uc3m | Universidad Carlos III de Madrid

OpenCourseWare

Procesamiento de Lenguaje Natural con Aprendizaje Profundo,

Máster en Ciencia y Tecnología Informática

Tema 2: Redes convolucionales para Procesamiento de Lenguaje Natural

Isabel Segura Bedmar

Objetivos

- Conocer los fundamentos de las redes convolucionales.
- Comprender las operaciones de convolución y pooling.
- Saber cómo aplicar redes convolucionales pueden ser aplicadas a tareas de PLN.

Índice

- Redes convolucionales.
- Operación de convolución.
- Operación de pooling.
- Redes convolucionales para PLN.
- Resumen

Redes convolucionales

- En inglés, Convolutional Neural Network (CNN)
- Propuestas originariamente en sistemas de clasificación de imágenes.
- Principal idea: algunas partes más pequeñas (filtros) de una imagen son suficientes para identificar el objeto de la imagen.

Redes convolucionales

¿Qué nos ayuda a identificar que es un pájaro?

Image by David Mark from Pixabay

Redes convolucionales

¿Qué nos ayuda a identificar que es un pájaro?

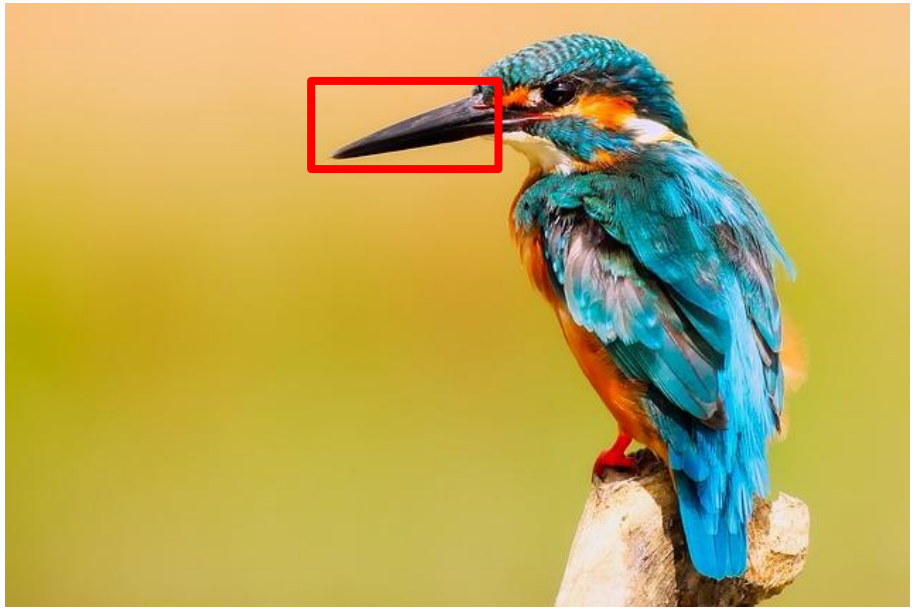


Image by David Mark from Pixabay

Redes convolucionales

Estos filtros (o patrones) pueden tener diferentes tamaños y diferentes localizaciones. ¿Cómo podemos encontrarlos?

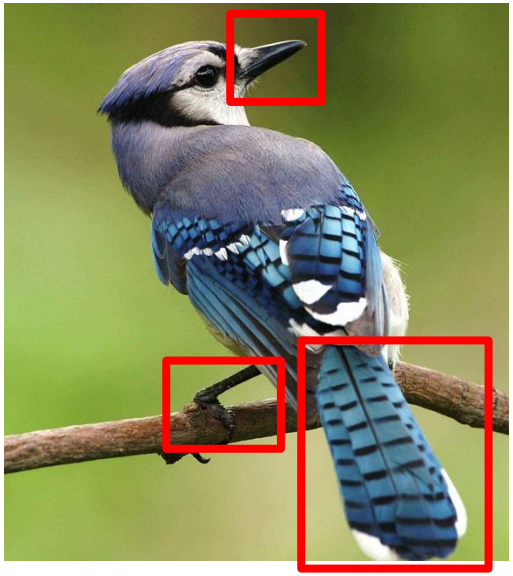


Image by PublicDomainImages from Pixabay

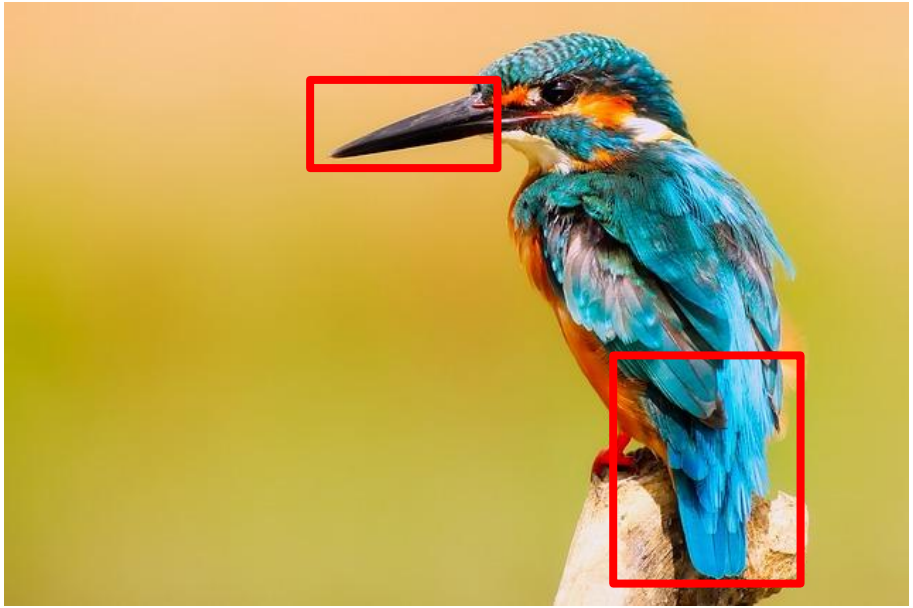


Image by David Mark from Pixabay

Redes convolucionales

Idea:

- Crear muchos filtros de diferentes tamaños
- Mover los filtros por toda la imagen.

Durante el entrenamiento, la red aprenderá a detectar qué filtros aportan más información para clasificar la imagen:

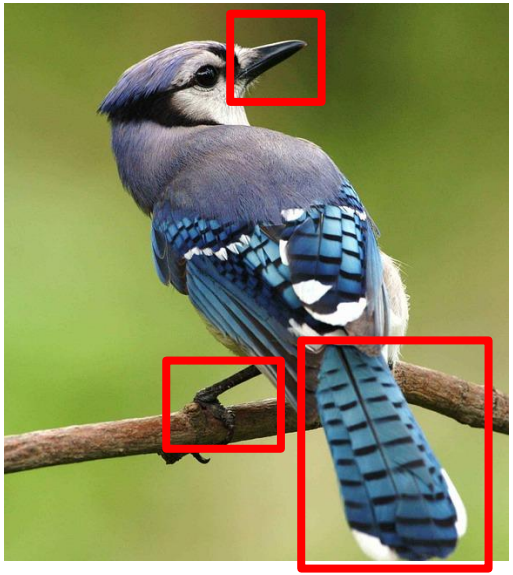


Image by PublicDomainImages from Pixabay

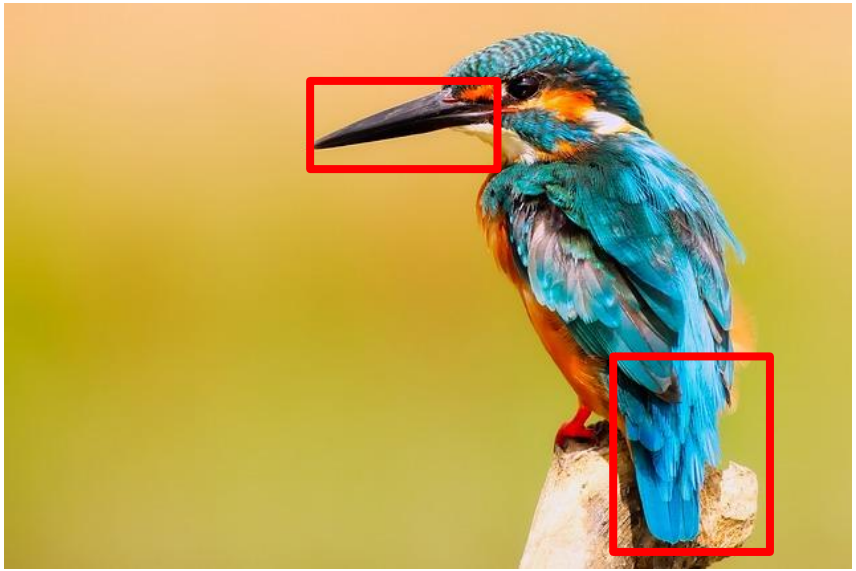


Image by David Mark from Pixabay

Redes convolucionales

Las entradas (textos o imágenes, etc) se representan como matrices.
Las principales operaciones en una red convolucional son:

- convolución.
- pooling.

Operación de convolución

Durante la convolución:

- Se define el número de filtros y su tamaño.
- Cada filtro puede tener un tamaño distinto.
- Los valores de un filtro se inicializan de forma aleatoria, y se ajustan durante el entrenamiento de la red.
- A mayor número de filtros, la red será capaz de aprender mejor, pero aumentará el tiempo de entrenamiento.
- Cada filtro se mueve a lo largo de la matriz de entrada
- en cada movimiento, se calcula un valor aplicando *element-wise multiplication* (ver siguiente página).
- Como salida, cada filtro produce una nueva submatriz (*feature map*).

Operación de convolución

1	0	0	-1
0	3	2	0
4	1	0	2
-2	0	3	1
0	0	1	0

Matriz (entrada)

1	1
1	0

Filtro 2x2

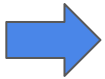
Los valores del filtro se inicializan aleatoriamente.
El filtro podría tener cualquier tamaño: 1x4, 2x3, 3x2, etc, (respetando la dimensión de la matriz de entrada).

Operación de convolución

1x1	0x1	0	-1
0x1	3x0	2	0
4	1	0	2
-2	0	3	1
0	0	1	0

Matriz (entrada)

1	1
1	0



1x1 + 0x1 + 0x1 + 3x0 = 1

element-wise multiplication

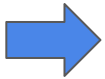
1		

Operación de convolución

1	0x1	1x1	-1
0	3x1	2x0	0
4	1	0	2
-2	0	3	1
0	0	1	0

Matriz (entrada)

1	1
1	0



0x1 + 1x1 + 3x1 + 2x0 = 4

element-wise multiplication

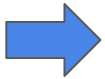
1	**4**	

Operación de convolución

1	0	1x1	-1x1
0	3	2x1	0x0
4	1	0	2
-2	0	3	1
0	0	1	0

Matriz (entrada)

1	1
1	0



1x1 + -1x1 + 2x1 + 0x0 = 2

element-wise multiplication

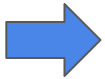
1	4	**2**

Operación de convolución

1	0	1	-1
0x1	3x1	2	0
4x1	1x0	0	2
-2	0	3	1
0	0	1	0

Matriz (entrada)

1	1
1	0



0x1 + 3x1 + 4x1 + 1x0 = 7

element-wise multiplication

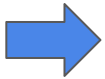
1	4	2
7		

Operación de convolución

1	0	1	-1
0	3x1	2x1	0
4	1x1	0x0	2
-2	0	3	1
0	0	1	0

Matriz (entrada)

1	1
1	0



3x1 + 2x1 + 1x1 + 0x0 = 6

element-wise multiplication

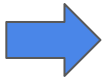
1	4	2
7	**6**	

Operación de convolución

1	0	1	-1
0	3	2x1	0x1
4	1	0x1	2x0
-2	0	3	1
0	0	1	0

Matriz (entrada)

1	1
1	0



2x1 + 0x1 + 0x1 + 2x0 = 2

element-wise multiplication

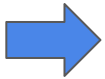
1	4	2
7	6	**2**

Operación de convolución

1	0	1	-1
0	3	2	0
4x1	1x1	0	2
-2x1	0x0	3	1
0	0	1	0

Matriz (entrada)

1	1
1	0



4x1 + 1x1 + -2x1 + 0x0 = 3

element-wise multiplication

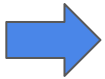
1	4	2
7	6	2
3		

Operación de convolución

1	0	1	-1
0	3	2	0
4	1x1	0x1	2
-2	0x1	3x0	1
0	0	1	0

Matriz (entrada)

1	1
1	0



1x1 + 0x1 + 0x1 + 3x0 = 1

element-wise multiplication

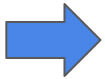
1	4	2
7	6	2
3	**1**	

Operación de convolución

1	0	1	-1
0	3	2	0
4	1	0x1	2x1
-2	0	3x1	1x0
0	0	1	0

Matriz (entrada)

1	1
1	0



0x1 + 2x1 + 3x1 + 1x0 = 5

element-wise multiplication

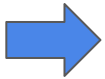
1	4	2
7	6	2
3	1	**5**

Operación de convolución

1	0	1	-1
0	3	2	0
4	1	0	2
-2x1	0x1	3	1
0x1	0x0	1	0

Matriz (entrada)

1	1
1	0



-2x1 + 0x1 + 0x1 + 0x0 = -2

element-wise multiplication

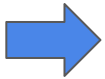
1	4	2
7	6	2
3	1	5
-2		

Operación de convolución

1	0	1	-1
0	3	2	0
4	1	0	2
-2	0x1	3x1	1
0	0x1	1x0	0

Matriz (entrada)

1	1
1	0



0x1 + 3x1 + 0x1 + 1x0 = 3

element-wise multiplication

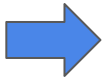
1	4	2
7	6	2
3	1	5
-2	**3**	

Operación de convolución

1	0	1	-1
0	3	2	0
4	1	0	2
-2	0	3x1	1x1
0	0	1x1	0x0

Matriz (entrada)

1	1
1	0



3x1 + 1x1 + 1x1 + 0x0 = 5

element-wise multiplication

1	4	2
7	6	2
3	1	5
-2	3	**5**

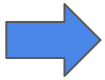
Cada filtro produce como salida una nueva representación de la entrada (**feature map**)

Operación de convolución

1	0	1	-1
0	3	2	0
4	1	0	2
-2	0	3	1
0	0	1	0

Matriz (entrada)

1	1
1	0



1	4	2
7	6	2
3	1	5
-2	3	**5**

En el anterior ejemplo, el filtro se ha movido fila por fila y columna por columna (stride = 1).
Sin embargo, es posible definir parámetros para que el desplazamiento sea diferente:

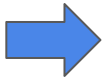
Operación de convolución

1	0	1	-1
0	3	2	0
4	1	0	2
-2	0	3	1
0	0	1	0

Matriz (entrada)

Filtro 1 (2x2)

1	1
1	0



Salida (feature map) del filtro 1: 4x3

1	4	2
7	6	2
3	1	5
-2	3	**5**

Filtros de tamaños distintos, producen submatrices (feature map) también con tamaños distintos.

