INF5820: Language technological applications

Convolutional Neural Networks

Erik Velldal

University of Oslo

9 October 2018

So far: MLPs + embeddings as inputs

- Embeddings have benefits over discrete one-hot encodings; makes use of unlabeled data + information sharing across features.
- But we still lack power for representing sentences and documents.
- Concatenation? Would blow up the parameter space for a fully connected layer.
- Averaging? gives a fixed-length representation, but no information about order or structure.
- Need for specialized NN architectures that extract higher-level features:
- CNNs and RNNs – the agenda for the coming weeks.
- Learns intermediate representations that are then plugged into additional layers for prediction.
- Pitch: layers and architectures are like Lego bricks – mix and match.

Example text classification tasks

Document- / sentence-level polarity; positive or negative?

- *The food was expensive but hardly impressive.*
- *The food was hardly expensive but impressive.*

- Strong local indicators of class,
- some ordering constraints,
- but independent of global position.
- In sum: a small set relevant n-grams could provide strong features.

Many text classification tasks have the similar traits:...

- sentences as subjective or objective
- questions types
- authorship of texts
- text topics
- emails as spam
- comments as abusive

What would be a suitable model?

- BoW or CBoW? Not suitable:
- Do not capture local ordering.
- An MLP can learn feature combinations, but not easily positional / ordering information.
- Bag-of-n-grams or n-gram embeddings?
- Want to be able to share statistical strength between related features.
- Potentially wastes many parameters; only a few n-grams relevant.
- Data sparsity issues + does not scale to higher order n-grams.

- Want to learn to efficiently model relevant n-grams.
- Enter convolutional neural networks.

CNNs: overview

- AKA convolution-and-pooling architectures or ConvNets.

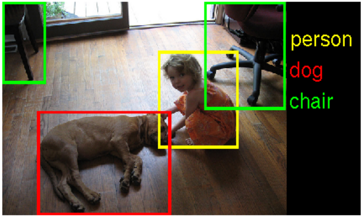
CNNs explained in three lines

- A convolution layer extracts n-gram features across a sequence.
- A pooling layer then samples the features to identify the most informative ones.
- These are then passed to a downstream network for prediction.

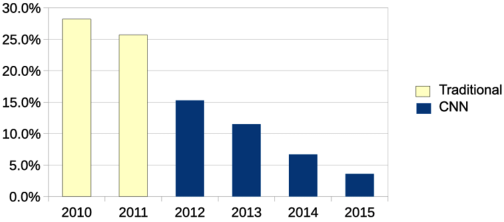
- We'll spend the next two lectures fleshing out the details.

CNNs and vision / image recognition

- Evolved in the 90s in the fields of signal processing and computer vision.
- 1989–98: Yann LeCun, Léon Bottou et al.: digit recognition
- 2012: Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton: great reduction of error rates for ImageNet object recognition



(Taken from image-net.org)



(Taken from Bottou et al. 2016)

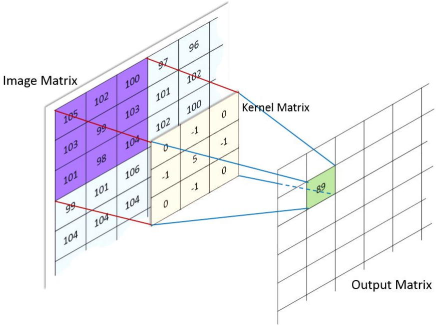
- These roots are witnessed by the terminology associated with CNNs.

2d convolutions for image recognition

- Generally, we can consider an image as a matrix of pixel values.
- The size of this matrix is *height* x *width* x *channels*:
- A gray-scale image has 1 channel, an RGB color image has 3.
- Several standard convolution operations are available for image processing: Blurring, sharpening, edge detection, etc.
- A convolution operation is defined on the basis of a kernel or filter: a matrix of weights.
- Several terms often used interchangeably: filter, filter kernel, filter mask, filter matrix, convolution matrix, kernel matrix, . . .
- The size of the filter referred to as the receptive field.

2d convolutions for image processing

- The output of an image convolution is computed as follows: (We're assuming square symmetrical kernels.)
  - Slide the filter matrix across every pixel.
  - For each pixel, compute the matrix convolution operation:
  - Multiply each element of the filter matrix with its corresponding element of the image matrix, and sum the products.
  - Edges requires special treatment (e.g. zero-padding or reduced filter).
- Each pixel in the resulting filtered image is a weighted combination of its neighboring pixels in the original image.



