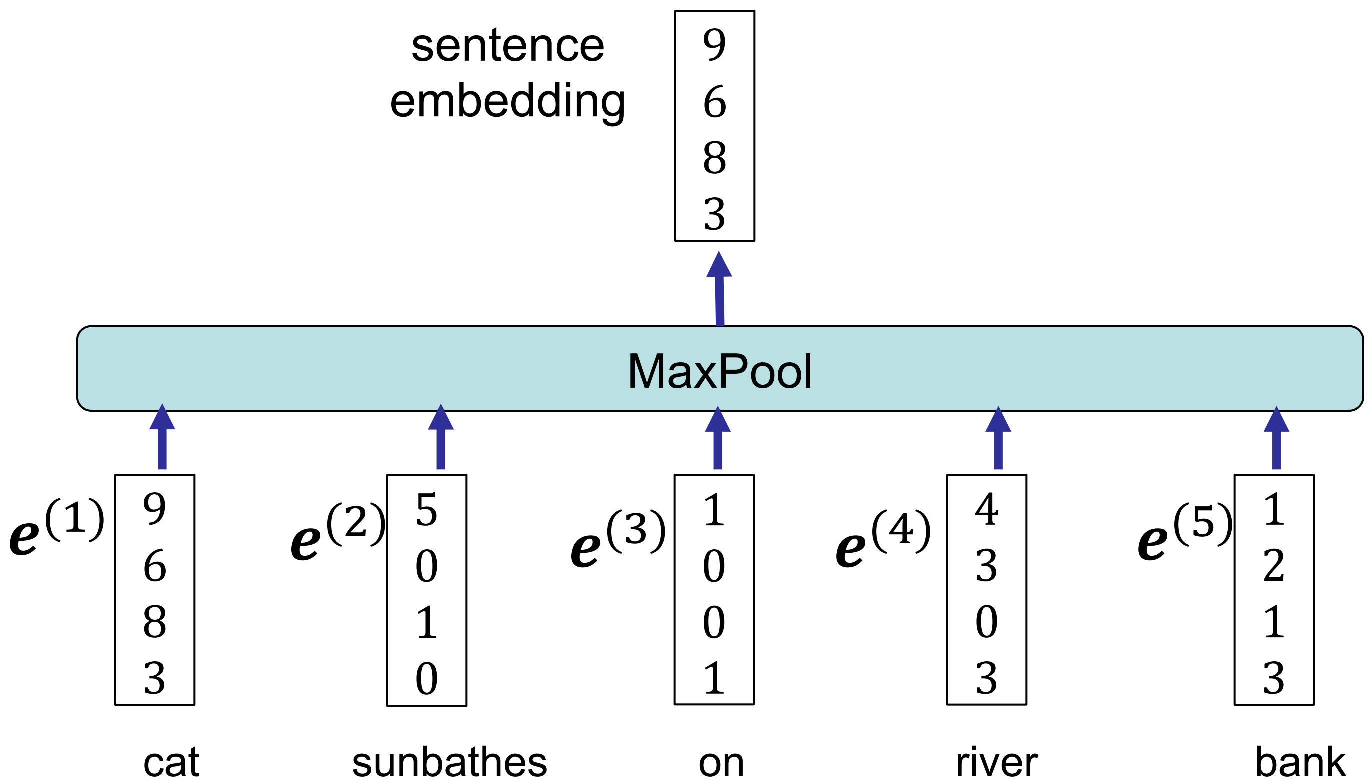
Sentence embedding from word embeddings

- Pooling: element-wise operation on input vectors resulting to an output vector
- AvgPool: element-wise average of inputs



Sentence embedding from word embeddings

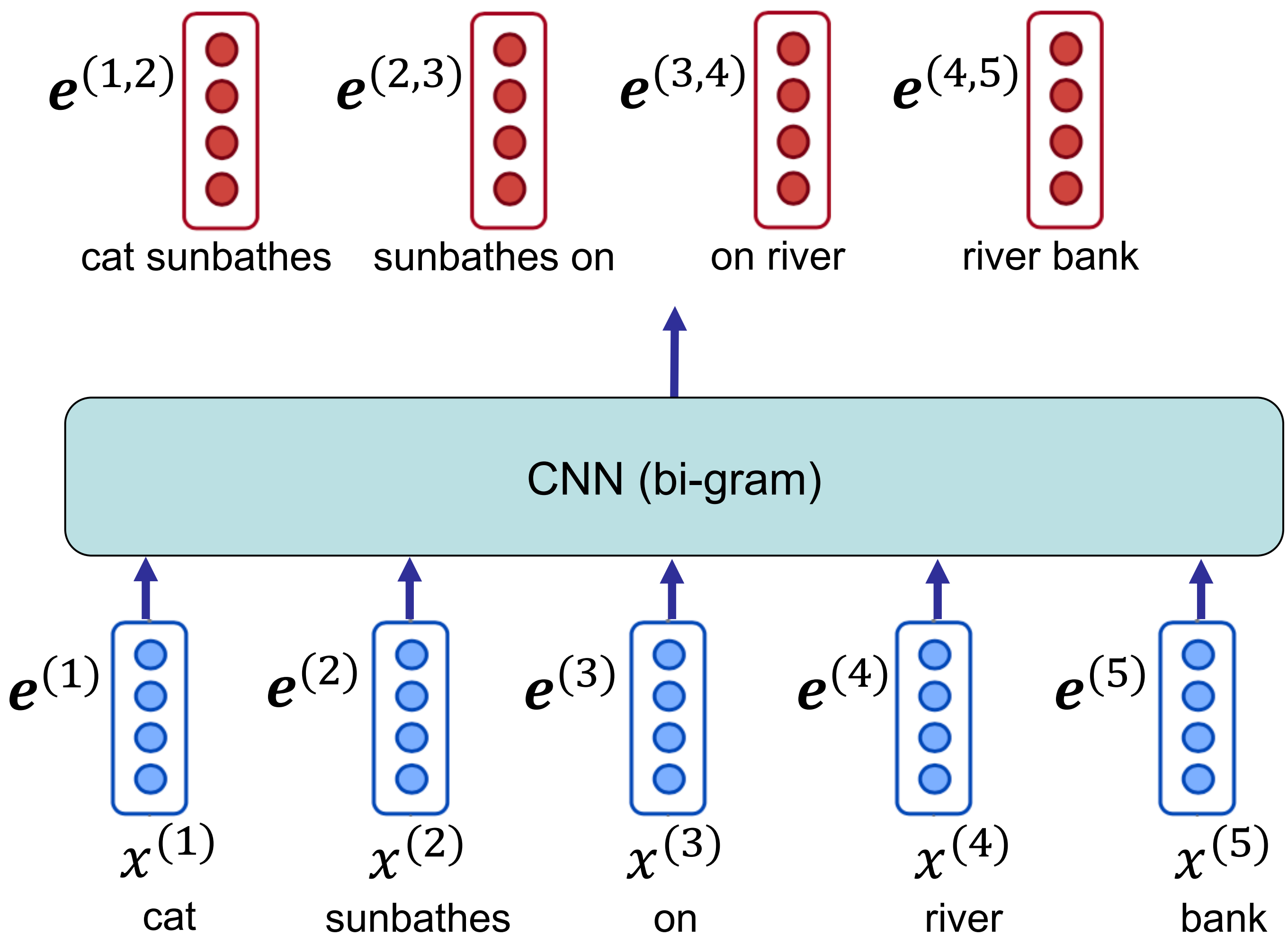
- Pooling: element-wise operation on input vectors resulting to an output vector
- AvgPool: element-wise average of inputs
- MaxPool: element-wise maximum of inputs



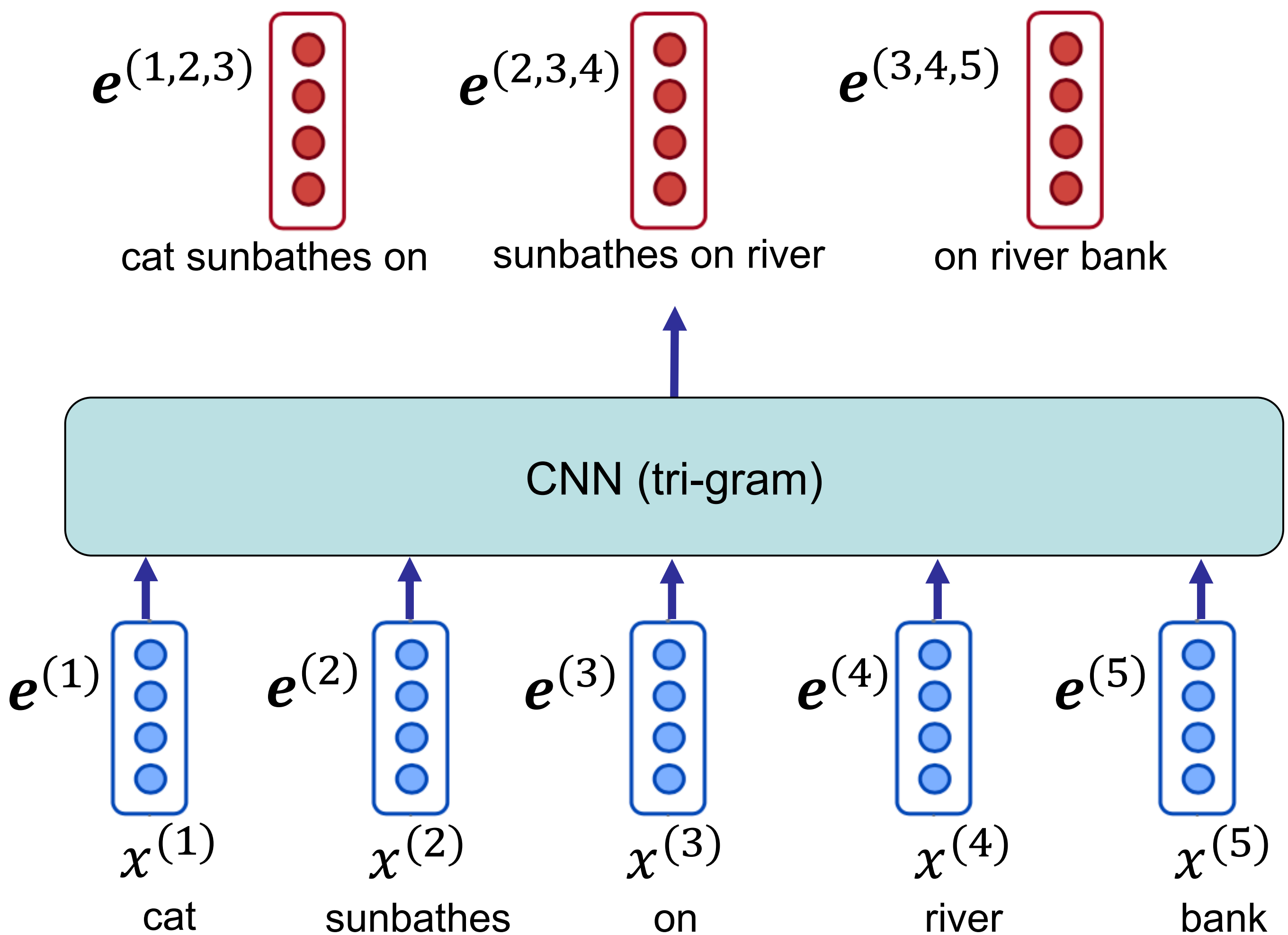
Agenda

- ***N*-Gram Embeddings with CNN**
- CNN in practice
  - Document classification
  - From characters to word embedding
  - CNN in information retrieval models