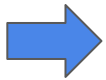
Operación de convolución

1	0	1	-1
0	3	2	0
4	1	0	2
-2	0	3	1
0	0	1	0

Matriz (entrada)

Filtro 2 (1x4)

1	0	0	1



Salida (feature map) del Filtro 2: 5x1

0
0
6
-1
0

Filtros de tamaños distintos, producen submatrices (feature map) también con tamaños distintos.

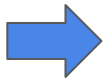
Operación de convolución

1	0	1	-1
0	3	2	0
4	1	0	2
-2	0	3	1
0	0	1	0

Matriz (entrada)

Filtro 3: 5x3

1	0	0
0	0	1
1	0	0
-1	1	1
0	1	0



Salida (feature map) del Filtro 3: 1x2

12	6

Filtros de tamaños distintos, producen submatrices (feature map) también con tamaños distintos.

Operación de convolución

El tamaño de la nueva submatriz será:

output_size = (input_size - filter_size + 2 * padding) / stride + 1

donde:

- input_size: tamaño de la entrada (que puede ser la matriz de entrada o también un submatriz (feature map) producida por una capa anterior de convolución).
- filter_size: tamaño del filtro
- padding: the number of zeros added to the border of the input image to preserve its size
- stride: the number of pixels by which the filter moves across the image

Operación de pooling.

- Consiste en reducir el tamaño de las representaciones (feature map) obtenidos por los filtros

Operación de pooling. Por qué?

- Si transformamos la imagen de entrada en una imagen más pequeña, esto no afectará al objeto representado en la imagen.

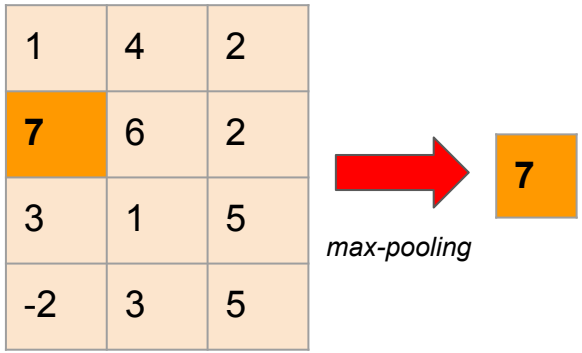
Image by PublicDomainImages from Pixabay

Image by PublicDomainImages from Pixabay

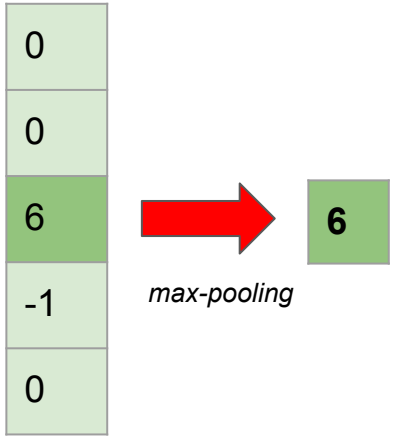
- Por tanto, usamos menos parámetros para representar la entrada.

Tipos de pooling

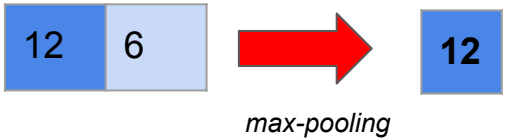
- **max-pooling**: de la salida de cada filtro, se selecciona el elemento mayor.



Filtro 1
(feature map)



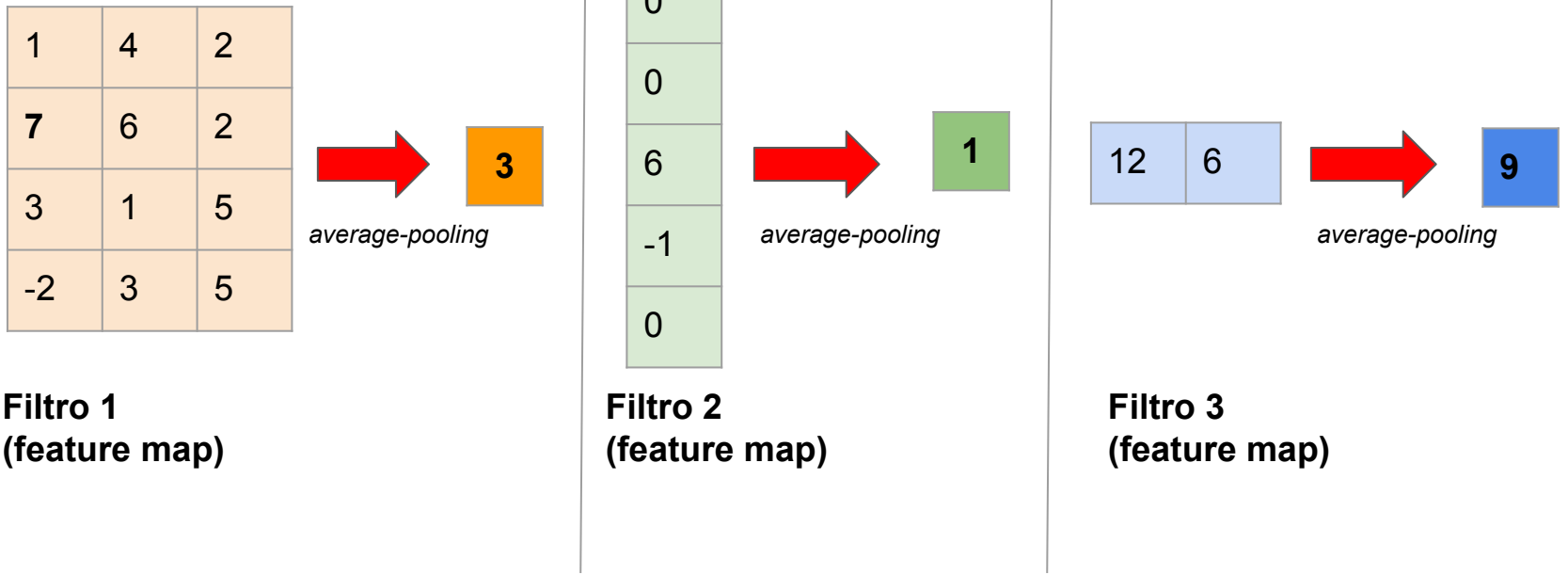
Filtro 2
(feature map)



Filtro 3
(feature map)

Tipos de pooling

- **average-pooling**: de la salida de cada filtro, se calcula la media de sus elementos.