2d convolutions for image processing

- Examples of some standard filters and their kernel matrices.
- `https://en.wikipedia.org/wiki/Kernel_(image_processing)`

Convolutions and CNNs

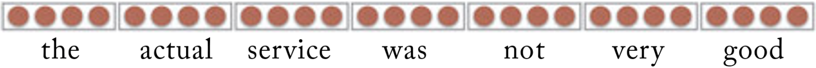
- Convolutions are also used for feature extraction for ML models.
- Forms the basic build block of convolutional neural networks.
- But then we want to learn the weights of the filter,
- and typically apply a non-linear activation function to the result,
- and typically also apply several filters.

But let's not get carried away, back to NLP:

- Convolution filters can also be used for feature extraction from text:
- 'n-gram detectors'.
- Pioneered by Collobert et al. (2008, 2011) for various tagging tasks, and later by Kalchbrenner et al. (2014) and Kim (2014) for sentence classification.
- A massive proliferation of CNN-based work in the field since.

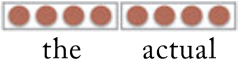
1d CNNs for NLP

- In NLP we apply CNNs to sequential data: 1-dimensional input.
- Consider a sequence of words $w_{1:n} = w_1, \ldots, w_n$.
- Each word is represented by a d dimensional embedding $\boldsymbol{E}_{[w_i]} = \boldsymbol{w_i}$.



- A convolution corresponds to 'sliding' a window of size k across the sequence and applying a filter to each.
- Let $\oplus(\boldsymbol{w_{i:i+k-1}}) = [\boldsymbol{w_i}; \boldsymbol{w_{i+1}}; \ldots; \boldsymbol{w_{i+k-1}}]$ be the concatenation of the embeddings $\boldsymbol{w_i}, \ldots, \boldsymbol{w_{i+k-1}}$.
- The vector for the ith window is $\boldsymbol{x_i} = \oplus(\boldsymbol{w_{i:i+k-1}})$, where $\boldsymbol{x_i} \in \mathbb{R}^{kd}$.

$\boldsymbol{x_1} \longrightarrow$

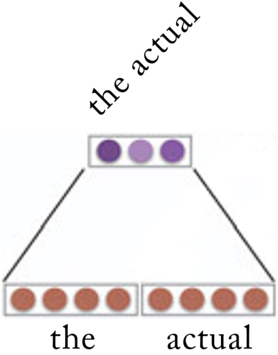


Convolutions on sequences

To apply a filter to a window $\boldsymbol{x_i}$:

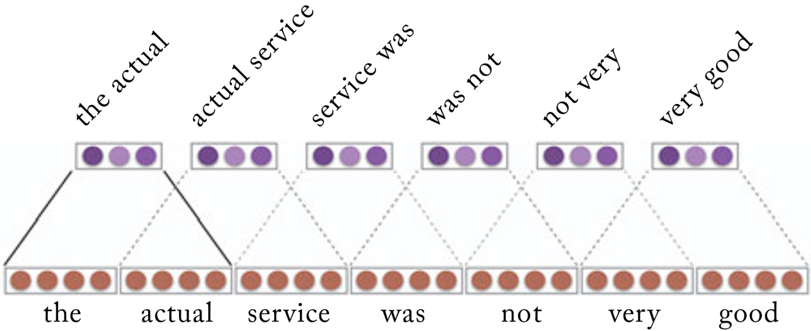
- compute its dot-product with a weight vector $\boldsymbol{u} \in \mathbb{R}^{kd}$
- and then apply a non-linear activation g,
- resulting in a scalar value $p_i = g(\boldsymbol{x_i} \cdot \boldsymbol{u})$

- Typically use ℓ different filters, $\boldsymbol{u_1}, \ldots, \boldsymbol{u_\ell}$.
- Can be arranged in a matrix $\boldsymbol{U} \in \mathbb{R}^{kd \times \ell}$.
- Also include a bias vector $\boldsymbol{b} \in \mathbb{R}^{\ell}$.
- Gives an ℓ-dimensional vector $\boldsymbol{p_i}$ summarizing the ith window: $\boldsymbol{p_i} = g(\boldsymbol{x_i} \cdot \boldsymbol{U} + \boldsymbol{b})$
- Ideally different dimensions captures different indicative information.