N-gram embeddings



N-gram embeddings



Convolutional Neural Networks for NLP

- In many NLP models, we can benefit from the vectors which correspond to every sequence of input with a certain length
  - Like bi-gram, tri-gram, 4-gram embeddings

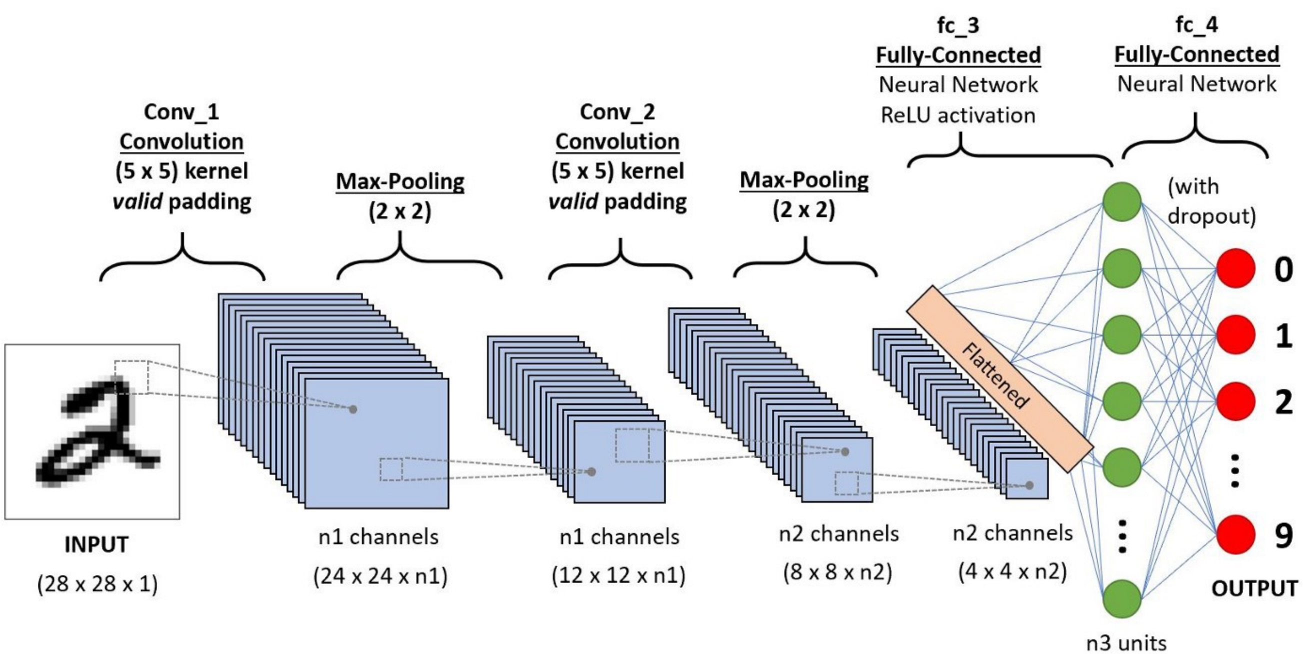
This lecture

- First part: How to create *n*-gram embeddings using Convolutional Neural Nets (CNNs)

- Second part: How to use these embeddings in different NLP models

CNNs

- CNNs are widely used to extract features from images
  - CNNs capture position-invariant patterns from the input data, where …
  - the patterns are captured by a set of kernels
- Kernel (or filter)
  - A kernel is a set of parameters, …
  - applied to every sequence of input values of a certain length …
  - to create the output vector in respect to that sequence



Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

CNNs

- CNNs are widely used to extract features from images
  - CNNs capture position-invariant patterns from the input data, where …
  - the patterns are captured by a set of kernels
- Kernel (or filter)
  - A kernel is a set of parameters, …
  - applied to every sequence of input values of a certain length …
  - to create the output vector in respect to that sequence

Example: 2d Image data with Conv2d

Input (like pixel values)

$$\text{height}\left\{\underbrace{\begin{bmatrix} 1 & 0 & 8 & 0 & 1 \\ 4 & 2 & 5 & 1 & 0 \\ 3 & 2 & 0 & 0 & 2 \\ 9 & 3 & 6 & 1 & 4 \\ 2 & 3 & 4 & 1 & 3 \end{bmatrix}}_{\text{width}}\right.$$

Kernel size

3×3

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

Computing convolution

$$\begin{bmatrix} 1\times1 & 0\times0 & 8\times0 & 0 & 1 \\ 4\times0 & 2\times1 & 5\times1 & 1 & 0 \\ 3\times1 & 2\times1 & 0\times0 & 0 & 2 \\ 9 & 3 & 6 & 1 & 4 \\ 2 & 3 & 4 & 1 & 3 \end{bmatrix}$$

$$1\times1 + 0\times0 + 8\times0 + 4\times0 + 2\times1 + 5\times1 + 3\times1 + 2\times1 + 0\times0 = 13$$

Output (convolved feature)

$$\begin{bmatrix} 13 & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}$$

Parameters are shown in red

2-dimensional CNN (CONV2D) – 2d image with 1 input channel

Input channel

$|C_{in}| = 1$

$$\text{height}\left\{\begin{array}{|ccccc|}\hline 1 & 0 & 8 & 0 & 1 \\ 4 & 2 & 5 & 1 & 0 \\ 3 & 2 & 0 & 0 & 2 \\ 9 & 3 & 6 & 1 & 4 \\ 2 & 3 & 4 & 1 & 3 \\ \hline \end{array}\right.$$

width

Kernel size

3×3

$$\begin{array}{|ccc|}\hline \color{red}{1} & \color{red}{0} & \color{red}{0} \\ \color{red}{0} & \color{red}{1} & \color{red}{1} \\ \color{red}{1} & \color{red}{1} & \color{red}{0} \\ \hline \end{array}$$

Computing convolution

$$\begin{array}{|ccccc|}\hline 1\color{red}{\times1} & 0\color{red}{\times0} & 8\color{red}{\times0} & 0 & 1 \\ 4\color{red}{\times0} & 2\color{red}{\times1} & 5\color{red}{\times1} & 1 & 0 \\ 3\color{red}{\times1} & 2\color{red}{\times1} & 0\color{red}{\times0} & 0 & 2 \\ 9 & 3 & 6 & 1 & 4 \\ 2 & 3 & 4 & 1 & 3 \\ \hline \end{array}$$

$$1\times1 + 0\times0 + 8\times0 + 4\times0 + 2\times1 + 5\times1 + 3\times1 + 2\times1 + 0\times0 = 13$$

Output channel

$|C_{out}| = 1$

$$\begin{array}{|ccc|}\hline 13 & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \hline \end{array}$$

Parameters are shown in red

2-dimensional CNN (CONV2D) – 2d image with 1 input channel

Input channel

$|C_{in}| = 1$

$$\text{height}\left\{\underbrace{\begin{array}{|ccccc|}\hline 1 & 0 & 8 & 0 & 1 \\ 4 & 2 & 5 & 1 & 0 \\ 3 & 2 & 0 & 0 & 2 \\ 9 & 3 & 6 & 1 & 4 \\ 2 & 3 & 4 & 1 & 3 \\ \hline \end{array}}_{\text{width}}\right.$$

Kernel size

3×3

$$\begin{array}{|ccc|}\hline \color{red}{1} & \color{red}{0} & \color{red}{0} \\ \color{red}{0} & \color{red}{1} & \color{red}{1} \\ \color{red}{1} & \color{red}{1} & \color{red}{0} \\ \hline \end{array}$$

Computing convolution

$$\begin{array}{|ccccc|}\hline 1\color{red}{\times1} & 0\color{red}{\times0} & 8\color{red}{\times0} & 0 & 1 \\ 4\color{red}{\times0} & 2\color{red}{\times1} & 5\color{red}{\times1} & 1 & 0 \\ 3\color{red}{\times1} & 2\color{red}{\times1} & 0\color{red}{\times0} & 0 & 2 \\ 9 & 3 & 6 & 1 & 4 \\ 2 & 3 & 4 & 1 & 3 \\ \hline \end{array}$$

$$1\times1+0\times0+8\times0+4\times0+2\times1 + 5\times1+3\times1+2\times1+0\times0 = 13$$

$$\begin{array}{|ccccc|}\hline 1 & 0\color{red}{\times1} & 8\color{red}{\times0} & 0\color{red}{\times0} & 1 \\ 4 & 2\color{red}{\times0} & 5\color{red}{\times1} & 1\color{red}{\times1} & 0 \\ 3 & 2\color{red}{\times1} & 0\color{red}{\times1} & 0\color{red}{\times0} & 2 \\ 9 & 3 & 6 & 1 & 4 \\ 2 & 3 & 4 & 1 & 3 \\ \hline \end{array}$$

$$0\times1+8\times0+0\times0+2\times0+5\times1 + 1\times1+2\times1+0\times1+0\times0 = 8$$

Output channel

$|C_{out}| = 1$

$$\begin{array}{|ccc|}\hline 13 & 8 & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \hline \end{array}$$

Parameters are shown in red

2-dimensional CNN (CONV2D) – 2d image with 1 input channel

Input channel

$|C_{in}| = 1$

$$\text{height}\left\{\begin{array}{|ccccc|} \hline 1 & 0 & 8 & 0 & 1 \\ 4 & 2 & 5 & 1 & 0 \\ 3 & 2 & 0 & 0 & 2 \\ 9 & 3 & 6 & 1 & 4 \\ 2 & 3 & 4 & 1 & 3 \\ \hline \end{array}\right.$$

$$\underbrace{\hphantom{1\quad 0\quad 8\quad 0\quad 1}}_{\text{width}}$$

Kernel size

3×3

$$\begin{array}{|ccc|} \hline 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ \hline \end{array}$$

Computing convolution

$$\begin{array}{|ccccc|} \hline 1\times1 & 0\times0 & 8\times0 & 0 & 1 \\ 4\times0 & 2\times1 & 5\times1 & 1 & 0 \\ 3\times1 & 2\times1 & 0\times0 & 0 & 2 \\ 9 & 3 & 6 & 1 & 4 \\ 2 & 3 & 4 & 1 & 3 \\ \hline \end{array}$$

$1\times1 + 0\times0 + 8\times0 + 4\times0 + 2\times1 + 5\times1 + 3\times1 + 2\times1 + 0\times0 = 13$

$$\begin{array}{|ccccc|} \hline 1 & 0\times1 & 8\times0 & 0\times0 & 1 \\ 4 & 2\times0 & 5\times1 & 1\times1 & 0 \\ 3 & 2\times1 & 0\times1 & 0\times0 & 2 \\ 9 & 3 & 6 & 1 & 4 \\ 2 & 3 & 4 & 1 & 3 \\ \hline \end{array}$$

$0\times1 + 8\times0 + 0\times0 + 2\times0 + 5\times1 + 1\times1 + 2\times1 + 0\times1 + 0\times0 = 8$

$$\begin{array}{|ccccc|} \hline 1 & 0 & 8 & 0 & 1 \\ 4 & 2\times1 & 5\times0 & 1\times0 & 0 \\ 3 & 2\times0 & 0\times1 & 0\times1 & 2 \\ 9 & 3\times1 & 6\times1 & 1\times0 & 4 \\ 2 & 3 & 4 & 1 & 3 \\ \hline \end{array}$$

$2\times1 + 5\times0 + 1\times0 + 2\times0 + 0\times1 + 0\times1 + 3\times1 + 6\times1 + 1\times0 = 11$

Output channel

$|C_{out}| = 1$

$$\begin{array}{|ccc|} \hline 13 & 8 & \cdots \\ \cdots & 11 & \cdots \\ \cdots & \cdots & \cdots \\ \hline \end{array}$$

Calculate other values!