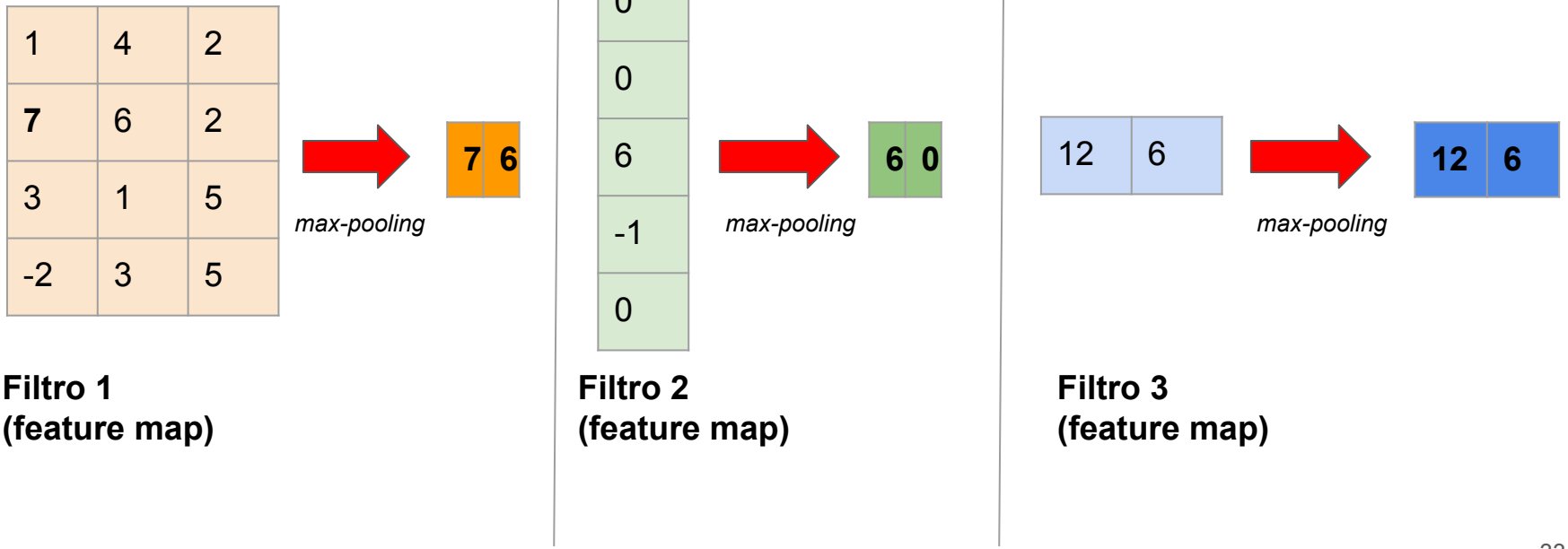
Filtro 1 (feature map)

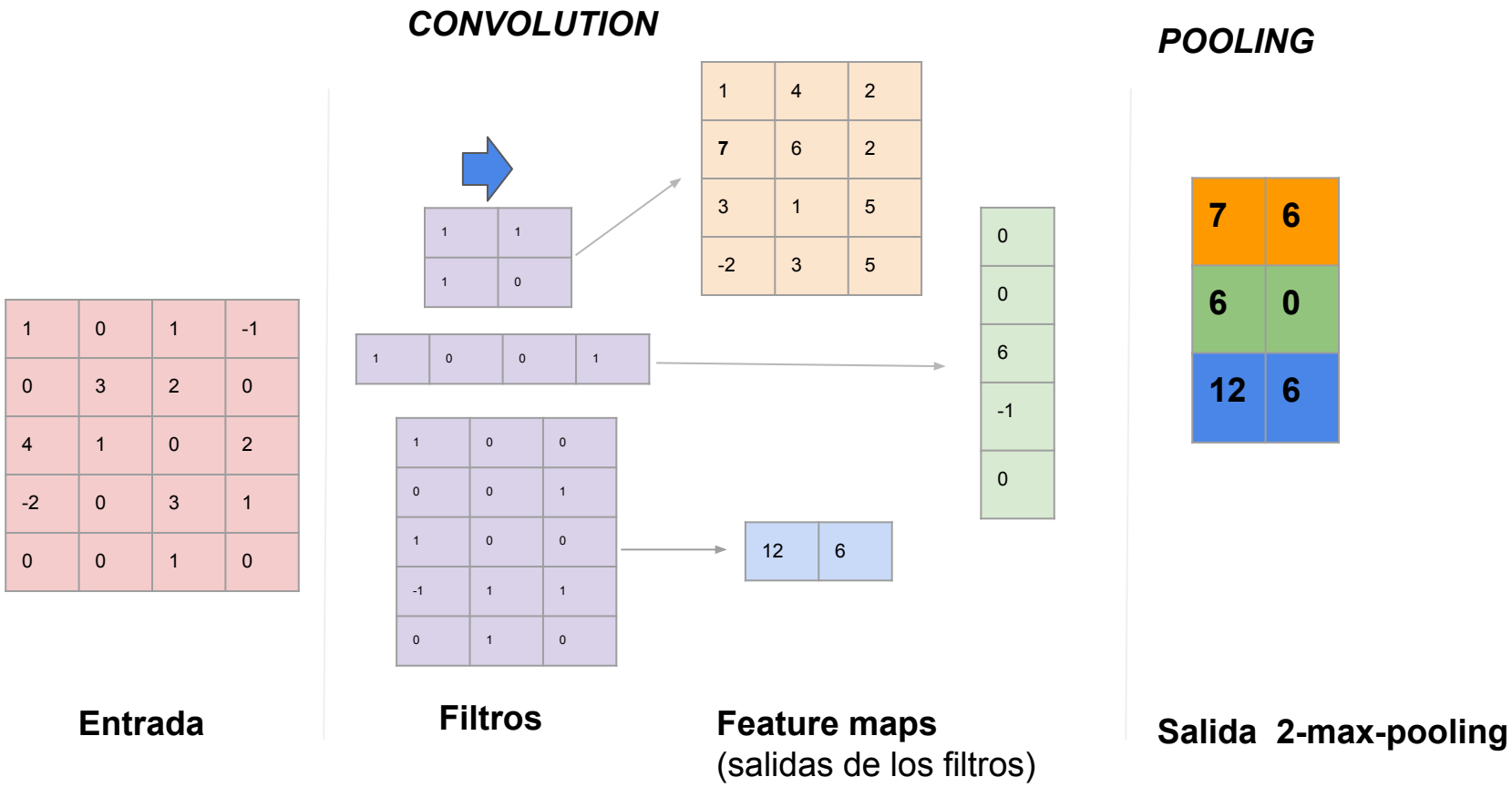
Filtro 2 (feature map)

Filtro 3 (feature map)

Tipos de pooling

- **k-max-pooling**: de la salida de cada filtro, se selecciona los k elementos de mayor valor. Por ejemplo k=2




CONVOLUTION
POOLING
1 0 1 -1
0 3 2 0
4 1 0 2
-2 0 3 1
0 0 1 0
1 1
1 0
1 4 2
7 6 2
3 1 5
-2 3 5
1 0 0 1
1 0 0
0 0 1
1 0 0
-1 1 1
0 1 0
12 6
0
0
6
-1
0
7 6
6 0
12 6
Entrada
Filtros
Feature maps
(salidas de los filtros)
Salida 2-max-pooling

¿Las redes convolucionales pueden ser útiles para tareas de PLN?

Redes convolucionales para PLN

Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only hate half of it but will definitely be buying again!!!

Imagen de Дарья Яковлева en Pixabay

¿Es una opinión positiva?, ¿por qué?

Redes convolucionales para PLN

Amazing! *This box of cereal gave me a* ***perfectly balanced breakfast****, as all things should be. I only hate half of it but* ***will definitely be buying again****!!!*

Imagen de Дарья Яковлева en Pixabay

Redes convolucionales para PLN

- En una imagen, siempre era posible encontrar pequeños filtros que nos ayudan a clasificar o identificar un objeto.
- De forma similar, en el texto también hay partes más pequeñas que nos van a permitir clasificarlo.
- Por tanto, CNN puede ser útil para tareas de NLP, en especial, para tareas de clasificación de textos.

