# 92586 Computational Linguistics

Lesson 17. Convolutions in Text

Alberto Barrón-Cedeño

Alma Mater Studiorum-Università di Bologna
a.barron@unibo.it @_albarron_

29/04/2021


## Table of Contents



## Prologue to CNN and RNN

## Prologue

- We have learned to build embedding spaces for words and texts
- We are considering the neighborhood of the words (∼the bag)
- We are not considering *actual* connections yet
- The downstream application is usually classification or regression

**We will start heading towards text generation**

## Words have relations and influence each other

Word order

$s_1$ = The dog chased the cat.
$s_2$ = The cat chased the dog.

$$\begin{aligned} sim(tfidf(s_1), tfidf(s_2)) &= 1 \\ sim(wv(s_1), wv(s_2)) &= 1 \\ sim(dv(s_1), dv(s_2)) &= 1 \end{aligned}$$

But $s_1$ and $s_2$ are not the same!

**Word proximity**

$s$ = His mother, besides her son's willingness to amend the issue, decided to punish him

mother. . . decided | son. . . him

(Lane et al., 2019, p. 220)

## Words have relations and influence each other

**Spatial relation**
Consider the position of words (∼written)

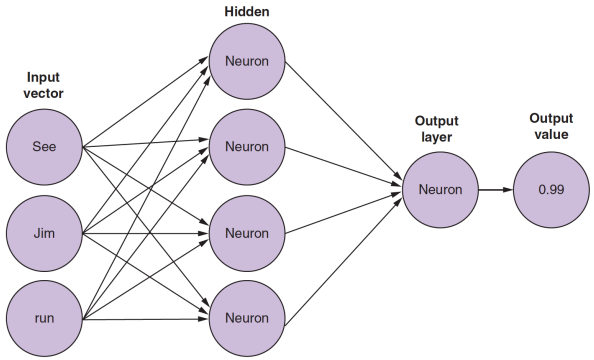
→ fixed-width window
**convolutional neural networks**

**Temporal relation**
Consider words as time series (∼spoken)

→ ongoing (unk) amount of time
**recurrent neural networks**

(Lane et al., 2019, p. 220)

## Multiple Input Words




- Three tokens are passed at a time
- Two input alternatives
  - one-hot vector
  - pre-trained word vector

`See Jim run` ≠ `run See Jim` (!)

(Lane et al., 2019, p. 221)

## Back to Keras

**Sequential()**

- Python class
- Neural network abstraction
- Grants access to the basic Keras API

**Sequential.compile()**

- Builds the underlying weights
- Builds the interconnected relationships

**Sequential.fit()**

- Computes the training errors (loss)
- Applies backpropagation (weight adjustment)

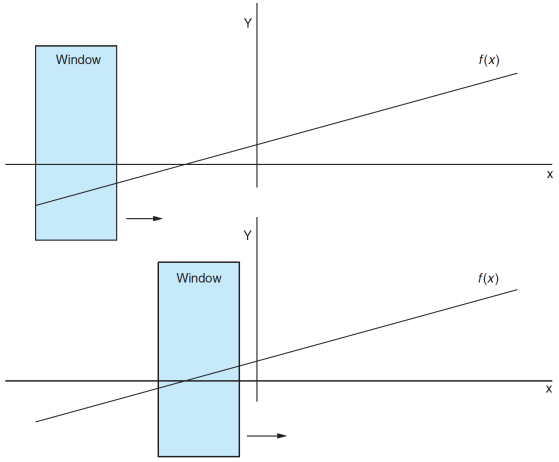
**Some "cooking" hyperparameters**

epochs number of iterations over the data
batch_size number of instances before adjusting
optmizer function