Parameters are shown in red

2-dimensional CNN (CONV2D) – 2d image with 3 input channels

Input channels (like RGB) $|C_{in}| = 3$

Kernel size 3×3

Computing convolution

Output channel $|C_{out}| = 1$

$$C_{in}^{(1)} \quad \begin{bmatrix} 1 & 0 & 8 & 0 & 1 \\ 4 & 2 & 5 & 1 & 0 \\ 3 & 2 & 0 & 0 & 2 \\ 9 & 3 & 6 & 1 & 4 \\ 2 & 3 & 4 & 1 & 3 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1\times1 & 0\times0 & 8\times0 & 0 & 1 \\ 4\times0 & 2\times1 & 5\times1 & 1 & 0 \\ 3\times1 & 2\times1 & 0\times0 & 0 & 2 \\ 9 & 3 & 6 & 1 & 4 \\ 2 & 3 & 4 & 1 & 3 \end{bmatrix}$$

$$C_{in}^{(2)} \quad \begin{bmatrix} 1 & 7 & 4 & 6 & 0 \\ 3 & 1 & 3 & 2 & 1 \\ 5 & 0 & 9 & 5 & 4 \\ 0 & 2 & 6 & 4 & 8 \\ 0 & 0 & 2 & 3 & 2 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1\times0 & 7\times0 & 4\times0 & 6 & 0 \\ 3\times0 & 1\times0 & 3\times0 & 2 & 1 \\ 5\times1 & 0\times0 & 9\times0 & 5 & 4 \\ 0 & 2 & 6 & 4 & 8 \\ 0 & 0 & 2 & 3 & 2 \end{bmatrix} \quad C_{out}^{(1)} \begin{bmatrix} 28 & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}$$

$$C_{in}^{(3)} \quad \begin{bmatrix} 3 & 1 & 0 & 0 & 6 \\ 4 & 2 & 2 & 0 & 7 \\ 2 & 1 & 0 & 0 & 1 \\ 6 & 2 & 0 & 2 & 2 \\ 4 & 1 & 0 & 3 & 6 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 3\times0 & 1\times1 & 0\times1 & 0 & 6 \\ 4\times1 & 2\times0 & 2\times1 & 0 & 7 \\ 2\times1 & 1\times1 & 0\times0 & 0 & 1 \\ 6 & 2 & 0 & 2 & 2 \\ 4 & 1 & 0 & 3 & 6 \end{bmatrix}$$

$$\begin{gathered} (1\times1 + 0\times0 + 8\times0 + 4\times0 + 2\times1 + 5\times1 + 3\times1 + 2\times1 + 0\times0) \\ + (1\times0 + 7\times0 + 4\times0 + 3\times0 + 1\times0 + 3\times0 + 5\times1 + 0\times0 + 9\times0) \\ + (3\times0 + 1\times1 + 0\times1 + 4\times1 + 2\times0 + 2\times1 + 2\times1 + 1\times1 + 0\times0) \\ = 28 \end{gathered}$$

Parameters are shown in red

1-dimensional CNN (CONV1D) – towards language processing

Input channels

$|C_{in}| = 4$

$C_{in}^{(4)}$	1	0	8	0	1
$C_{in}^{(3)}$	1	7	4	6	0
$C_{in}^{(2)}$	3	1	0	0	6
$C_{in}^{(1)}$	4	2	1	0	2

input sequence length = 5

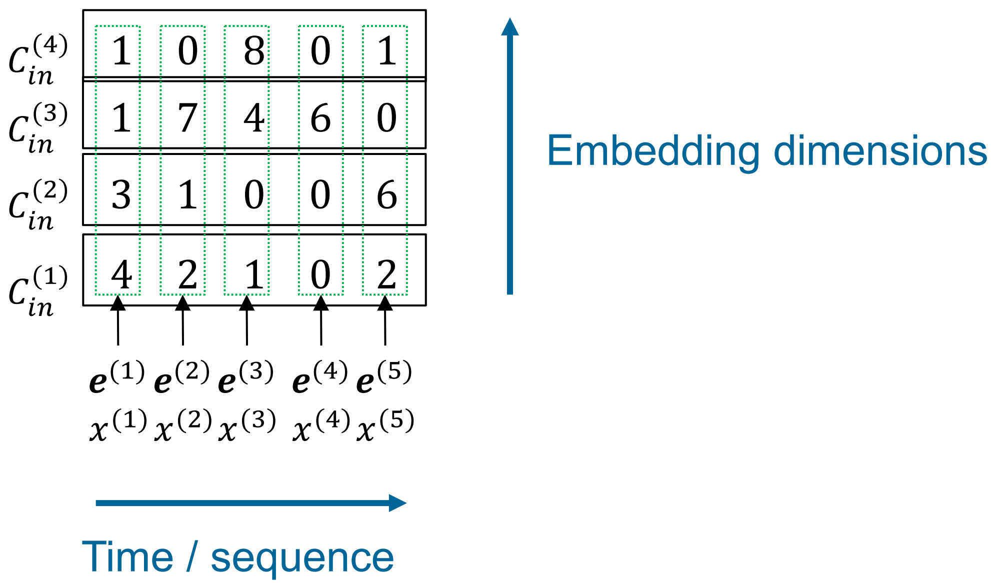
1-dimensional CNN in NLP

Input channels

$|C_{in}| = 4$

Number of input channels $|C_{in}|$
=
dimension of word embedding.

Conv1d sees every dimension as a channel



Parameters are shown in red

1-dimensional CNN in NLP

Input channels
$|C_{in}| = 4$

Kernel size
$k = 3$

Computing convolution

Output channel
$|C_{out}| = 1$

$\boldsymbol{w}_i^{(j)}$
kernel weights for
jth input channel and
ith output channel

$$
\begin{array}{c|ccccc|c|ccc|}
C_{in}^{(4)} & 1 & 0 & 8 & 0 & 1 & \boldsymbol{w}_1^{(4)} & 1 & 0 & 0 \\
C_{in}^{(3)} & 1 & 7 & 4 & 6 & 0 & \boldsymbol{w}_1^{(3)} & 0 & 0 & 0 \\
C_{in}^{(2)} & 3 & 1 & 0 & 0 & 6 & \boldsymbol{w}_1^{(2)} & 0 & 1 & 1 \\
C_{in}^{(1)} & 4 & 2 & 1 & 0 & 2 & \boldsymbol{w}_1^{(1)} & 1 & 0 & 1
\end{array}
$$

input sequence length = 5

kernel size = 3

$C_{out}^{(1)} \quad \underbrace{\cdots \quad \cdots \quad \cdots}_{}$

output sequence length = 3

Parameters are shown in red

1-dimensional CNN in NLP

Input channels

$|C_{in}| = 4$

Kernel size

$k = 3$

Computing convolution

1×1	0×0	8×0	0	1
1×0	7×0	4×0	6	0
3×0	1×1	0×1	0	6
4×1	2×0	1×1	0	2

$(1\times1 + 0\times0 + 8\times0) + (1\times0 + 7\times0 + 4\times0) +$
$(3\times0 + 1\times1 + 0\times1) + (4\times1 + 2\times0 + 1\times1) = 7$

Output channel

$|C_{out}| = 1$

$\boldsymbol{w}_i^{(j)}$
kernel weights for
jth input channel and
ith output channel

$C_{in}^{(4)}$	1	0	8	0	1	$\boldsymbol{w}_1^{(4)}$	1	0	0
$C_{in}^{(3)}$	1	7	4	6	0	$\boldsymbol{w}_1^{(3)}$	0	0	0
$C_{in}^{(2)}$	3	1	0	0	6	$\boldsymbol{w}_1^{(2)}$	0	1	1
$C_{in}^{(1)}$	4	2	1	0	2	$\boldsymbol{w}_1^{(1)}$	1	0	1

input sequence length = 5

kernel size = 3

$C_{out}^{(1)}$ $\underbrace{7 \quad \cdots \quad \cdots}_{\text{output sequence length = 3}}$

Parameters are shown in red

1-dimensional CNN in NLP

Input channels

$|C_{in}| = 4$

Kernel size

$k = 3$

Computing convolution

Output channel

$|C_{out}| = 1$

$\boldsymbol{w}_i^{(j)}$
kernel weights for
jth input channel and
ith output channel

$C_{in}^{(4)}$	1	0	8	0	1	$\boldsymbol{w}_1^{(4)}$	1 0 0
$C_{in}^{(3)}$	1	7	4	6	0	$\boldsymbol{w}_1^{(3)}$	0 0 0
$C_{in}^{(2)}$	3	1	0	0	6	$\boldsymbol{w}_1^{(2)}$	0 1 1
$C_{in}^{(1)}$	4	2	1	0	2	$\boldsymbol{w}_1^{(1)}$	1 0 1

input sequence length = 5

kernel size = 3

1×1	0×0	8×0	0	1
1×0	7×0	4×0	6	0
3×0	1×1	0×1	0	6
4×1	2×0	1×1	0	2

$$(1\times1 + 0\times0 + 8\times0) + (1\times0 + 7\times0 + 4\times0) + (3\times0 + 1\times1 + 0\times1) + (4\times1 + 2\times0 + 1\times1) = 7$$

1	0×1	8×0	0×0	1
1	7×0	4×0	6×0	0
3	1×0	0×1	0×1	6
4	2×1	1×0	0×1	2

$$(0\times1 + 8\times0 + 0\times0) + (7\times0 + 4\times0 + 6\times0) + (1\times0 + 0\times1 + 0\times1) + (2\times1 + 1\times0 + 0\times1) = 2$$

$C_{out}^{(1)}$ $\underbrace{7 \quad 2 \quad \cdots}_{\text{output sequence length = 3}}$

Parameters are shown in red

1-dimensional CNN in NLP

Input channels

$|C_{in}| = 4$

Kernel size

$k = 3$

Computing convolution

Output channel

$|C_{out}| = 1$

> $\boldsymbol{w}_i^{(j)}$
> kernel weights for jth input channel and ith output channel