Convolutions on sequences

- Applying the convolutions over the text results in m vectors $\boldsymbol{p_{1:m}}$.
- Each $\boldsymbol{p_i} \in \mathbb{R}^{\ell}$ represents a particular k-gram in the input.
- Sensitive to the identity and order of tokens within the sub-sequence,
- but independent of its particular position within the sequence.

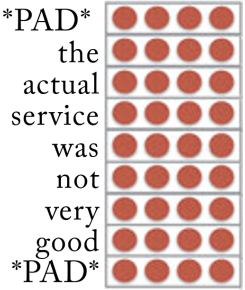


Narrow vs. wide convolutions

- What is m in $\boldsymbol{p_{1:m}}$?
- For a given window size k and a sequence $w_1, \ldots, w_n$, how many vectors $\boldsymbol{p_i}$ will be extracted?
- There are $m = n - k + 1$ possible positions for the window.
- This is called a narrow convolution.
- Another strategy: pad the with $k - 1$ extra dummy-tokens on each side.
- Let's us slide the window beyond the boundaries of the sequence.
- We then get $m = n + k + 1$ vectors $\boldsymbol{p_i}$.
- Called a wide convolution.
- Necessary when using window-sizes that might be wider than the input.

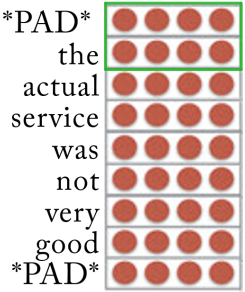
Stacking view (1:3)

- So far we've visualized inputs, filters, and filter outputs as sequences:
- What Goldberg (2017) calls the 'concatenation notation'.
- An alternative (and perhaps more common) view: 'stacking notation'.



- Imagine the n input embeddings stacked on top of each other, resulting in an $n \times d$ sentence matrix.
- Correspondingly, imagine each column $\boldsymbol{u}$ in the matrix $\boldsymbol{U} \in \mathbb{R}^{kd \times \ell}$ be arranged as a $k \times d$ matrix.
- We can then slide ℓ different $k \times d$ filter matrices down the sentence matrix, computing matrix convolutions:
- Sum of element-wise multiplications.

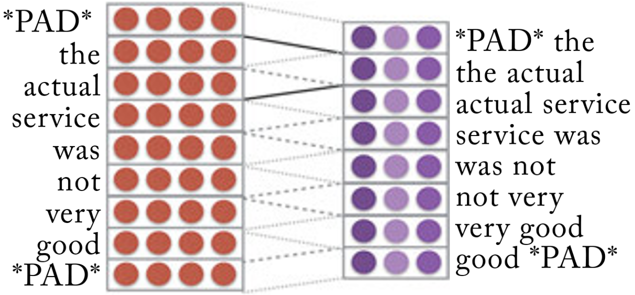
Stacking view (2:3)



- The stacking view makes the convolutions more similar to what we saw for images.
- Except the width of the 'receptive field' is always fixed to d,
- the height is given by k (aka *region size*),
- and we slide the filter in increments of d, corresponding to the word boundaries,
- i.e. along the height dimension only.

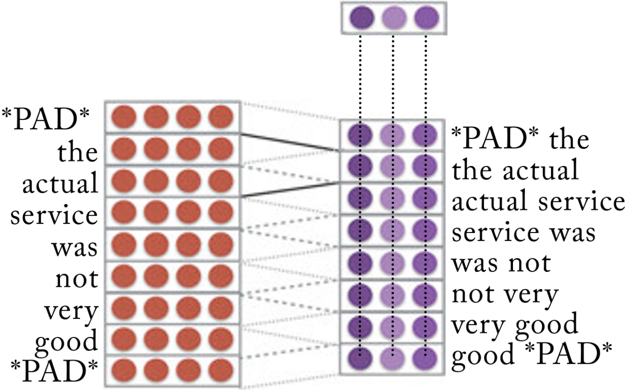
Stacking view (3:3)

- Now imagine the output vectors $\boldsymbol{p_{1:m}}$ stacked in a matrix $\boldsymbol{P} \in \mathbb{R}^{m \times \ell}$.
- Each ℓ-dimensional row of $\boldsymbol{P}$ holds the features extracted for a given k-gram by different filters.
- Each m-dimensional column of $\boldsymbol{P}$ holds the features extracted across the sequence for a given filter.
- These columns are sometimes referred to as feature maps.