Cómo usar CNN para PLN

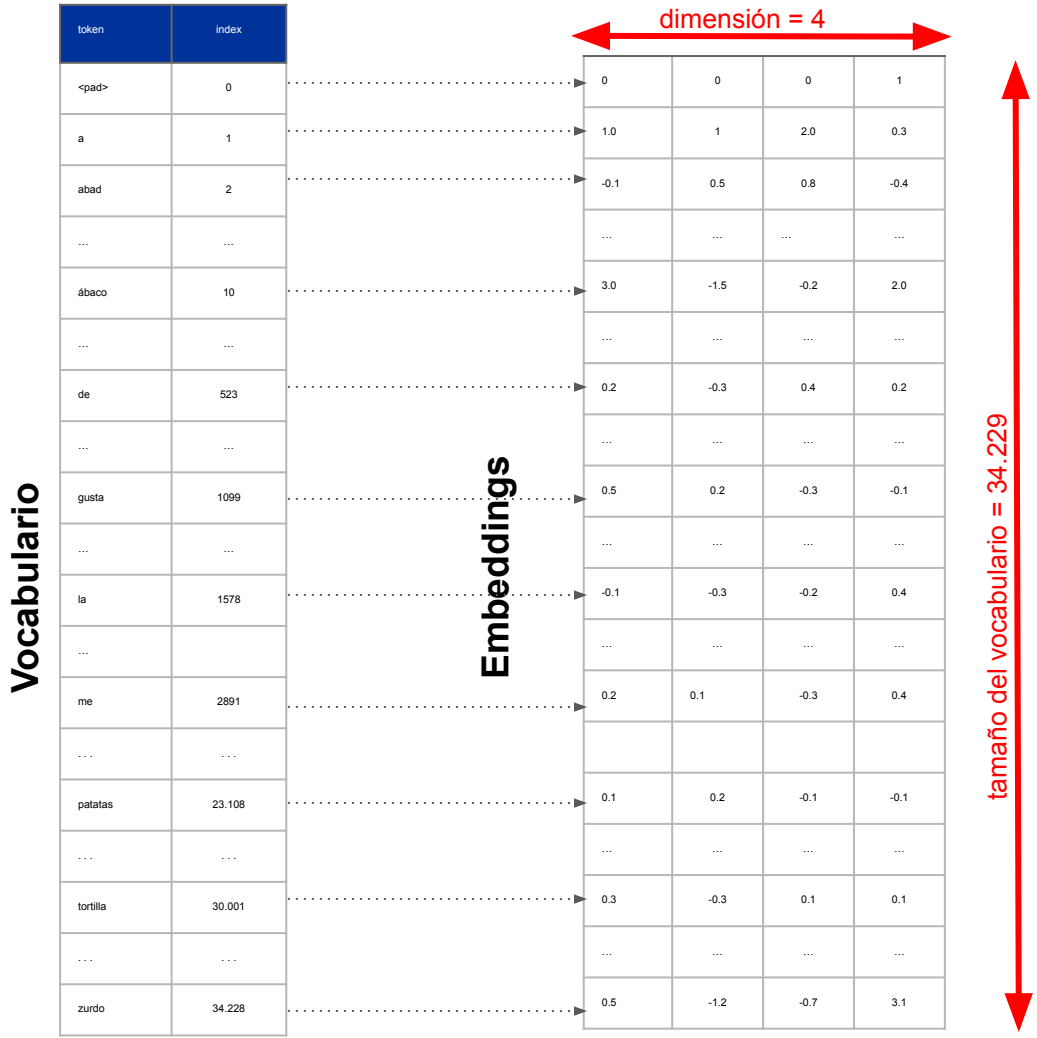
- Primero debemos entrenar un tokenizador con la colección de textos de entrenamiento.
- El **tokenizador** va a aprender un **vocabulario**, formado por todos los **tokens distintos** de la colección de textos.
- Cada **token** está asociado con un **índice** (un valor entero).
- El token **<pad>** también se guarda en dicho vocabulario y se le reserva el índice **0**.

Vocabulario

token	index
<pad>	0
a	1
abad	2
...	...
ábaco	10
...	...
de	523
...	...
gusta	1099
...	...
la	1578
...	
me	2891
. . .	. . .
patatas	23.108
. . .	. . .
tortilla	30.001
. . .	. . .
zurdo	34.228

Cómo usar CNN para PLN

- Cada **token (índice)** del vocabulario va a ser asociado con un **vector** de números reales.
- Todos los **vectores** tienen la **misma dimensión** (normalmente > 50). Una dimensión estándar suele ser 300.
- ¿Cómo se **inicializan los vectores**?. Dos opciones:
  - i) **modelo de word embeddings** (tema 4). En la arquitectura, podemos decir si los vectores deben ser ajustados durante el entrenamiento (non-static) o no (static).
  - ii) **inicialización aleatoria**. En este caso, los vectores sí son ajustados durante el entrenamiento.
- En general, la inicialización basada en modelos de word embeddings suelen dar mejores resultados (estos vectores ya codifican conocimiento de los tokens.)



token	index
<pad>	0
a	1
abad	2
...	...
ábaco	10
...	...
de	523
...	...
gusta	1099
...	...
la	1578
...	
me	2891
. . .	. . .
patatas	23.108
. . .	. . .
tortilla	30.001
. . .	. . .
zurdo	34.228

0	0	0	1
1.0	1	2.0	0.3
-0.1	0.5	0.8	-0.4
...	...	...	...
3.0	-1.5	-0.2	2.0
...	...	...	...
0.2	-0.3	0.4	0.2
...	...	...	...
0.5	0.2	-0.3	-0.1
...	...	...	...
-0.1	-0.3	-0.2	0.4
...	...	...	...
0.2	0.1	-0.3	0.4
0.1	0.2	-0.1	-0.1
...	...	...	...
0.3	-0.3	0.1	0.1
...	...	...	...
0.5	-1.2	-0.7	3.1

Notas:

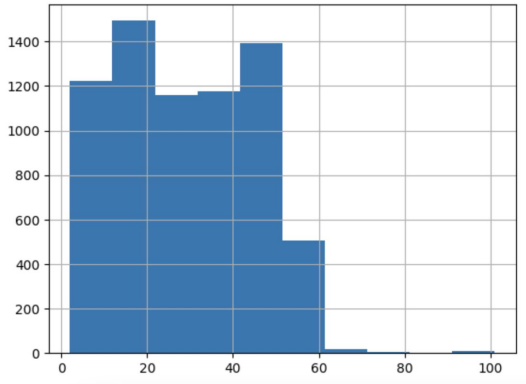
- Aunque lo normal es usar una dimensión > 50, en este ejemplo, suponemos que la dimensión es 4.
- Recuerda que estos vectores pueden ser o bien i) tomados de un modelo de word embeddings (tema 4), o bien ii) inicializados aleatoriamente.

Cómo usar CNN para PLN

- Para entrenar el modelo, usaremos una colección de textos de entrenamiento (que se ha para crear el vocabulario del tokenizador).
- Cada texto (junto con su etiqueta, si es una tarea de clasificación) será la entrada de la red.
- La red necesita que todas las entradas tenga el mismo tamaño (mismo número de tokens).
- Es necesario **elegir un tamaño** que sea representativo de los textos de la colección de entrenamiento. Para ello se recomienda estudiar la distribución de los textos.

Cómo usar CNN para PLN

- Por ejemplo, dada la siguiente distribución, se puede observar que aunque el **tamaño máximo** es de **101 tokens**, la **mayoría** de los **textos** tienen un tamaño **menor o igual** a 60 tokens.
- Si elegimos el tamaño de 101 tokens, estaremos desperdiciando mucha memoria para representar los textos.
- En este ejemplo, una buena elección podría ser 50 (percentil 90%) porque el 90% de los textos tienen un tamaño menor o igual a 50 tokens.



count	6977.000000
mean	28.870861
std	15.884901
min	2.000000
25%	15.000000
50%	28.000000
75%	43.000000
90%	50.000000
95%	53.000000
99%	58.000000
max	101.000000

Cómo usar CNN para PLN

- Una vez estimado el tamaño de las entradas (vamos a llamar **LENGTH**), **cada texto** debe ser **tokenizado**.
- El **tokenizador** se encargará de:
  - **dividir** el **texto** en **tokens**,
  - aplicar **padding** o **truncation** sobre el texto tokenizado para asegurar que tiene el tamaño de entrada fijado.
    - **padding**: consiste en añadir el token <pad> (índice 0) tantas veces como sea necesario para que el texto tenga un tamaño de **LENGTH** tokens. Este proceso puede aplicarse al principio, final o en ambos extremos de un texto.
    - **truncation**: si el texto tiene un tamaño que **LENGTH** tokens, el tokenizador eliminará tokens del texto, hasta conseguir que el texto tokenizado tiene un tamaño de **LENGTH** tokens. Este proceso puede aplicarse al principio, final o en ambos extremos de un texto.
  - **asociar** cada **token** con su **índice** del vocabulario.