$C_{in}^{(4)}$	1	0	8	0	1
$C_{in}^{(3)}$	1	7	4	6	0
$C_{in}^{(2)}$	3	1	0	0	6
$C_{in}^{(1)}$	4	2	1	0	2

input sequence length = 5

$\boldsymbol{w}_1^{(4)}$	1	0	0
$\boldsymbol{w}_1^{(3)}$	0	0	0
$\boldsymbol{w}_1^{(2)}$	0	1	1
$\boldsymbol{w}_1^{(1)}$	1	0	1

kernel size = 3

1×1	0×0	8×0	0	1
1×0	7×0	4×0	6	0
3×0	1×1	0×1	0	6
4×1	2×0	1×1	0	2

$(1{\times}1 + 0{\times}0 + 8{\times}0) + (1{\times}0 + 7{\times}0 + 4{\times}0) + (3{\times}0 + 1{\times}1 + 0{\times}1) + (4{\times}1 + 2{\times}0 + 1{\times}1) = 7$

1	0×1	8×0	0×0	1
1	7×0	4×0	6×0	0
3	1×0	0×1	0×1	6
4	2×1	1×0	0×1	2

$(0{\times}1 + 8{\times}0 + 0{\times}0) + (7{\times}0 + 4{\times}0 + 6{\times}0) + (1{\times}0 + 0{\times}1 + 0{\times}1) + (2{\times}1 + 1{\times}0 + 0{\times}1) = 2$

1	0	8×1	0×0	1×0
1	7	4×0	6×0	0×0
3	1	0×0	0×1	6×1
4	2	1×1	0×0	2×1

$(8{\times}1 + 0{\times}0 + 1{\times}0) + (4{\times}0 + 6{\times}0 + 0{\times}0) + (0{\times}0 + 0{\times}1 + 6{\times}1) + (1{\times}1 + 0{\times}0 + 2{\times}1) = 17$

$C_{out}^{(1)}$	7	2	17

output sequence length = 3

Parameters are shown in red

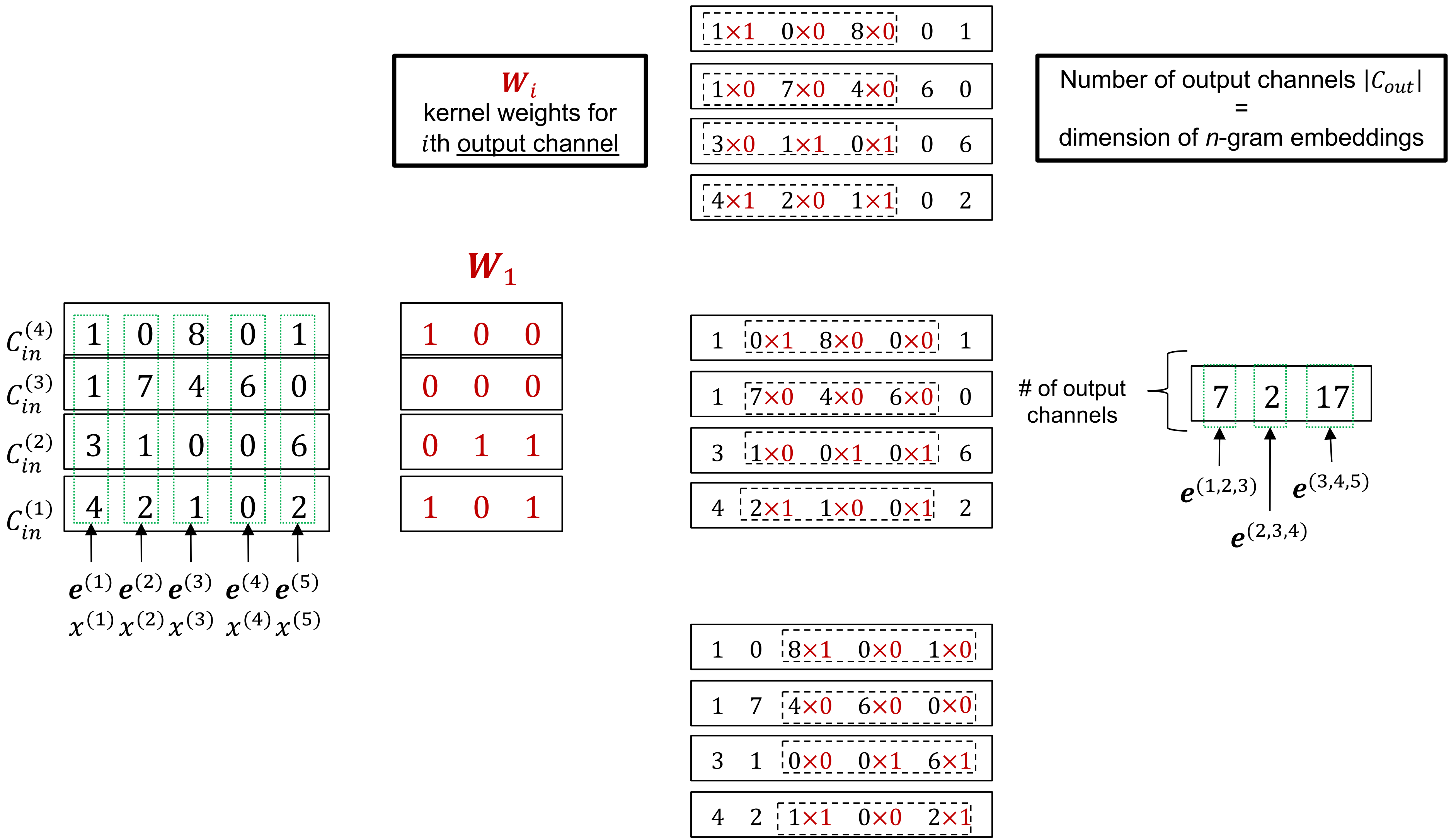
1-dimensional CNN in NLP – 1 output channel

Input channels $|C_{in}| = 4$

Kernel size $k = 3$

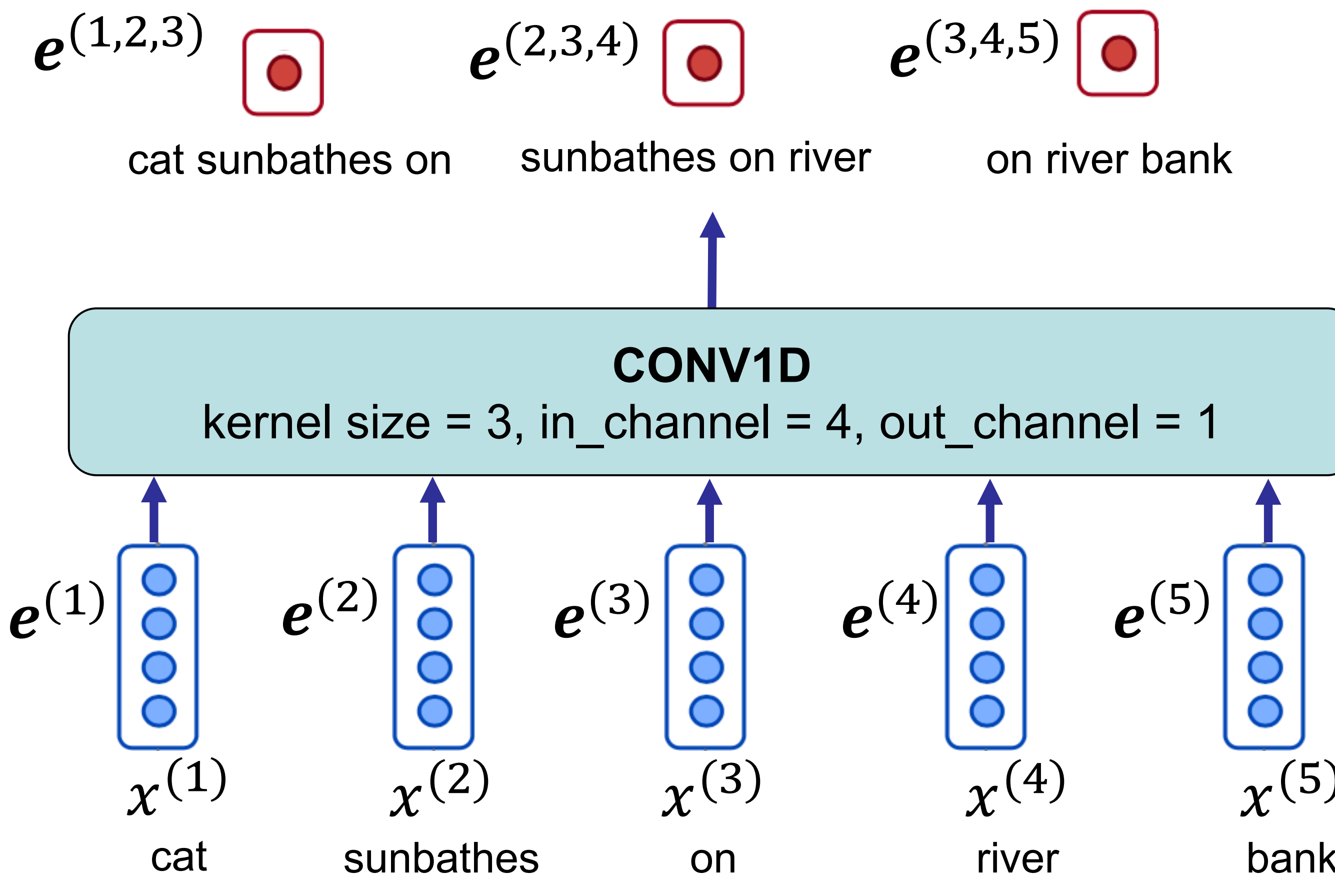
Computing convolution

Output channel $|C_{out}| = 1$



Parameters are shown in red

N-gram embeddings



1-dimensional CNN in NLP – 2 output channels

Input channels
$|C_{in}| = 4$

Kernel size
$k = 3$

Output channels
$|C_{out}| = 2$

$\boldsymbol{W}_i$: kernel weights for ith output channel

$C_{in}^{(4)}$	1	0	8	0	1
$C_{in}^{(3)}$	1	7	4	6	0
$C_{in}^{(2)}$	3	1	0	0	6
$C_{in}^{(1)}$	4	2	1	0	2
	$\boldsymbol{e}^{(1)}$	$\boldsymbol{e}^{(2)}$	$\boldsymbol{e}^{(3)}$	$\boldsymbol{e}^{(4)}$	$\boldsymbol{e}^{(5)}$

