Convolution and Pooling

Benjamin Roth, Nina Poerner

CIS LMU München

November 26, 2019

Convolutional Neural Networks (CNNs)

- Technique from Computer Vision (e.g., object recognition in images)
- Alternative to RNNs for many (not all) NLP tasks
- General idea: Filter bank with N learnable filters

Convolution with one filter

- For now, assume we have only one filter
- Move filter over input with step size (stride) s (here: 1)
- At every position, multiply filter and input entries together (elementwise), and sum the results into a single value

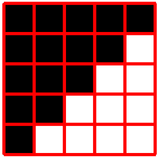
Input, Filter, Output

$$\begin{array}{|c|c|c|} \hline 0 & 0 & -1 \\ \hline 1 & -1 & 0 \\ \hline -2 & 0 & 0 \\ \hline -1 & -3 & 1 \\ \hline \end{array} \; * \; \begin{array}{|c|c|} \hline 2 & 1 \\ \hline 0 & -1 \\ \hline \end{array} \; = \; \begin{array}{|c|c|} \hline 1 & -1 \\ \hline 1 & -2 \\ \hline -1 & -1 \\ \hline \end{array}$$

- Filter size: 2×2
- Input size: 3×4

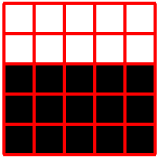
Building an edge detector filter

- Assume that -1 means black and +1 means white
- We want to build a filter that can detect diagonal edges where the upper left side is dark and the lower right side is bright
- = a filter that calculates a high positive number on windows that look like this:



-3	-3	-3	-3	-3
-3	-3	-3	-3	3
-3	-3	-3	3	3
-3	-3	3	3	3
-3	3	3	3	3

How would the filter react to this window?



- In CNNs, the filters are not manually chosen, but learned with gradient descent

Convolution with one filter: Tensor sizes

- Most images are not 2D but 3D
  - 3rd dimension is # channels, e.g., RGB values
  - image height $\times$ image width $\times$ # channels
- As a consequence, each filter is also 3D
  - filter height $\times$ filter width $\times$ # channels
- The operation stays the same, with an additional summation over the channel dimension

Convolution with N filters

- Apply N different filters of the same size $\rightarrow$ N matrices with the same size
- Stack the N matrices on top of each other $\rightarrow$ 3D tensor, where the last dimension is N
- Also known as a feature map
- Feature map is slightly smaller than input (why?)
- Because a filter of size k fits into an input of size h only $h - k + 1$ times
- ... unless we pad the input with zeros

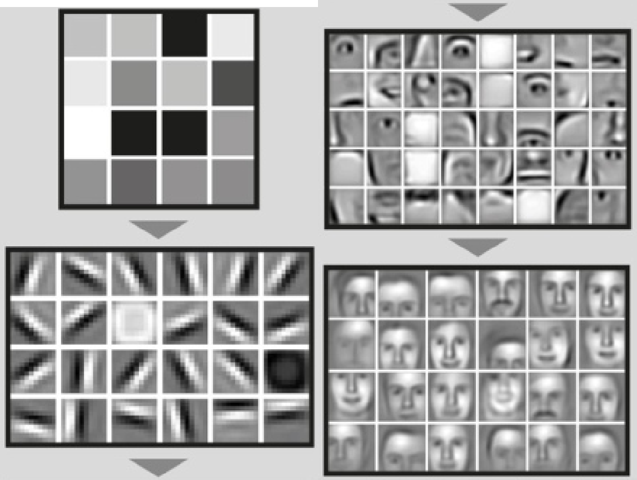
Convolution with N filters: Tensor sizes

- Tensor sizes:
  - **Input 3D**: input height $\times$ input width $\times$ # channels (if this is the first layer, otherwise # filters of previous layer)
  - **Parameter tensor 4D**: filter height $\times$ filter width $\times$ # channels $\times$ #filters
  - **Output 3D**: input height* $\times$ input width* $\times$ #filters
  - *height and width are slightly reduced by convolution unless we do padding

What does convolution do?

- Contextualization: Feature vector computed for position (i, j) contains info from $(i-k, j-k)$ to $(i+k, j+k)$ (where k is filter size).
- Locality-preserving: In one convolution layer, info can travel no further than k positions
- Computer Vision: Many convolutional layers applied one after another
- Typical nonlinearity between convolution layers: ReLU
- With every layer, feature maps become more complex
- Pixels $\rightarrow$ edges $\rightarrow$ shapes $\rightarrow$ small objects $\rightarrow$ bigger, compositional

Convolution



Source: Computer science: The learning machines. Nature (2014).

Pooling

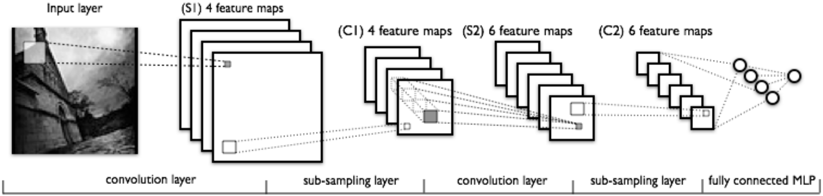
- Often applied between convolution steps
- Divide feature map into “grid”
- Combine vectors inside the same grid cell with some operator
- Most popular: Average pooling, Max pooling
- Max pooling: only select maximum value for each dimension
- “Feature detector”, “Cat neuron fires”

2	1	2	0
1	0	2	0
0	4	2	3
0	5	1	0

Max pooled

2	2
5	3

Convolution and Pooling: LeNet



LeCun et al. (1998). Gradient-based learning applied to document recognition.