**CNN**

# Convolutional Neural Networks

Sliding —or convolving[1]— a window over the sample

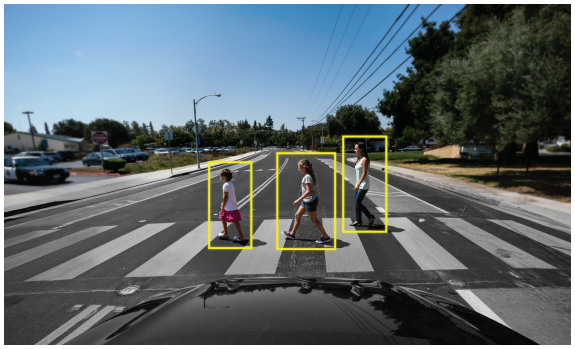



[1] To roll or wind together (Webster's)
(Lane et al., 2019, p. 222)

# Convolutional Neural Networks

Back to the roots: image recognition

- Input: pixels of an image
- Output: the image contains $x$

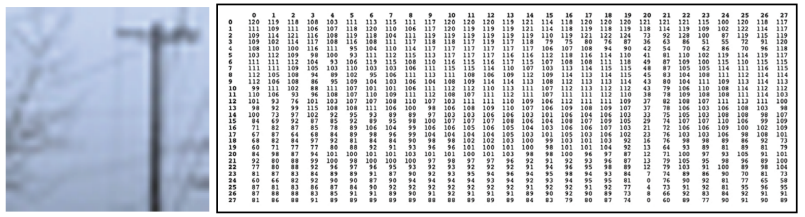


https://blogs.nvidia.com/wp-content/uploads/2019/04/ADAS-IMG_0052.jpg

# Convolutional Neural Networks

When the input is an image

- B&W: [0,1] (with a smooth binariser)
- Grayscaled: [0, 255]
- Colour: R: [0, 255] G: [0, 255] B: [0, 255]



(Lane et al., 2019, p. 223)

# Convolutional Neural Networks

When the input is an image

**An image is just a bunch of numbers**

- Appropriate as input for a NN
- But one single pixel has no real meaning

→ **Sliding over fragments of the image**

The convolution defines a set of filters (aka kernels) to do just that

- Take "snapshots" of different areas of the image
- Process them, one at a time

# Convolutional Neural Networks

Strides and filters

**Stride**

- The distance "traveled" when sliding
- Yet another parameter
- Never bigger than the size of the filter → overlapping areas

Sounds familiar? ***n*-grams!**

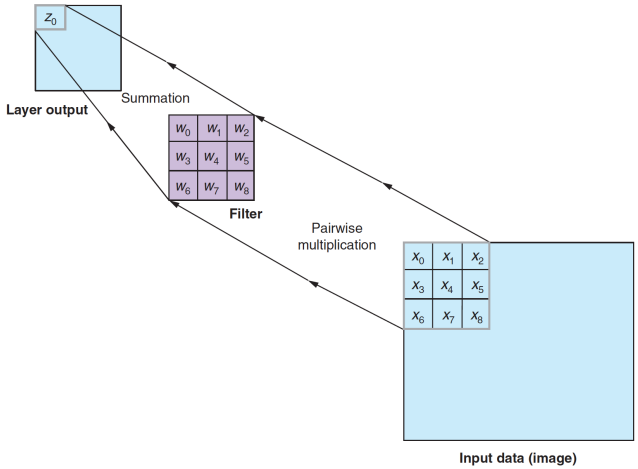
**Filter**

- $n \times m$ surfaces
- Typically $n = m = 3$ (often $n \neq m$)
- Includes a set of weights (fix for the whole image)
- Includes an activation function: usually ReLU

$$z = \max(sum(x * w), 0)$$

# Convolutional Neural Networks

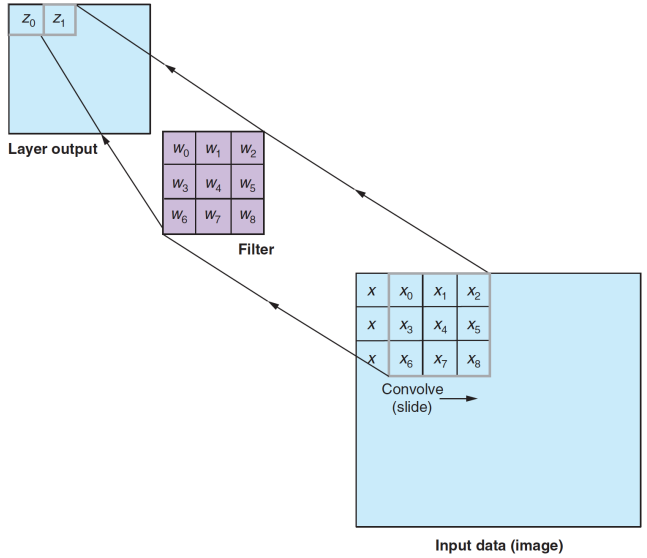
Convolutional step




(Lane et al., 2019, p. 225)