$\boldsymbol{W}_1$

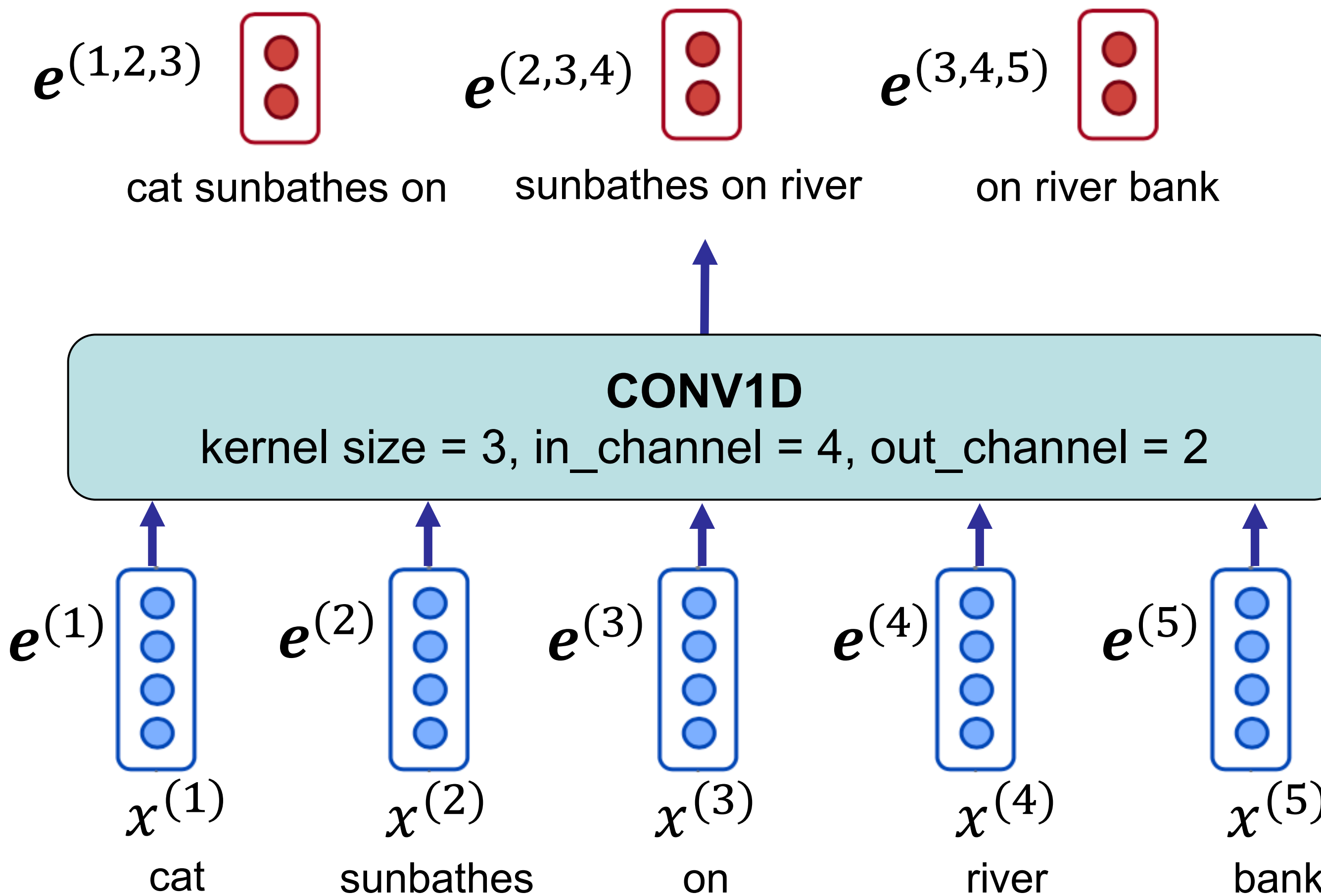
1	0	0
0	0	0
0	1	1
1	0	1

$\boldsymbol{W}_2$

1	1	0
0	0	0
0	0	1
0	0	0

$C_{out}^{(2)}$	1	8	14
$C_{out}^{(1)}$	7	2	17
	$\boldsymbol{e}^{(1,2,3)}$	$\boldsymbol{e}^{(2,3,4)}$	$\boldsymbol{e}^{(3,4,5)}$

N-gram embeddings



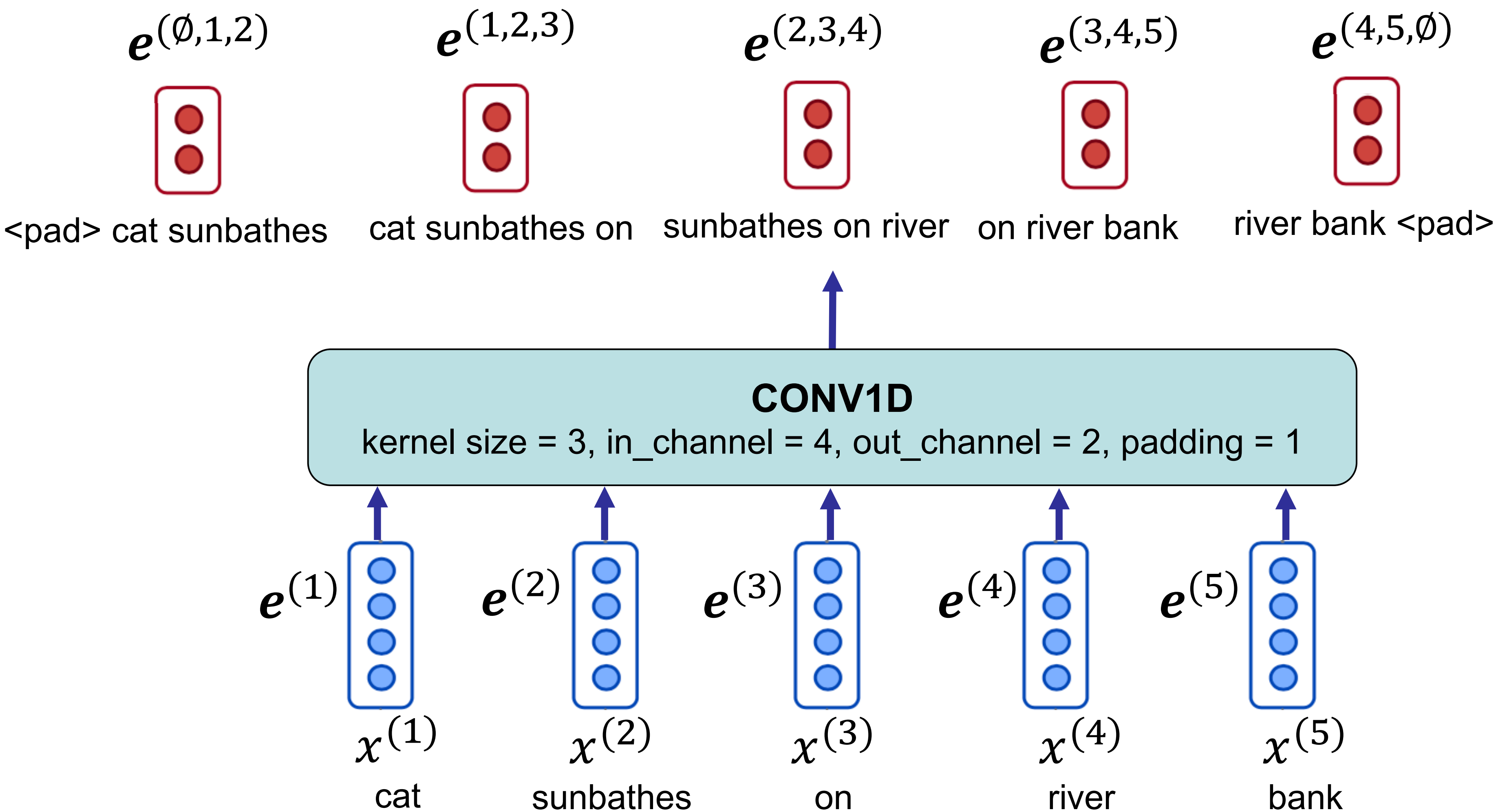
Other notions

- Padding:
  - adds zero vectors to the beginning and end of the sequence

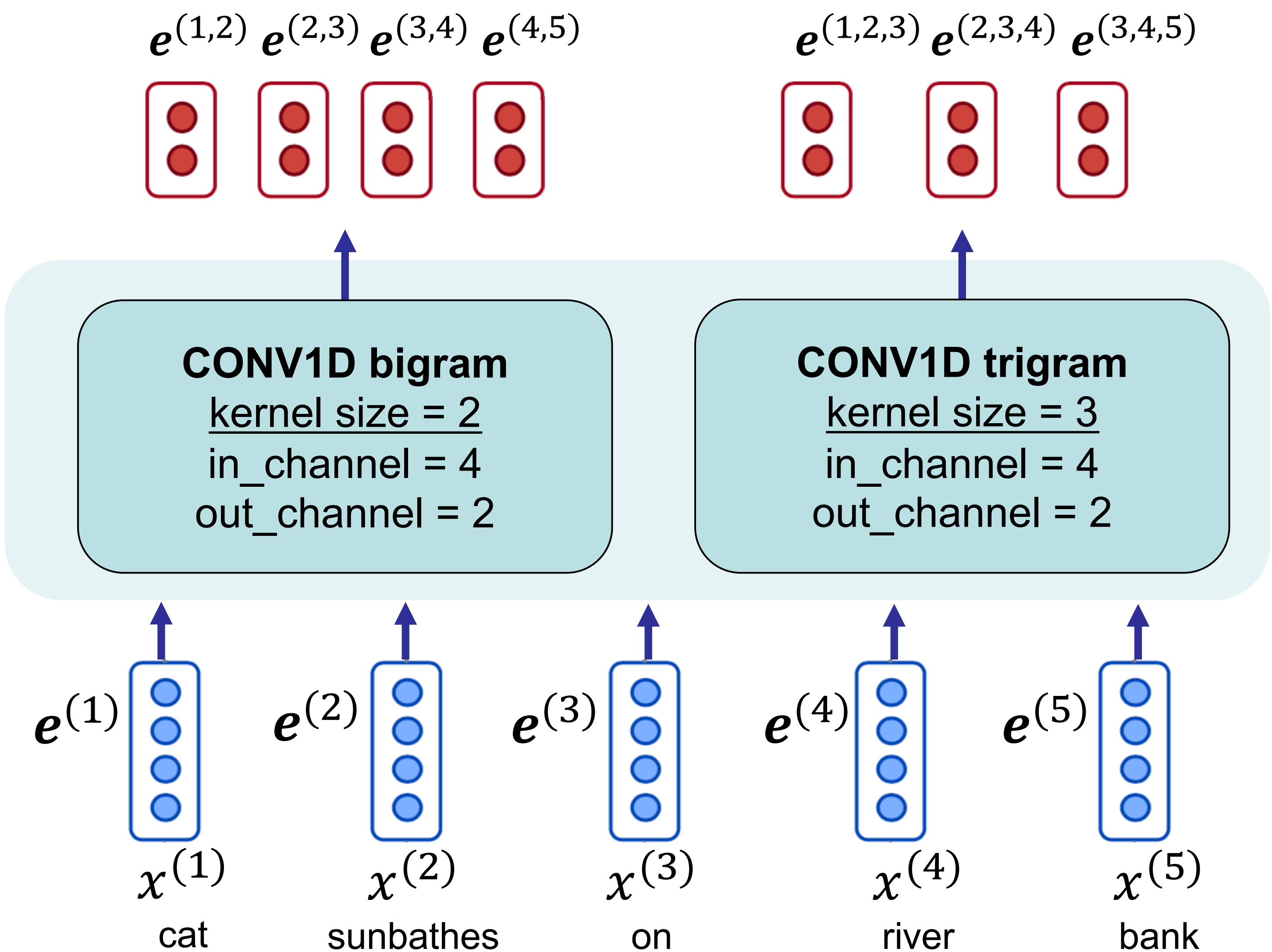
- Stride:
  - The length of the steps over the sequence on which the convolutions are applied
  - Default is 1