Cómo usar CNN para PLN

- Por ejemplo, vamos a suponer que **LENGTH** = 8, y que los procesos de padding y truncation se aplican en ambos lados del texto (principio y final).
- Dado el siguiente texto "*Me gusta la tortilla de patatas*", la salida del tokenizador sería:

<pad>	0
Me	2891
gusta	1099
la	1578
tortilla	30001
de	523
patatas	23108
<pad>	0

Nota: comprueba que los índices se corresponden con los índices definidos en la tabla en esta diapositiva

Cómo usar CNN para PLN

- Por último, cada token es representado con su vector de la matriz de embeddings.

Nota: comprueba que los vectores con los definidos para esos tokens en la diapositiva diapositiva

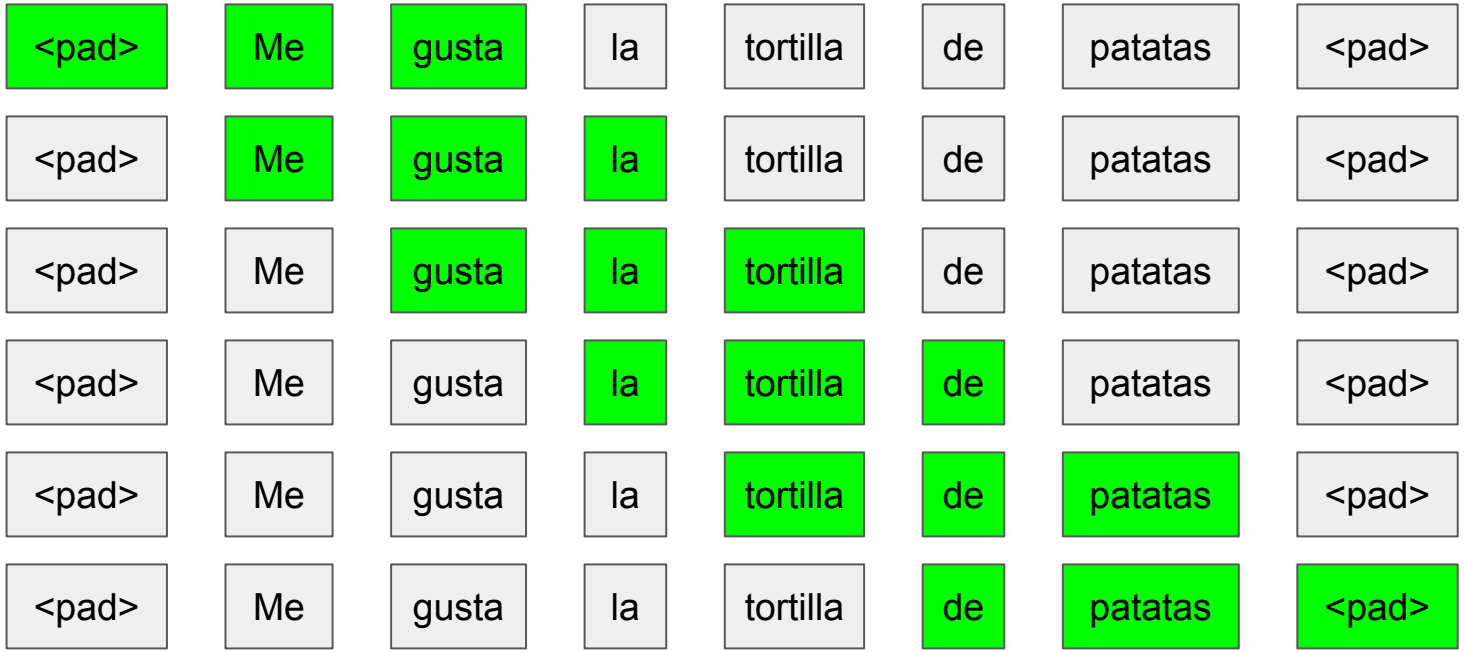
<pad>	0	0	0	1
Me	0.2	0.1	-0.3	0.4
gusta	0.5	0.2	-0.3	-0.1
la	-0.1	-0.3	-0.2	0.4
tortilla	0.3	-0.3	0.1	0.1
de	0.2	-0.3	0.4	0.2
patatas	0.1	0.2	-0.1	-0.1
<pad>	0	0	0	1

Cómo usar CNN para PLN

- Mientras que una imagen puede ser representada con dos o más dimensiones, los textos únicamente van a tener una dimensión: el número de tokens.
- En PLN, el **filtro,** que se aplica en la capa de convolución, se puede ver como una **ventana de tokens**. Su tamaño es el tamaño del filtro.

Cómo usar CNN para PLN

- En el siguiente ejemplo, estamos aplicando un filtro de tamaño 3.



Cómo usar CNN para PLN

- Puedes crear tantos filtros como quieras y de diferentes tamaños (los tamaños más habituales son 2, 3 y 4).
- No existe un número ideal de filtros. Se determina de forma empírica.
- A **mayor número de filtros**, mayor capacidad para aprender, pero **mayor coste computacional**.
- Un ejemplo de número de filtros podría ser: *32 filtros de tamaño 2, 64 filtros de tamaño 3, y 128 filtros de tamaño 4. (no existe un número ideal; depende de la tarea).*
- En PLN, un **filtro de tamaño X**, será una **matriz** de dimensión: **X * dim,** donde dim es la dimensión de los vectores (embeddings).
- Los valores de la matriz son **inicializados aleatoriamente**.
- Cada filtro es aplicado sobre cada uno de los textos tokenizados, que también son matrices de dimensión **LENGTH x dim**, donde LENGTH es el tamaño estimado para los textos de entrada.

Cómo usar CNN para PLN

<pad>	0	0	0	1
Me	0.2	0.1	-0.3	0.4
gusta	0.5	0.2	-0.3	-0.1
la	-0.1	-0.3	-0.2	0.4
tortilla	0.3	-0.3	0.1	0.1
de	0.2	-0.3	0.4	0.2
patatas	0.1	0.2	-0.1	-0.1
<pad>	0	0	0	1

- Operación de convolución. Vamos a aplicar un filtro de tamaño 3, que será una matriz de dimensión 3x4, porque hemos supuesto que la dimensión de los vectores es 4.

3	1	2	-3
-1	2	1	-3
1	1	-1	1

Filtro 1 (tamaño 3)

Cómo usar CNN para PLN

<pad>	0	0	0	1
Me	0.2	0.1	-0.3	0.4
gusta	0.5	0.2	-0.3	-0.1
la	-0.1	-0.3	-0.2	0.4
tortilla	0.3	-0.3	0.1	0.1
de	0.2	-0.3	0.4	0.2
patatas	0.1	0.2	-0.1	-0.1
<pad>	0	0	0	1

- Recuerda que podemos tener varios filtros de tamaños distintos.