# Convolutional Neural Networks

Convolution




(Lane et al., 2019, p. 226)

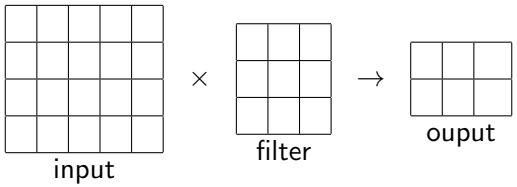
## Convolutional Neural Networks

Producing multiple images

- $k$ filters exist which carry out different operations
- Every filter will produce a new image, combination of source and filter

## Convolutional Neural Networks

Padding




**We are producing smaller images**

"I don't care": Keras' argument `padding='valid'`
The edges of the image are undersampled

"I do care": Keras' padding argument `padding='same'`

In NLP **we care**

## Convolutional Neural Networks

Pipeline

**Input:** an image, text
**Output:** a class, a real number

- Produce $k$ new images through $k$ filters
- Wire the filtered images to a feed-forward
- Proceed as usual

**We can add multiple convolution layers**
A full path of learning layers and abstractions

- Edges
- Shapes
- Colours
- **Concepts**

**What is learned**

- Good filters
- "Standard" weights

## Convolutional Neural Networks

Keras premier

```
from keras.models import Sequential
from keras.layers import Conv1D

model = Sequential()

model.add(Conv1D(filters=16,
                 kernel_size=3,
                 padding='same',
                 activation='relu',
                 strides=1,
                 input_shape=(100, 300))
)
```

**Quick Reminders on CNNs**

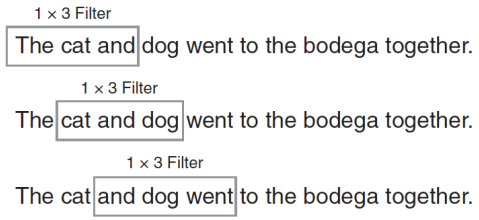
## Quick Reminders on CNNs

- Sliding —or convolving— a window over the sample
- Filters (kernels; matrices) slide over fragments of the image
- "Snapshots" of different areas of the image are taken and processed
- Multiple filters produce multiple images
- Multiple convolution layers can be added
- At the end, we can plug a "standard" fully-connected NN

**CNNs for NLP**

## Back to Text

- In images both vertical and horizontal relationships are relevant
- In text only horizontal ones do[2]
- **We need "1D" filters**

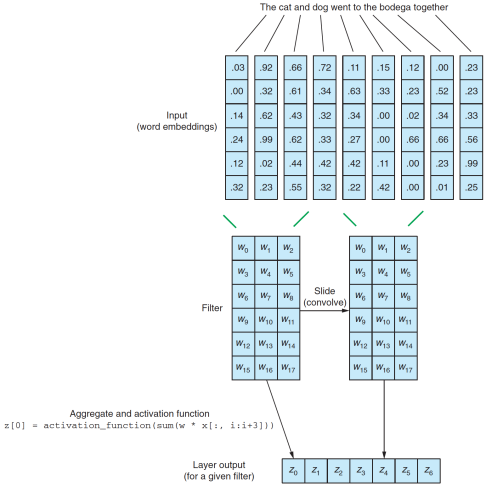



[2] l2r or r2l; for some languages it's the vertical direction that matters (e.g., Japanese)

(Lane et al., 2019, p. 229)

## But we do have 2D "filters"

Words are represented with word embeddings: vectors




(Lane et al., 2019, p. 229)