More notions with graphic:
https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md

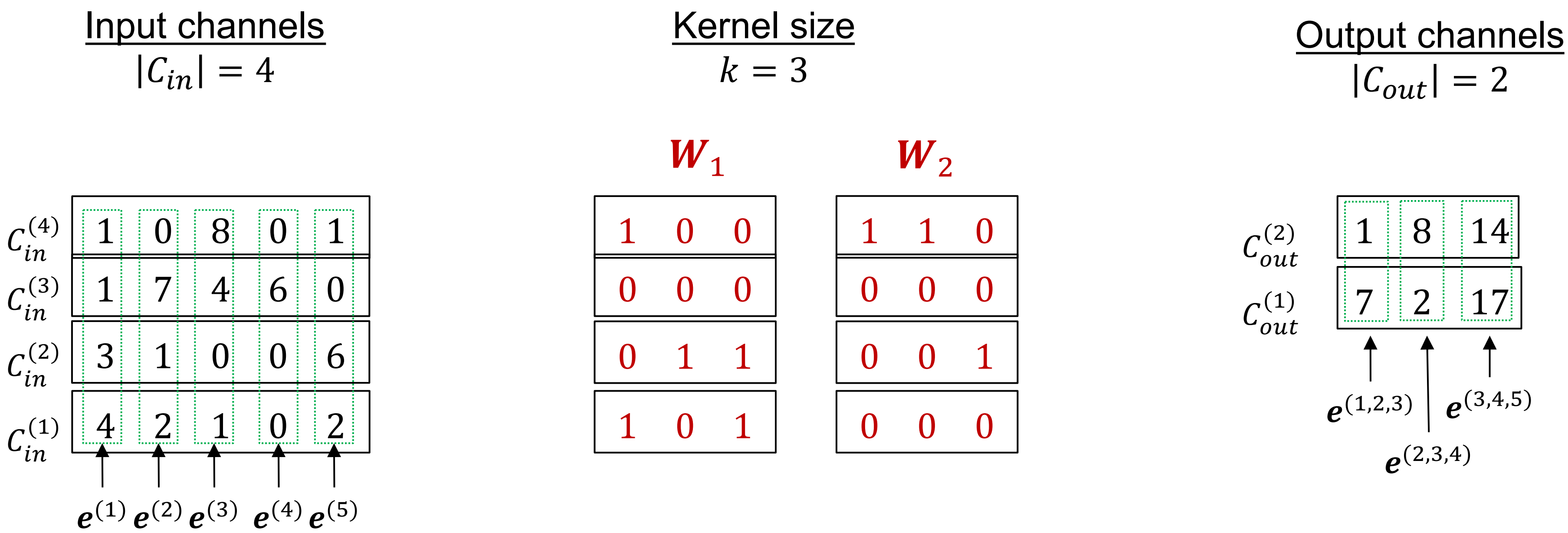
N-gram embeddings



N-gram embeddings



1-dimensional CNN in NLP



Informal formulation of the calculation in Conv1D:

$$e_i^{(x,\ldots,x+k)} = \mathrm{torch.sum}\left(\left[\boldsymbol{e}^{(x)}\,;\ldots\,;\boldsymbol{e}^{(x+k)}\right] \odot \boldsymbol{W}_i\right) + b_i$$

Position ith of the output embedding corresponding to inputs x till $x + k$

Input embedding x

Element-wise multiplication

Bias term of the ith output channel

CNN – summary

- A model to capture patterns in local proximities, learnt through many (linear) kernels
  - Output embeddings are position-invariant

- In comparison with fully connected multi-layer perceptron, CNNs are highly parameter efficient

- NLP mostly uses Conv1D
  - in_channels is the dimension of input embeddings
  - out_channels is the dimension of output embeddings
  - kernel_size is the length of *n*-gram

CONV1D

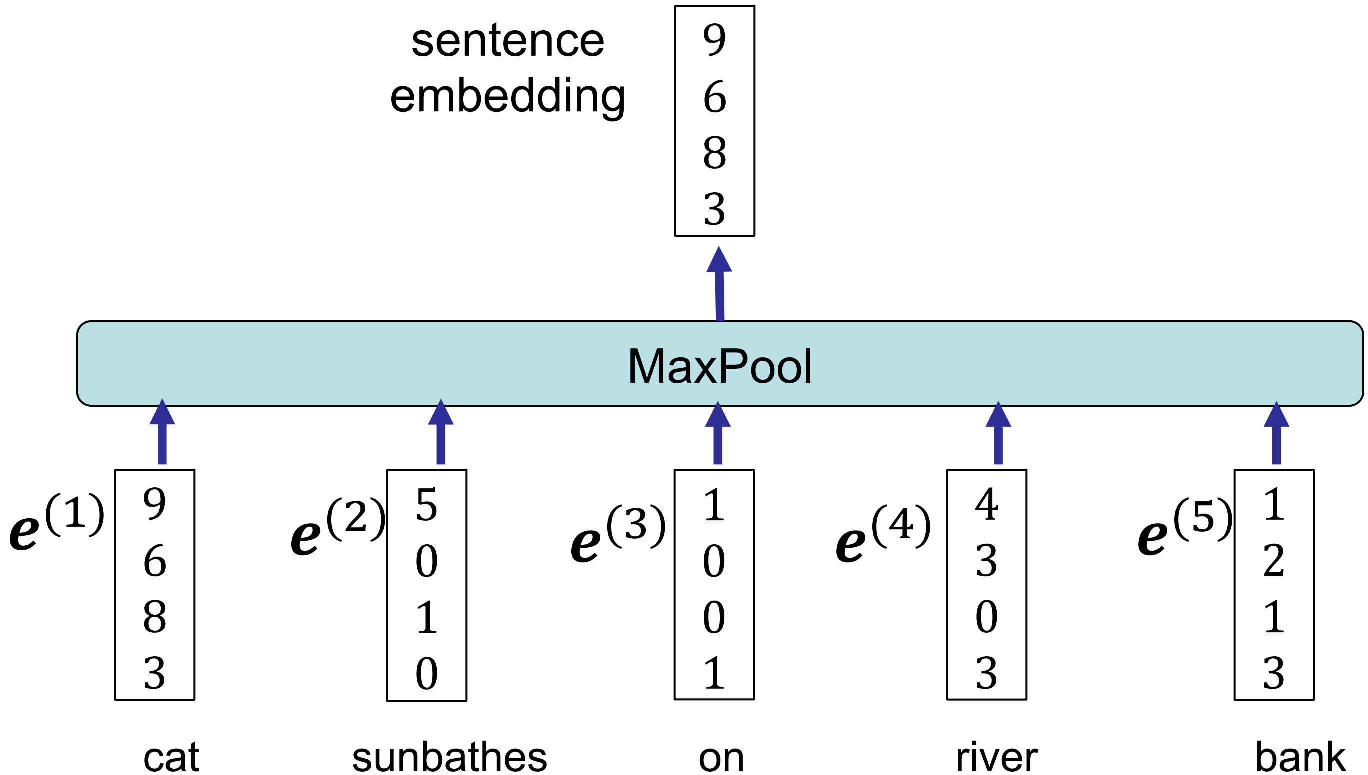
CLASS `torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Agenda

- *N*-Gram Embeddings with CNN
- **CNN in practice**
  - **Document classification**
  - **From characters to word embedding**
  - **CNN in information retrieval models**

Sentence embedding from word embeddings – recap

- Pooling: element-wise operation on input vectors resulting to an output vector
- MaxPool: element-wise maximum of inputs

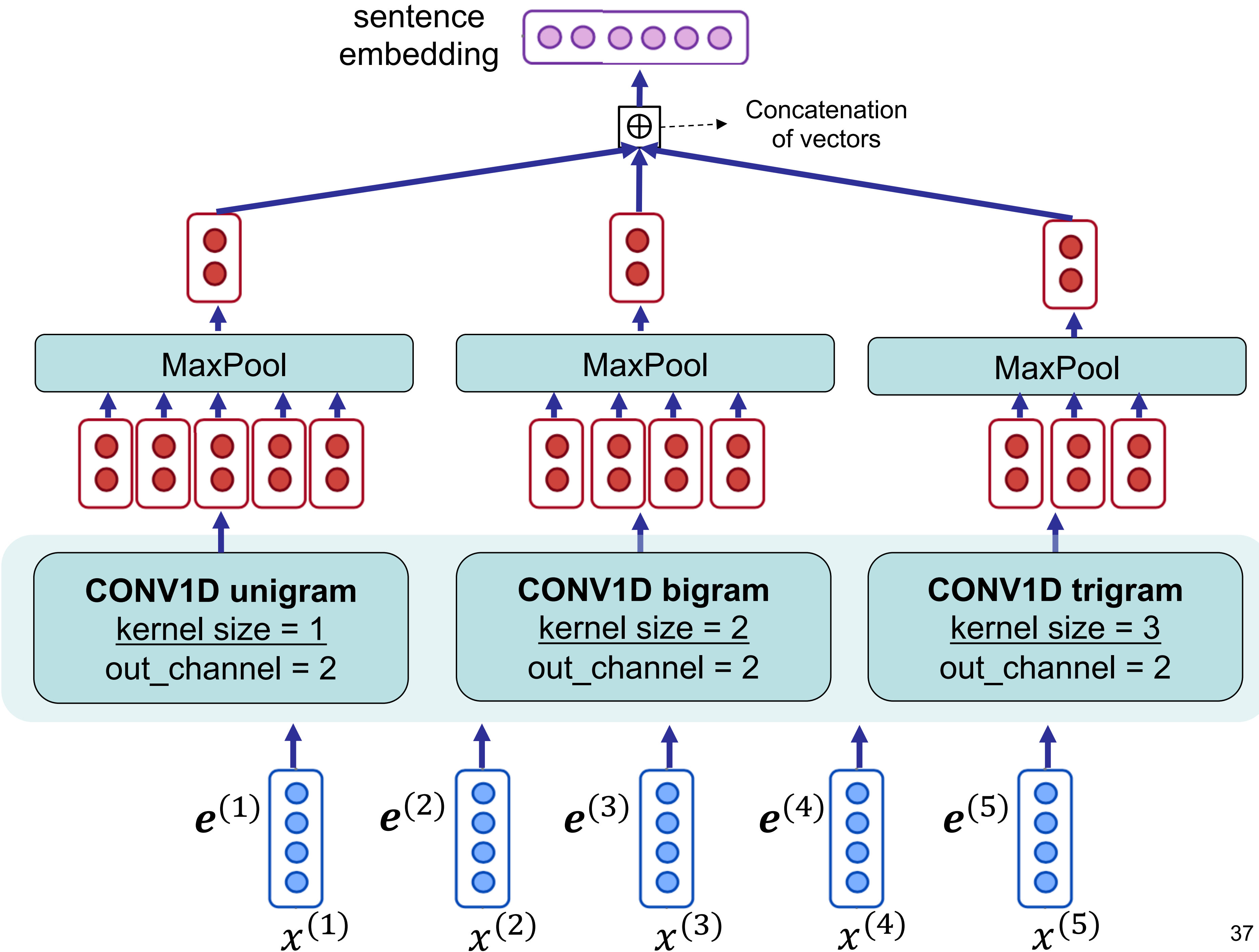


Document classification with CNNs

Steps:

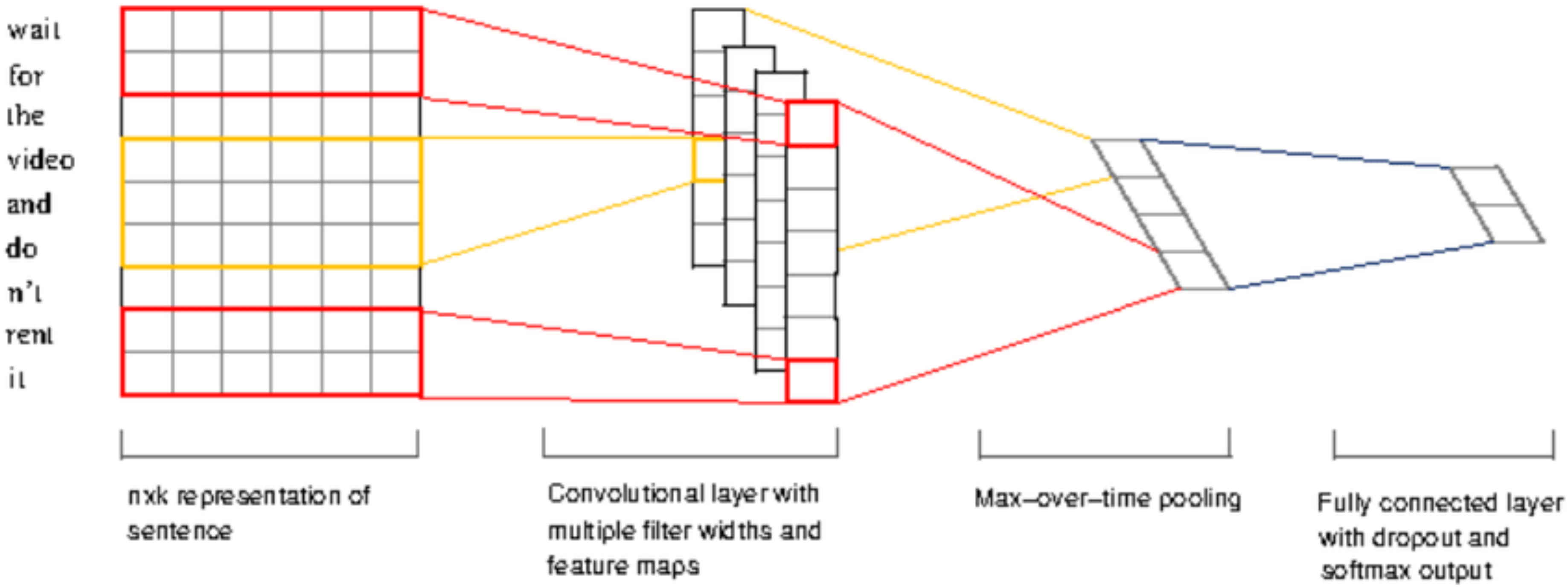
1. Create unigram, bigram, trigram, etc. embeddings
2. Apply pooling to merge embeddings of each *n*-gram over whole the sequence, resulting in several *n*-gram features
3. Concatenate *n*-gram features as the final document feature (document embedding)

Convolutional Neural Networks for Sentence Classification
Y. Kim. Proceedings of the Conference on Empirical Methods in Natural Language Processing, (EMNLP 2014)



sentence embedding
Concatenation of vectors
MaxPool
MaxPool
MaxPool
CONV1D unigram
kernel size = 1
out_channel = 2
CONV1D bigram
kernel size = 2
out_channel = 2
CONV1D trigram
kernel size = 3
out_channel = 2
$e^{(1)}$ $e^{(2)}$ $e^{(3)}$ $e^{(4)}$ $e^{(5)}$
$x^{(1)}$ $x^{(2)}$ $x^{(3)}$ $x^{(4)}$ $x^{(5)}$

Another view of the same model



Source: https://arxiv.org/abs/1806.08760

Why unigram embeddings?

What do we create unigram embeddings ($k = 1$)? … can't we just use the original word embeddings?

Yes, we can, but …

- Unigram CNN adds an extra neural network layer with very few additional parameters
- CNN with $k = 1$ applies the same parameters to all word embeddings (position invariant)
  - Unlike fully connected a feed forward layer which is position variant and adds a lot more parameters

Composing word embeddings from character embeddings

- Instead of predefined word vectors (static word embeddings), compose the embedding of a word from the embeddings of its characters
  1. Define one vector for every character
     - The embedding matrix will be much smaller in comparison with the ones of word embeddings
  2. Use CNNs to create a word embedding from its character embeddings
     - In the same way that we created a document embedding from word embeddings
     - Each CNN results in a character *n*-gram embedding

Word embeddings from character embeddings
Task: Language modeling

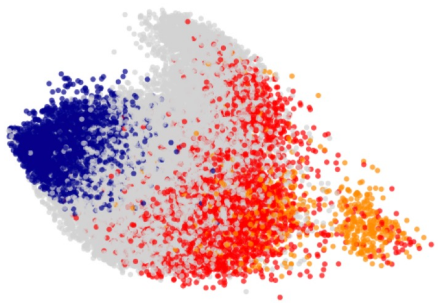
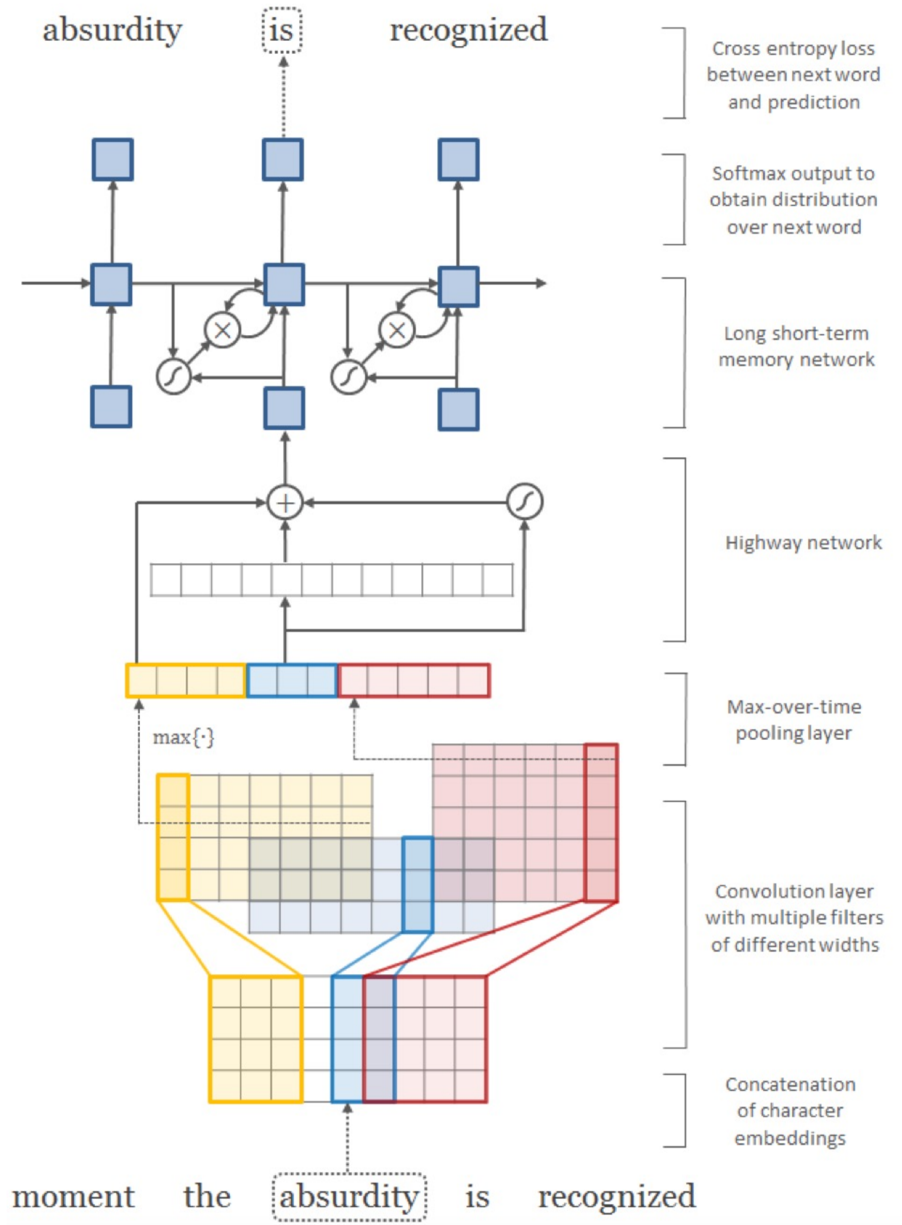
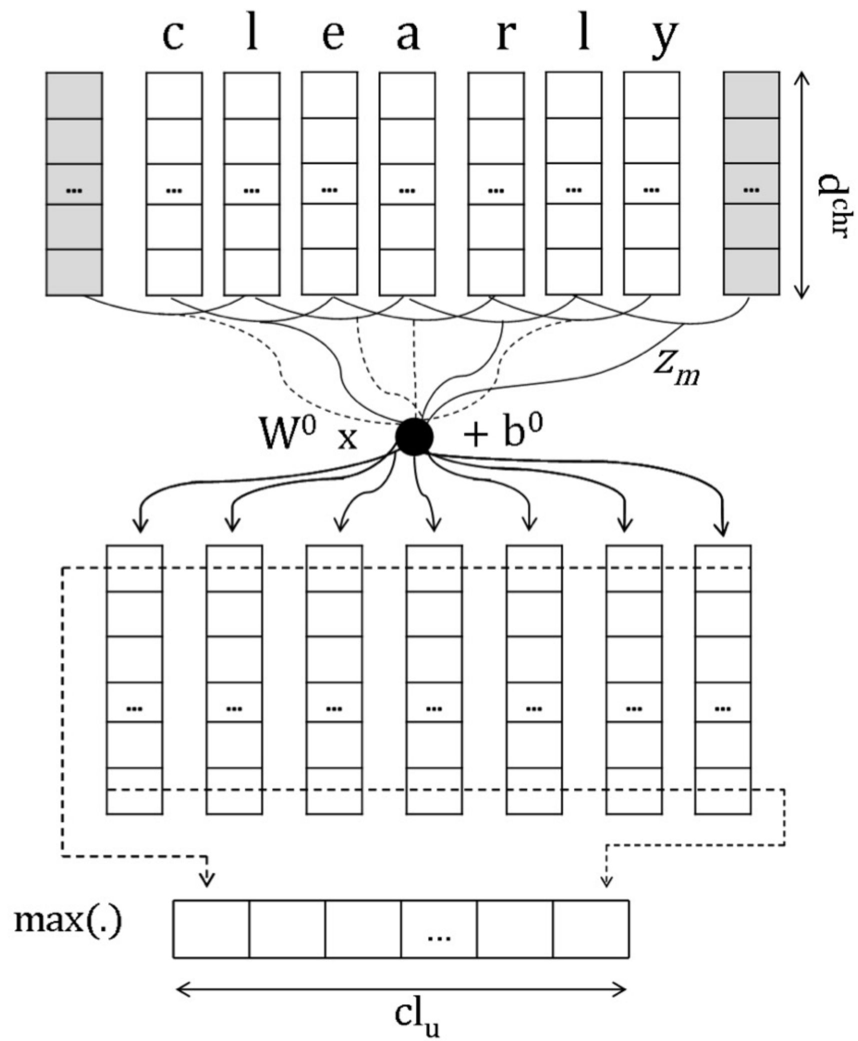


