

# Convolution for text

A Recurrent Neural Network may be an ideal mechanism for dealing with sequential data like text.

But a one dimensional CNN may be an even simpler mechanism.

We briefly introduce the idea as it may deepen our understanding of the particular issues of text.

An  $n$ -gram is a sequence of  $n$  consecutive tokens that encapsulates a single concept (*phrase*) such as:

- "New York City" versus [ "New", "York", "City" ]

An  $n$ -gram can also capture subtleties of ordering

- [ "hard", "not", "easy" ] versus [ "easy", "not", "hard" ]

How does one identify consecutive tokens that may improve prediction ?

There are two approaches.

The first is statistical

- The joint frequency of consecutive tokens being higher than the frequency assuming independence
- $p(\text{"New York City"}) > p(\text{"New"})p(\text{"York"})p(\text{"City"})$

The second way: use Machine Learning !

We have spoken about convolutions as

- Identifying the presence/absence of a feature
- At a *spatial* location

The one-dimensional convolution, when applied to a sequence of tokens

- Identifies the presence/absence of a feature
- At a *temporal* location (index within the sequence)

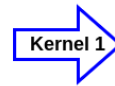
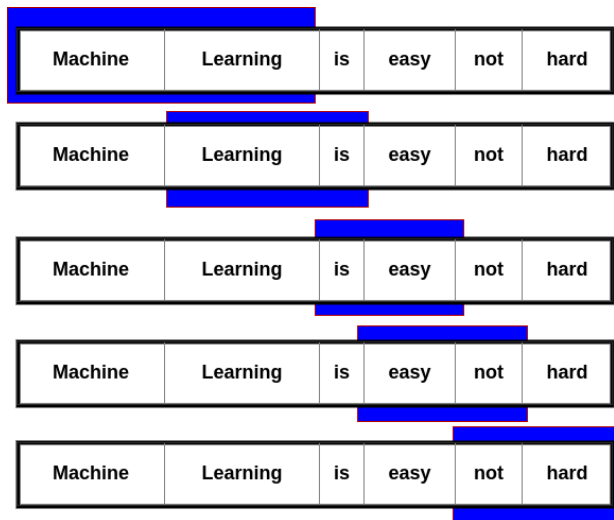
This is just an ordinary convolution, applied to a sequence.

It is only able to capture *local* relationships that occur within the width of the convolutional kernel.

Here is a picture:

- A kernel of size 2 (blue) recognizing the pattern "Machine Learning"
- Being slid over the input sequence
- Producing a high output (red) when the consecutive tokens match the pattern

One dimensional convolution  
Slide blue kernel over input



Pattern: "Machine Learning"

Machine Learning	Learning is	is easy	easy not	not hard
---------------------	----------------	------------	-------------	-------------



Using one dimensional convolution with kernel size  $n_{(l)}$

- The convolution creates an  $n$ -gram feature
- At each (temporal) location in the sequence

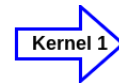
As with any other CNN, we can apply multiple kernels

- Each matching a different pattern
- To identify a different feature (n-gram)
- At each location in the sequence

## One dimensional convolution multiple kernels

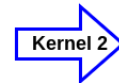
---

Machine	Learning	is	easy	not	hard
---------	----------	----	------	-----	------



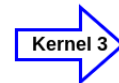
Pattern: "Machine Learning"

Machine Learning	Learning is	is easy	easy not	not hard
---------------------	----------------	------------	-------------	-------------



Pattern: "Is easy"

Machine Learning	Learning is	is easy	easy not	not hard
---------------------	----------------	------------	-------------	-------------



Pattern: "not hard"

Machine Learning	Learning is	is easy	easy not	not hard
---------------------	----------------	------------	-------------	-------------

In [ ]: Convolutional Layer  $\ell$  thus produces  $y_{\ell}$

- Of the same temporal/spatial dimension as  $y_{(\ell-1)}$
- With  $n_{\ell}$  features

After constructing  $n$ -gram features at layer  $l$

- We get  $\mathbf{y}_{(l)}$
- Of the same shape as  $\mathbf{y}_{(l-1)}$

That is: we transform a sequence of tokens into an equal sequence of  $n$ -grams

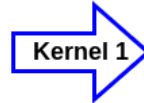
Here is a picture

- Using 3 kernels of width 2 to identify
- 3 synthetic features ("2-gram") at each location in the sequence
- Followed by Global Poolin to reduce the sequence for each feature
- To a single value per feature

Global Pooling  
3 features over spatial locations  
to 3 features over one location

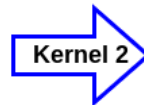
---

Where does feature occur in input



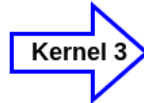
Pattern: "Machine Learning"

Machine Learning	Learning is	is easy	easy not	not hard
------------------	-------------	---------	----------	----------



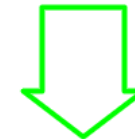
Pattern: "Is easy"

Machine Learning	Learning is	is easy	easy not	not hard
------------------	-------------	---------	----------	----------



Pattern: "not hard"

Machine Learning	Learning is	is easy	easy not	not hard
------------------	-------------	---------	----------	----------



Global Pooling

Feature exists somewhere in input

Machine Learning

Machine Learning is easy not hard

is easy

Machine Learning is easy not hard

not hard

Machine Learning is easy not hard

The resulting vector of 3 features can then be fed into a Classical ML layer such as Classification.

Our notebook will demonstrate code for the entire process.



# Conclusion

Ordering of tokens is important for understanding text.

Convolutional Layers

- By capturing temporally local relationships
- May create features ("n-grams") that are more useful
- Than isolated tokens

This is important in general, but particularly when a subsequent layer (e.g., Global Pooling) loses ordering.

In [2]: `print("Done")`

Done