Convolution for text

A Recurrent Neural Network may be an ideal mechanism for deal 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*)

An n-gram can capture

- multi-token concept
    - "New York City" versus ["New", "York", "City"]
- ordering information
    - ["hard", "not", "easy"] versus ["easy", "not", "hard"]

Rather than capturing the entire sequence an n-gram captures spatial locality in a limited window. (kernel width)

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: train a Convolutional layer.

If

- There is a sequence of consecutive tokens
- That creates a synthetic feature
- Useful for prediction
- Training will discover a kernel that creates this feature

This is similar to what we speculated was happening when using CNN's to recognize images.

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

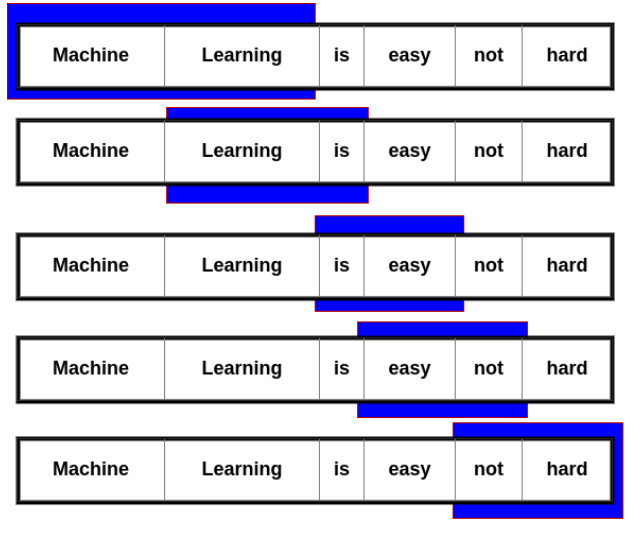
- The convolution creates an n-gram
- At each location in the sequence

By using multiple kernels in layer l, we can create many n-grams.

The diagram illustrates

- 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

After constructing n-gram features at layer l

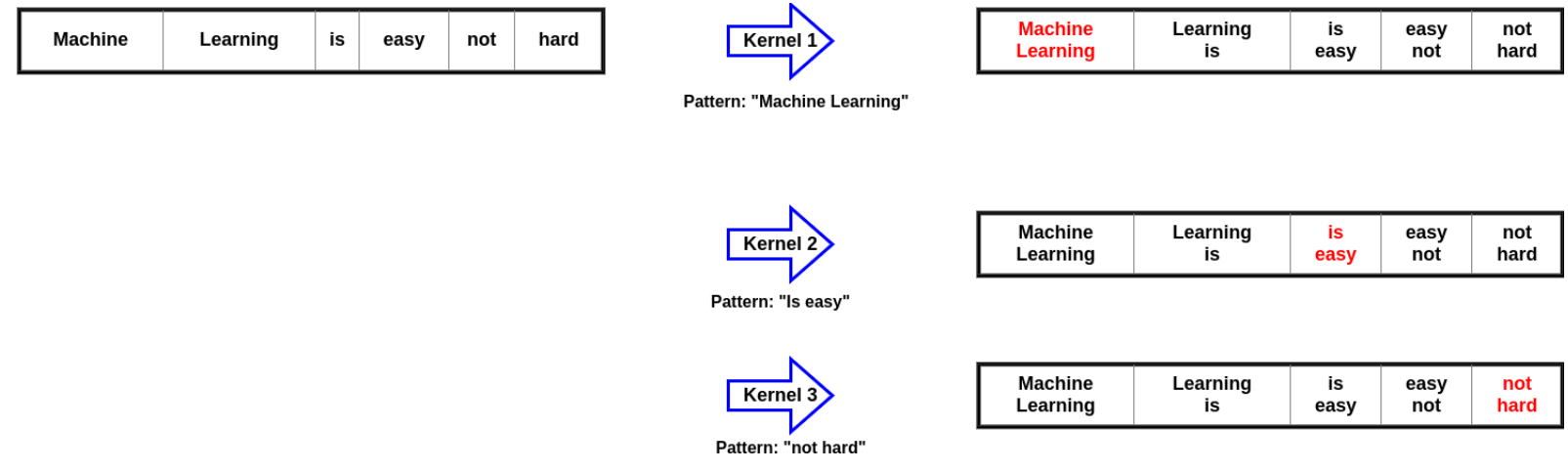
- We get $\mathbf{y}_{(l)}$
- Of the same shape as $y_{(l-1)}$