Figure 2: Plot of character n-gram representations via PCA for English. Colors correspond to: prefixes (red), suffixes (blue), hyphenated (orange), and all others (grey). Prefixes refer to character n-grams which start with the start-of-word character. Suffixes likewise refer to character n-grams which end with the end-of-word character.



Kim, Y., Jernite, Y., Sontag, D., & Rush, A. (2016, March). Character-aware neural language models. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 30, No. 1).

Word embeddings from character embeddings
Task: part-of-speech tagging



Dos Santos, C., & Zadrozny, B. (2014, June). Learning character-level representations for part-of-speech tagging. In *International Conference on Machine Learning* (pp. 1818-1826). PMLR.

Kim, Y., Jernite, Y., Sontag, D., & Rush, A. (2016, March). Character-aware neural language models. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 30, No. 1).

CNN word embeddings from character embeddings

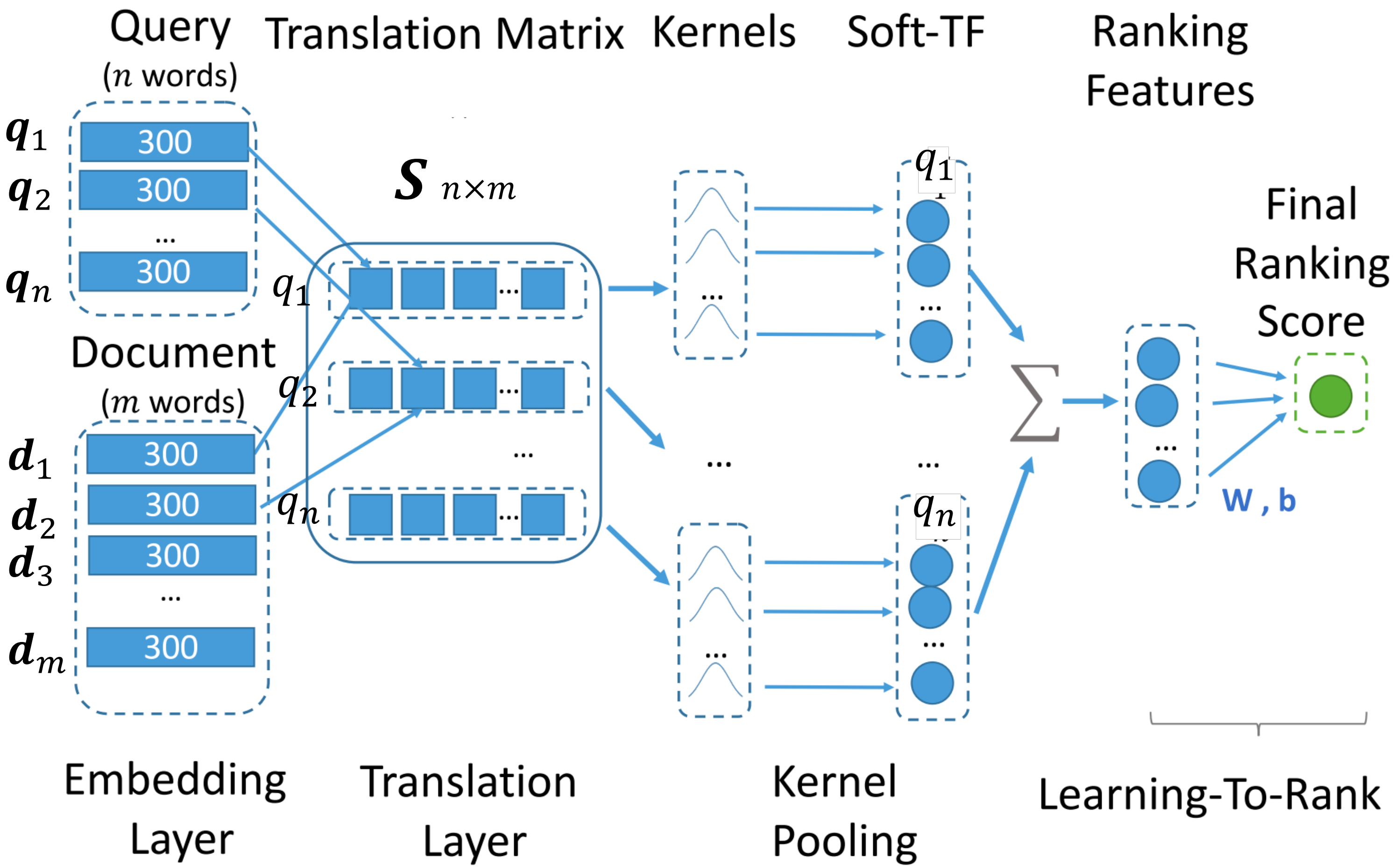
Pros:

- Overall, less parameters in comparison with static word embeddings
- This method resolves the difficulties of handling out-of-vocabularies (OOV)
- Semantic and syntactic regularities are transferred across words, which can benefit some words by providing better generalization

Cons:

- Achieving word embeddings require some computation (feedforward through the CNNs)
- Since every word is composed solely from character embeddings, the quality of some word embeddings might not be as good as static word embeddings

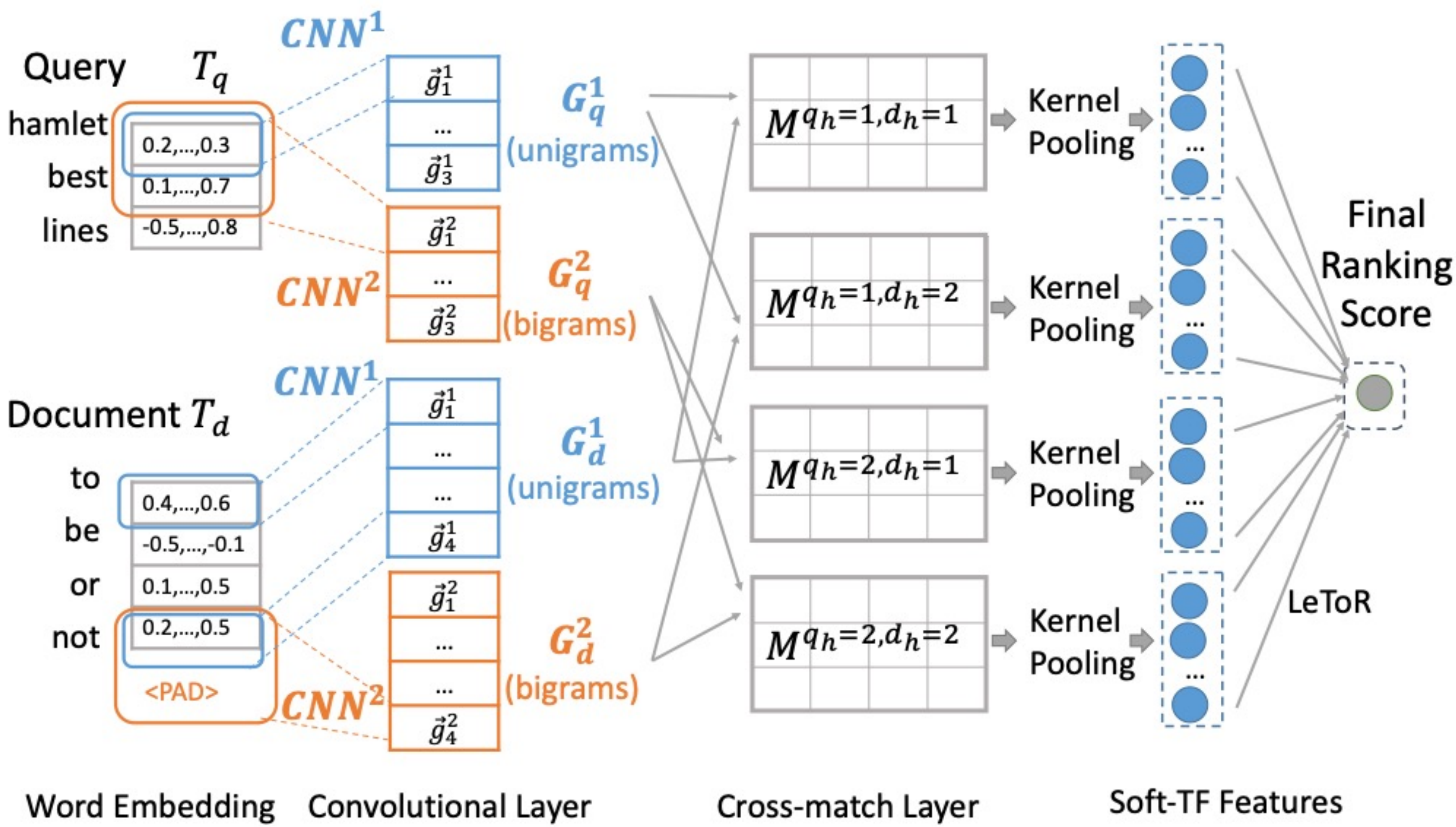
A neural information retrieval model



For details look at Natural Language Processing course - Lecture 6: Information Retrieval with Neural Networks: https://www.jku.at/en/institute-of-computational-perception/teaching/alle-lehrveranstaltungen/natural-language-processing/

Reference: Xiong, C., Dai, Z., Callan, J., Liu, Z., & Power, R. (2017). End-to-end neural ad-hoc ranking with kernel pooling. In *Proceedings of the 40th International ACM SIGIR conference on research and development in information retrieval*

The same model enhanced with *n*-gram embeddings



Dai, Z., Xiong, C., Callan, J., & Liu, Z. (2018). Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *Proceedings of the eleventh ACM international conference on web search and data mining*