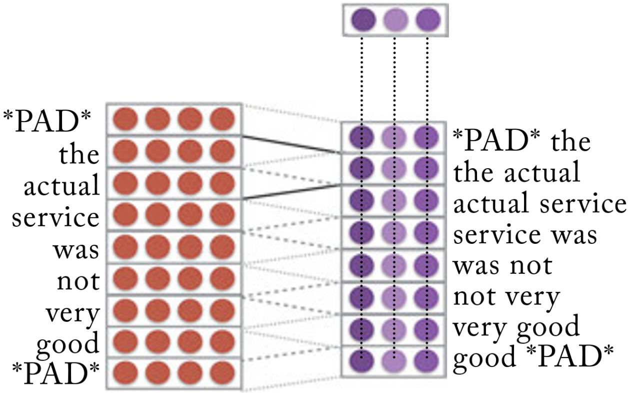
Next step: pooling (1:2)

- The convolution layer results in m vectors $\boldsymbol{p}_{\mathbf{1:m}}$.
- Each $\boldsymbol{p}_{\boldsymbol{i}} \in \mathbb{R}^{\ell}$ represents a particular k-gram in the input.
- m (the length of the feature maps) can vary depending on input length.
- Pooling combines these vectors into a single fixed-sized vector $\boldsymbol{c}$.



Next step: pooling (2:2)

- The fixed-sized vector $\boldsymbol{c}$ (possibly in combination with other vectors) is what gets passed to a downstream network for prediction.
- Want $\boldsymbol{c}$ to contain the most important information from $\boldsymbol{p_{1:m}}$.
- Different strategies available for 'sampling' features.



Pooling strategies

Max pooling

- Most common. AKA max-over-time pooling.
- $\boldsymbol{c}[j] = \arg\max_{1<i\leq m} \boldsymbol{p_i}_{[j]} \quad \forall j \in [1, l]$
- Picks the maximum value across each dimension (feature map).

Average pooling

- $\boldsymbol{c} = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{p_i}$
- CBOW or average of all the filtered k-gram representations.

K-max pooling

- Concatenate the k highest values for each dimension / filter.
- Preserves relative ordering information (but insensitive to specific positions).

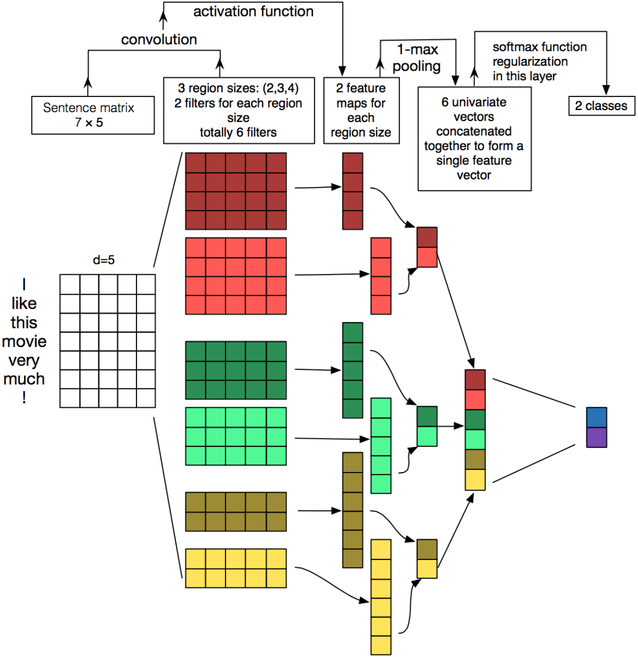
Dynamic pooling

- Combines with any of the strategies above.
- Perform pooling separately over r different regions of the input.
- Concatenate the r resulting vectors $\boldsymbol{c_1}, \ldots \boldsymbol{c_r}$.
- Allows us to retain positional information relevant to a given task (e.g. based on document structure).
- *Not* using dynamic pooling is sometimes called *global pooling*.

Multiple window sizes

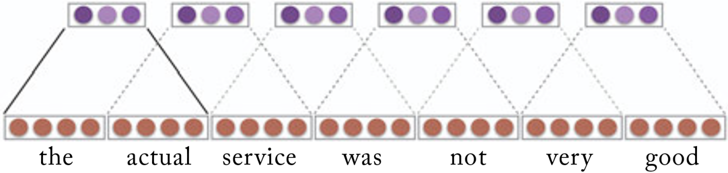
- So far considered CNNs with ℓ different filters for a single window size k.
- Typically, CNNs in NLP are applied with multiple window sizes, and multiple filters for each.
- Pooled separately, with the results concatenated.
- Rather large window sizes often used:
- 2–5 is most typical, but even $k > 20$ is not uncommon.
- With standard n-gram features, anything more than 3-grams quickly become infeasible.
- CNNs learn to represent large n-grams efficiently, without blowing up the parameter space and without having to represent the whole vocabulary.
- (Related to the notion of 'neuron' in a CNN – will get back to this!)