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

As with any other CNN, we can apply multiple kernels

- Each matching a different pattern

One dimensional convolution
multiple kernels



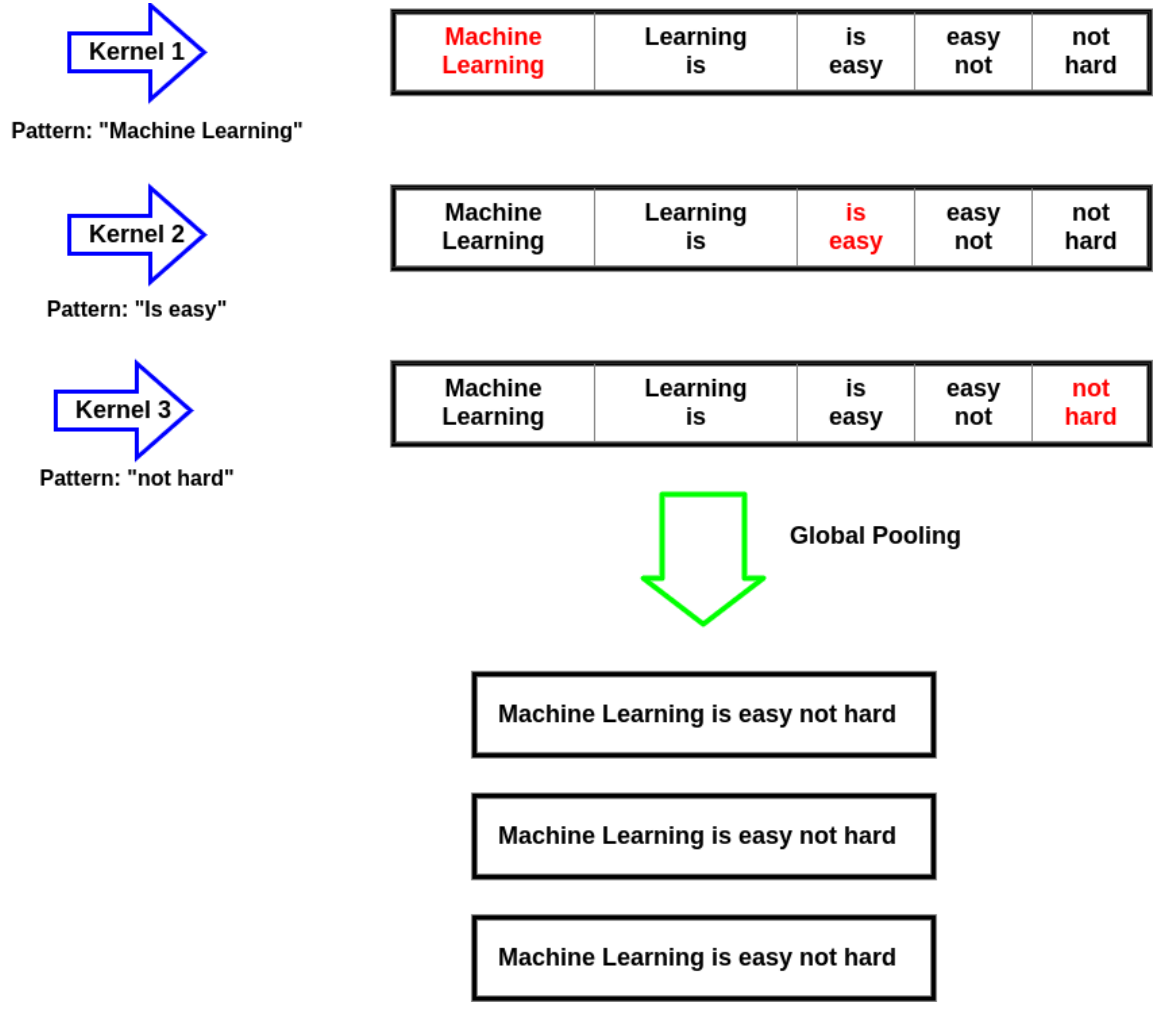
And we can then apply whatever method we choose

- To transform the variable length sequence
- To a fixed length

Here's an illustration of summarization using Pooling

- Each row is a synthetic feature, at each spatial location
- Pooled to reduce the spatial locations to a single value
- For each feature

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



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 isolted tokens
- Particularly after Pooling, which looses ordering

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

```
Done
```