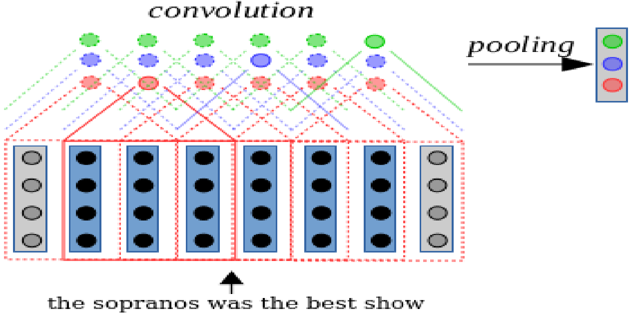
Convolution for NLP

- Images have width and height, but text only has "width" (length)
- $\rightarrow$ We can discard the "height" dimension from our filters
- Tensor sizes (in NLP):
  - **Input 2D**: sentence length $\times$ # channels (word embedding size, or # filters of previous convolution)
  - **Parameter tensor 3D**: filter length $\times$ # channels $\times$ #filters
  - **Output 2D**: sentence length* $\times$ #filters
  - *length slightly reduced unless we do padding
- Computer vision: 2D convolution (over height and width)
- NLP: 1D convolution (over length)
- Typically fewer convolutional layers than Computer Vision

Pooling for NLP

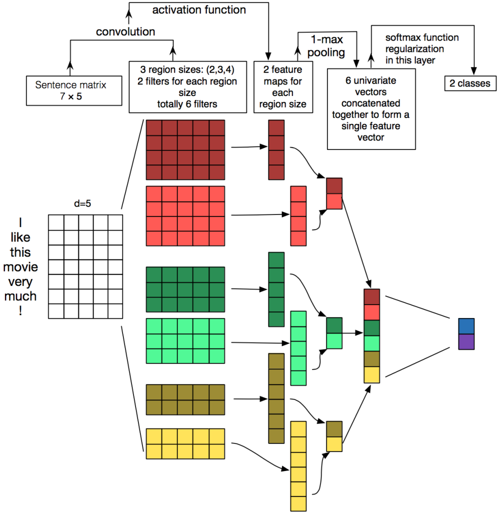
- Pooling between convolutional layers less frequently used than in Computer Vision
- After last convolutional layer: "global" pooling step
- Calculate max/average over the entire sequence ("pooling over time")

Convolution and Pooling for NLP



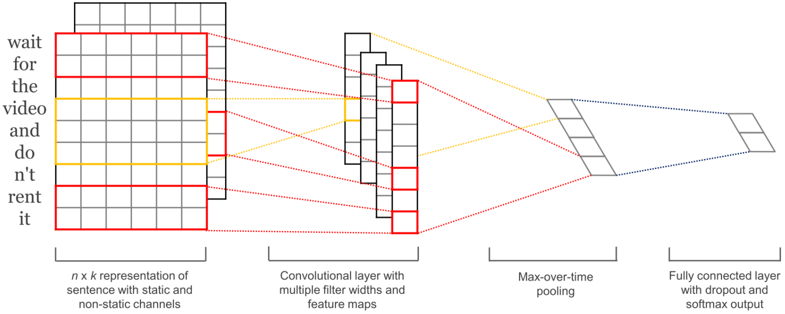
- What is the unpadded input size (=length)? 6
- What is the padded input size? 8
- How many filters? 3
- How many input channels (=word vector dimensions)? 4
- What is the filter size (=filter width)? 3
- What stride (=step size)? 1
- What is the output size of the convolution operation? 6×3
- What is the output size of the pooling operation? 3
- How many parameters have to be learned? $3 \times 3 \times 4 = 36$

Convolution and Pooling for NLP



Source: Zhang, Y., & Wallace, B. (2015). A Sensitivity Analysis of ConvNets for Sentence Classification.

Convolution and Pooling for NLP



Source: Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification.

```python
# binary classifier, e.g., sentiment polarity

from keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense
from keras.models import Sequential

embedding = Embedding(input_dim = VOCAB_SIZE, output_dim = EMB_DIM)
conv_layer = Conv1D(filters = NUM_FILTERS, kernel_size = FILTER_WIDTH,
activation = "relu")
pool_layer = GlobalMaxPooling1D()
dense_layer = Dense(units = 1, activation = "sigmoid")

model = Sequential(layers = [emb_layer, conv_layer, pool_layer, dense_layer])
model.compile(loss = "binary_crossentropy", optimizer = "sgd")
X, Y = # load_data()
model.fit(X, Y)
```

RNN vs. CNN

- Range
  - CNN: Cannot capture dependencies with range above $k \times L$ (where k is filter width and L is the number of layers
  - RNN: Can capture long-range dependencies
- Information transport
  - RNN: Must learn to "transport" salient information across many time steps.
  - CNN: No information transport across time, salient information "fast-tracked" by global max pooling
- Efficiency
  - RNN: Sequential data processing $\rightarrow$ not parallelizable over time
  - CNN: Input windows are independent from one another $\rightarrow$ highly parallelizable over time

RNN vs. CNN: Quiz

- Given a task description, choose appropriate architecture!
  - Task: predict the number of the main verb (sleep or sleeps)
    - *The cats, who were sitting on the map inside the house, [sleep/sleeps?]*
  - Which architecture should we use? RNN
  - Task: predict the polarity of the review:
    - *[... many useless sentences ...] best book ever [... many useless sentences ...]*
  - Which architecture should we use? CNN
- Task: Machine Translation
- Which architecture should we use?
  - Intuitively RNN (because MT is all about long-range dependencies), but ...
  - **Attention** gives CNNs the ability to capture long-range dependencies, while maintaining parallel processing (Gehring et al.)
  - More about attention: Later in this course