3	1	2	-3
-1	2	1	-3
1	1	-1	1

Filtro 1 (tamaño 3)

0	1	0	-1
1	5	0	-2

Filtro 2 (tamaño 2)

-1	1	0	-1
1	0	1	1
2	3	0	0
0	0	1	0

Filtro 3 (tamaño 4)

Cómo usar CNN para PLN

<pad>	0	0	0	1
Me	0.2	0.1	-0.3	0.4
gusta	0.5	0.2	-0.3	-0.1
la	-0.1	-0.3	-0.2	0.4
tortilla	0.3	-0.3	0.1	0.1
de	0.2	-0.3	0.4	0.2
patatas	0.1	0.2	-0.1	-0.1
<pad>	0	0	0	1

Aplicamos el filtro sobre la entrada:

3	1	2	-3
-1	2	1	-3
1	1	-1	1

Filtro 1 (tamaño 3)

<pad>+me+gusta	-3.6
me+gusta+la	
gusta+la+tortilla	
la+tortilla+de	
tortilla+de+patatas	
de+patatas+pad	

Cómo usar CNN para PLN

<pad>	0	0	0	1
Me	0.2	0.1	-0.3	0.4
gusta	0.5	0.2	-0.3	-0.1
la	-0.1	-0.3	-0.2	0.4
tortilla	0.3	-0.3	0.1	0.1
de	0.2	-0.3	0.4	0.2
patatas	0.1	0.2	-0.1	-0.1
<pad>	0	0	0	1

Aplicamos el filtro sobre la entrada:

3	1	2	-3
-1	2	1	-3
1	1	-1	1

Filtro 1 (tamaño 3)

<pad>+me+gusta	-3.6
me+gusta+la	1.4
gusta+la+tortilla	
la+tortilla+de	
tortilla+de+patatas	
de+patatas+pad	

Cómo usar CNN para PLN

<pad>	0	0	0	1
Me	0.2	0.1	-0.3	0.4
gusta	0.5	0.2	-0.3	-0.1
la	-0.1	-0.3	-0.2	0.4
tortilla	0.3	-0.3	0.1	0.1
de	0.2	-0.3	0.4	0.2
patatas	0.1	0.2	-0.1	-0.1
<pad>	0	0	0	1

Aplicamos el filtro sobre la entrada:

3	1	2	-3
-1	2	1	-3
1	1	-1	1

Filtro 1 (tamaño 3)

<pad>+me+gusta	-3.6
me+gusta+la	1.4
gusta+la+tortilla	-0.5
la+tortilla+de	
tortilla+de+patatas	
de+patatas+pad	

Cómo usar CNN para PLN

<pad>	0	0	0	1
Me	0.2	0.1	-0.3	0.4
gusta	0.5	0.2	-0.3	-0.1
la	-0.1	-0.3	-0.2	0.4
tortilla	0.3	-0.3	0.1	0.1
de	0.2	-0.3	0.4	0.2
patatas	0.1	0.2	-0.1	-0.1
<pad>	0	0	0	1

Aplicamos el filtro sobre la entrada:

3	1	2	-3
-1	2	1	-3
1	1	-1	1

Filtro 1 (tamaño 3)

<pad>+me+gusta	-3.6
me+gusta+la	1.4
gusta+la+tortilla	-0.5
la+tortilla+de	-3.6
tortilla+de+patatas	
de+patatas+pad	

Cómo usar CNN para PLN

<pad>	0	0	0	1
Me	0.2	0.1	-0.3	0.4
gusta	0.5	0.2	-0.3	-0.1
la	-0.1	-0.3	-0.2	0.4
tortilla	0.3	-0.3	0.1	0.1
de	0.2	-0.3	0.4	0.2
patatas	0.1	0.2	-0.1	-0.1
<pad>	0	0	0	1

Aplicamos el filtro sobre la entrada:

3	1	2	-3
-1	2	1	-3
1	1	-1	1

Filtro 1 (tamaño 3)

<pad>+me+gusta	-3.6
me+gusta+la	1.4
gusta+la+tortilla	-0.5
la+tortilla+de	-3.6
tortilla+de+patatas	-0.2
de+patatas+pad	

Cómo usar CNN para PLN

<pad>	0	0	0	1
Me	0.2	0.1	-0.3	0.4
gusta	0.5	0.2	-0.3	-0.1
la	-0.1	-0.3	-0.2	0.4
tortilla	0.3	-0.3	0.1	0.1
de	0.2	-0.3	0.4	0.2
patatas	0.1	0.2	-0.1	-0.1
<pad>	0	0	0	1

Aplicamos el filtro sobre la entrada:

3	1	2	-3
-1	2	1	-3
1	1	-1	1

Filtro 1 (tamaño 3)

<pad>+me+gusta	-3.6
me+gusta+la	1.4
gusta+la+tortilla	-0.5
la+tortilla+de	-3.6
tortilla+de+patatas	-0.2
de+patatas+pad	2.0

Cómo usar CNN para PLN

<pad>	0	0	0	1
Me	0.2	0.1	-0.3	0.4
gusta	0.5	0.2	-0.3	-0.1
la	-0.1	-0.3	-0.2	0.4
tortilla	0.3	-0.3	0.1	0.1
de	0.2	-0.3	0.4	0.2
patatas	0.1	0.2	-0.1	-0.1
<pad>	0	0	0	1

Filtro 1

3	1	2	-3
-1	2	1	-3
1	1	-1	1

Feature map del Filtro 1

-3.6
1.4
-0.5
-3.6
-0.2
2.0

Cómo usar CNN para PLN

<pad>	0	0	0	1
Me	0.2	0.1	-0.3	0.4
gusta	0.5	0.2	-0.3	-0.1
la	-0.1	-0.3	-0.2	0.4
tortilla	0.3	-0.3	0.1	0.1
de	0.2	-0.3	0.4	0.2
patatas	0.1	0.2	-0.1	-0.1
<pad>	0	0	0	1

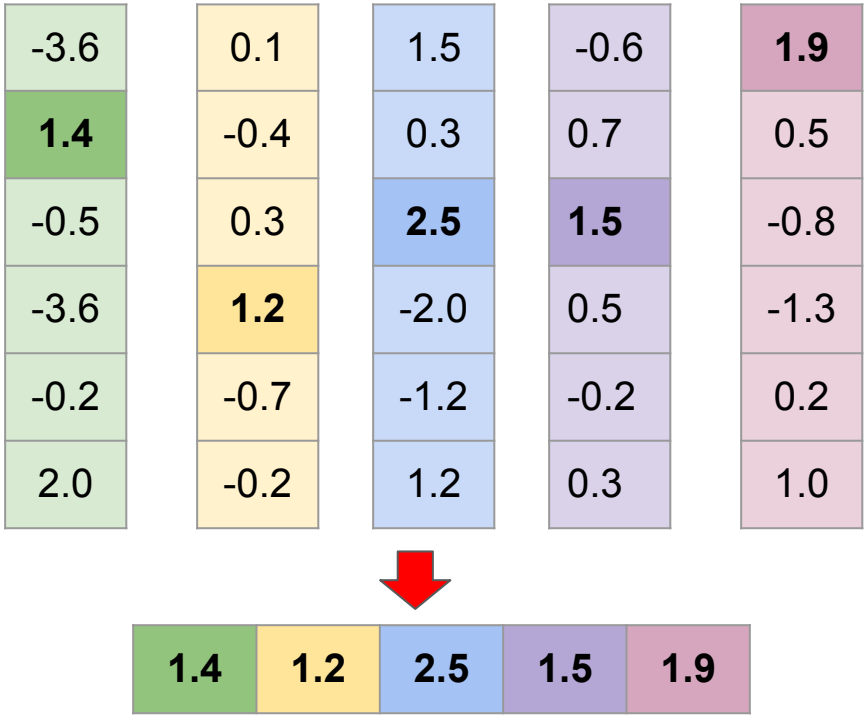
- Supongamos que hemos aplicado 5 filtros (de tamaño 3), y hemos obtenido los siguientes feature maps:

Feature map Filtro 1	Feature map Filtro 2	Feature map Filtro 3	Feature map Filtro 4	Feature map Filtro 5
-3.6	0.1	1.5	-0.6	1.9
1.4	-0.4	0.3	0.7	0.5
-0.5	0.3	2.5	1.5	-0.8
-3.6	1.2	-2.0	0.5	-1.3
-0.2	-0.7	-1.2	-0.2	0.2
2.0	-0.2	1.2	0.3	1.0

Cómo usar CNN para PLN

<pad>	0	0	0	1
Me	0.2	0.1	-0.3	0.4
gusta	0.5	0.2	-0.3	-0.1
la	-0.1	-0.3	-0.2	0.4
tortilla	0.3	-0.3	0.1	0.1
de	0.2	-0.3	0.4	0.2
patatas	0.1	0.2	-0.1	-0.1
<pad>	0	0	0	1

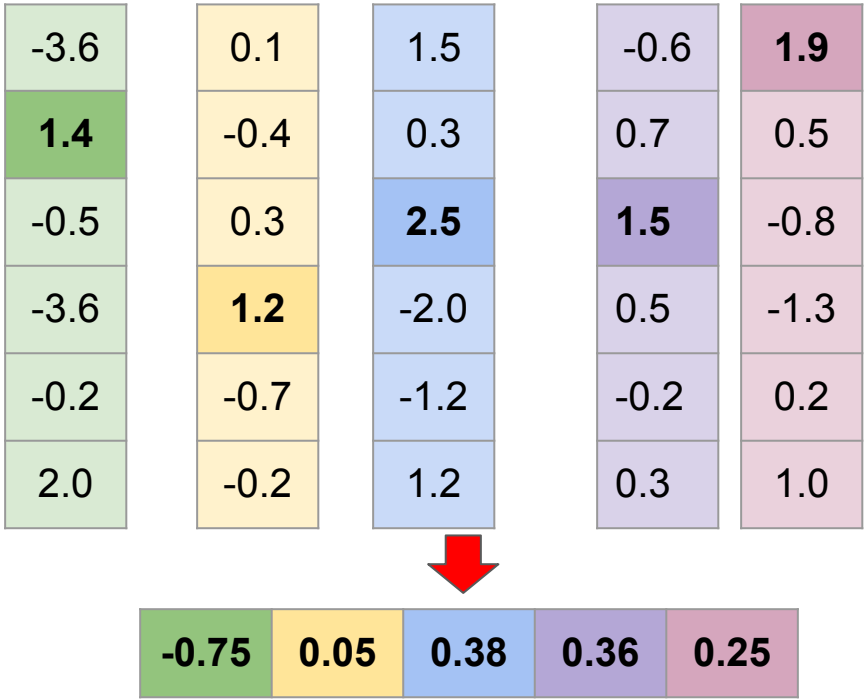
Aplicamos, por ejemplo, max-pooling:



Cómo usar CNN para PLN

<pad>	0	0	0	1
Me	0.2	0.1	-0.3	0.4
gusta	0.5	0.2	-0.3	-0.1
la	-0.1	-0.3	-0.2	0.4
tortilla	0.3	-0.3	0.1	0.1
de	0.2	-0.3	0.4	0.2
patatas	0.1	0.2	-0.1	-0.1
<pad>	0	0	0	1

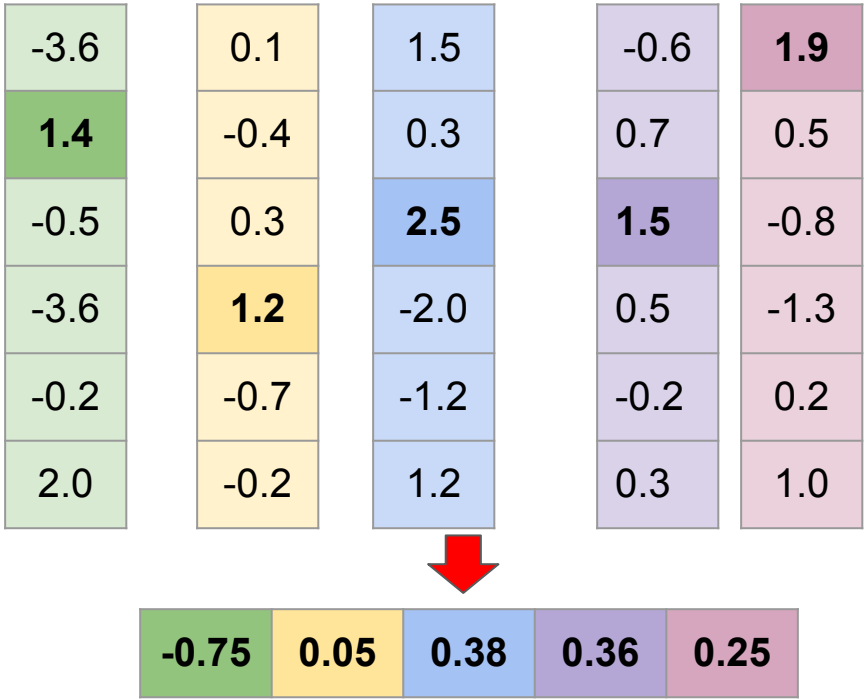
Si aplicaramos average-pooling:



Cómo usar CNN para PLN

<pad>	0	0	0	1
Me	0.2	0.1	-0.3	0.4
gusta	0.5	0.2	-0.3	-0.1
la	-0.1	-0.3	-0.2	0.4
tortilla	0.3	-0.3	0.1	0.1
de	0.2	-0.3	0.4	0.2
patatas	0.1	0.2	-0.1	-0.1
<pad>	0	0	0	1

Si aplicaramos average-pooling:



Cómo usar CNN para PLN

- Un modelo **CNN** puede contener varias combinaciones de **capas convolucionales** y **pooling**.
- También puede **incluir otras capas** como Dropout, RELU correction layer o una capa densa (fully-connected).
  - **Dropout**: permite ignorar aquellas neuronas cuyos pesos son más bajos que la probabilidad definido en la capa de dropout. Evitan overfitting.
  - **RELU** correction layer (f(x) = max(0,x)), muy simple y rápido de calcular (su derivada es 0 o 1). => mejora los tiempos de entrenamiento.
  - **Capa densa**, suele mejorar los resultados en tareas de clasificación de textos.

Cómo usar CNN para PLN

- La **última capa** de la red CNN dependerá de la tarea que se está resolviendo.
- Por ejemplo:
  - En **clasificación binaria**, esta capa debería ser la función **sigmoid**. Esta función devuelve un valor entre 0 y 1, que indica la probabilidad de que la entrada pertenezca a la clase positiva.
  - En **multi-clasificación**, esta capa debería estar formada por **neuronas softmax** (tantas neuronas como clases). La función softmax devuelve la probabilidad por cada clase. Entonces la red devuelve la **clase con mayor probabilidad**.

Resumen

- Una red convolucional es capaz de detectar de forma automática las características más importantes de un objeto (texto o imagen) sin necesidad de supervisión humana.
- Más eficiente que otras redes neuronales. Por ejemplo, una red densa tiene muchas más conexiones, y su entrenamiento tendrán una mayor complejidad temporal.
- Además, la operación de pooling reduce aún más la complejidad, tomando los elementos más representativos, y eliminando el resto.

OpenCourseWare
Procesamiento de Lenguaje Natural con Aprendizaje Profundo,

Gracias!!!

https://github.com/isegura