## The convolution is (practically) the same as for images

- We now *convolve* in one dimension (not two)
- The computation order is irrelevant, but the outputs have to be fed in the same order
- The filters' weights are fixed for a full sample (parallel computation)
- Their output becomes the features for the classifier

Let us see

## Padding

- (In general) in image processing the inputs are of fixed size, regardless of the image (same source!)
- Texts are not fixed length (regardless of their source)

- Instances longer than `maxlen` will be truncated
- Instances shorter than `maxlen` will be **padded**

$x_0, x_1, x_2, x_3, \ldots x_{398} x_{399}$ $x_{400} x_{401}$

$x_0, x_1, x_2, x_3, \ldots x_{397}$ PAD PAD

Let us see

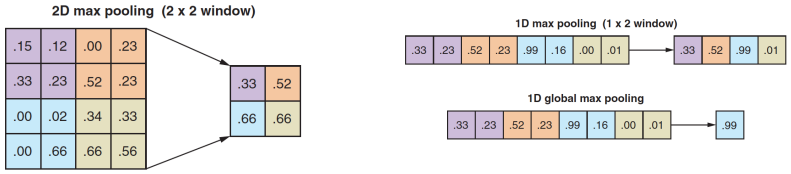
## Pooling

- For each filter one new version of the instance is produced (250 in the example)
- Pooling evenly divides the output of each filter into subsections
- It selects (or computes) a representative value for each subsection

# Pooling

Pooling is "the CNN path to dimensionality reduction [...] by learning higher-order representations of the source data" (Lane et al., 2019, p. 236)

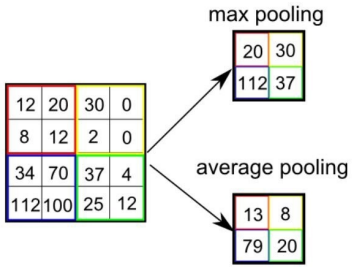
- The filters job is finding patterns → relationships between words and their neighbours
- Pooling in text: a 1D window (e.g., $1 \times 2$ or $1 \times 3$)




(Lane et al., 2019, p. 237)

# Pooling

Max vs Average Pooling




- Average is more intuitive: retaining most of the info
- Max is better: the NN keeps the most prominent feature

Let us see

Image borrowed from `www.quora.com/What-is-max-pooling-in-convolutional-neural-networks`

# Recap

- Each filter will produce a $1 \times 398$
- For each of the 250 filter outputs , we take the single maximum value for each 1D vector
- Output: one $1 \times 250$ vector

**This is a crude semantic representation of the text**

# Dropout: Preventing Overfitting

On each training pass **turn off** a percentage of the input of a layer; it will become 0

- Chosen randomly on each pass
- It will not rely heavily on any feature
- It will be generalize better
- Dropout is applied on training only

Let us see

Photogram of "The Platform" (2019)

## Workhorse Loss Functions

Out of the 13+ available loss functions:

binary_crossentropy: the output neuron is either on or off

categorical_crossentropy: the output is one out of many classes

Let us see

## Closing Remarks

- Your input is a series of max 400 words; 300 elements each
- Nothing prevents you from stacking other embeddings (think of RGB)
- The output of the convolution layer is tied to the task (in this case, sentiment analysis)
- A CNN is more efficient, thanks to the pooling process and the filters
- You can add many convolution layers

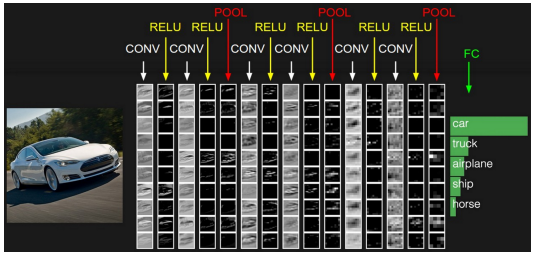



Image borrowed from `https://blog.mapillary.com`

## Next time

- Recurrent Neural Networks

## References

Lane, H., C. Howard, and H. Hapkem
2019. *Natural Language Processing in Action*. Shelter Island, NY: Manning Publication Co.