Baseline architecture of Zhang et al. (2017)


activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

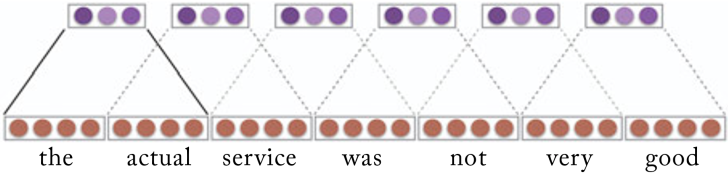
What is a neuron in a convolution? (1:2)

- A CNN has no backward connections between layers, no cycles (as we'll have once we get to RNNs).
- Can therefore be seen as a type of feed-forward network.
- But in contrast to the fully-connected ('dense') layers of an MLP, the convolution layers are 'sparsely connected'.
- Each filter defines m identical neurons:
- Each neuron instance is fully-connected only for a given k-gram.
- After (max-)pooling; only the most strongly activated neurons are used.



What is a neuron in a convolution? (2:2)

- Alternatively: Think of each filter as defining an abstract neuron (like a mathematical function).
- Allows us to apply this neuron multiple times.
- Example of weight sharing / parameter tying:
- The parameters are shared for all copies of the neuron.
- Allows us to have lots of neurons while having a relatively small number of parameters to be learned.



Input length

- Conceptually, CNNs are independent of input-length.
- Pooling allows us to represent variable-length input with a fixed-sized vector.
- Naturally deals with e.g. sentences of varying length.
- In practice, however, it is common to pad all inputs to match the maximum input length (or some specified lower cut-off).
- Using some reserved token such as `<PAD>`.
- Main reason; batch computation: Each example in a batch is required to have the same length.

Estimated parameters

- Backpropagation after the final prediction layer.
- Estimates MLP weights, the convolution weights and bias, and (possibly) the embeddings.
- Embedding layer can be: learned from scratch or pre-trained.
- When pre-trained, the embedding layer can be:
- Static: fixed, no backpropagation.
- Dynamic: further trained / fine-tuned.
- CNNs also useful for representation learning!

CNNs and representation learning (1:2)

- Kim (2014) shows the effect of fine-tuning embeddings with a CNN for SA.
- Compares the 4 nearest neighbors of words with static and non-static embeddings.
- Deals with a well-known challenge for distributional semantics:
- Antonyms end up similar.
- Learned task-specific embeddings can be useful beyond the CNN.

Target	Pre-trained	Fine-tuned
bad	good terrible horrible lousy	terrible horrible lousy stupid
good	great bad terrific decent	nice decent solid terrific
n't	os ca ireland wo	not never nothing neither

CNNs and representation learning (2:2)

- A CNN can also be used for creating document embeddings:
- The vectors produced by the pooling layer.
- Yields a fixed-sized representation, independent of input length.
- Similar documents / sentences will have pooling vectors that are close to each other.
- Can be used for retrieval or other document similarity tasks.

Example CNN applications by LT MSc students

- Camilla E. S.: predicting abusive comments expressing threats of violence, using the YouTube Threat Corpus.
- Eivind A. B.: predicting review ratings (1–6) using NoReC (Norwegian Review Corpus).
- Karianne K. A.: will create sentiment lexicons based on embeddings fine-tuned with a CNN for document-level SA classification.
- Mateo C. A.: CNN sentence classification for review summarization.
- Atle O.: CNN for predicting document meta-data, used for learning document representations (the pooling layer) for text retrieval in the Lovdata legal document collection.
- Eivind H. T.: will use them for predicting party affiliations of speeches in the Talk of Norway Corpus of parliamentary proceedings.
- Celina M.: document classification for the Norwegian welfare administration.

Next week: more on CNNs

- More advanced CNN architectures:
  - Hierarchical convolutions
  - Multiple channels
- Overview of the parameter space and design choices
- Tuning (Zhang & Wallace, 2015/2017)
- Use cases.