Lecture 4

Representation Learning with Convolutions

Part 1: Representation Learning

Machine Learning for Structured Data

Vlad Niculae · LTL, UvA · `https://vene.ro/mlsd`

Representation Learning with Convolutions

Representation Learning

So far, we have used hand-crafted representations $\boldsymbol{h}(x)$.

Starting today, we explore deep learning methods that can generate representations of structured data.

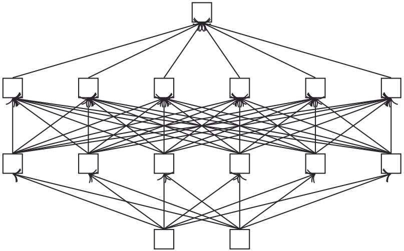
Fully-connected feed-forward layers.

For unstructured data, where we can represent each data point as a fixed-dim vector $\boldsymbol{z}_0 = \boldsymbol{h}_0(\boldsymbol{x}) \in \mathbb{R}^d$, feed-forward network.

The last hidden layer $\boldsymbol{z}_m$ can be seen as a richer vector representation of the data point.

How to handle structured inputs? Often of different sizes?

We will explore architectures that can handle sequences, grids, graphs of different dimensions.



$f(x)$

$\boldsymbol{z}_2 = \phi(\boldsymbol{W}_2\boldsymbol{z}_1 + \boldsymbol{b}_2)$

$\boldsymbol{z}_1 = \phi(\boldsymbol{W}_1\boldsymbol{z}_0 + \boldsymbol{b}_1)$

$\boldsymbol{z}_0 = \boldsymbol{h}_0(x)$

Lecture 4

Representation Learning with Convolutions

Part 2: 1-d convolutions

Machine Learning for Structured Data

Vlad Niculae · LTL, UvA · `https://vene.ro/mlsd`

Representation Learning with Convolutions

Convolutions

When applying a dense linear layer $\boldsymbol{W}\boldsymbol{z} + \boldsymbol{b}$,
the *input* and *output* dimensions must be fixed, because

$$\text{shape}(\boldsymbol{W}) = (d_{\text{out}}, d_{\text{in}})$$

Convolutions

When applying a dense linear layer $\boldsymbol{W}\boldsymbol{z} + \boldsymbol{b}$,
the *input* and *output* dimensions must be fixed, because

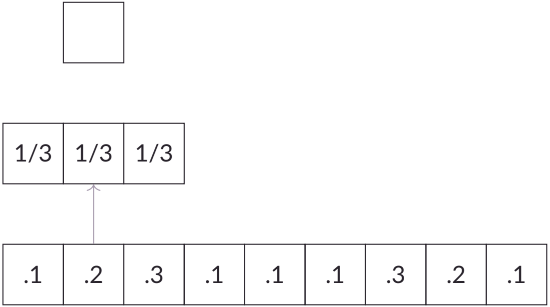
$$\text{shape}(\boldsymbol{W}) = (d_{\text{out}}, d_{\text{in}})$$

Convolutions: what if we learned small linear layers
that we **slide** along an input of variable size.

1-d, single-channel convolution

Simplest case: z_0 is just a sequence of numbers.

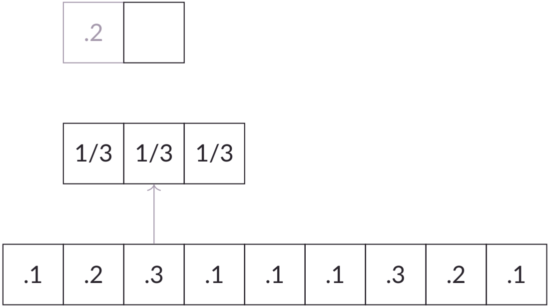
(Example: audio signal processing, time series...)



1-d, single-channel convolution

Simplest case: z_0 is just a sequence of numbers.

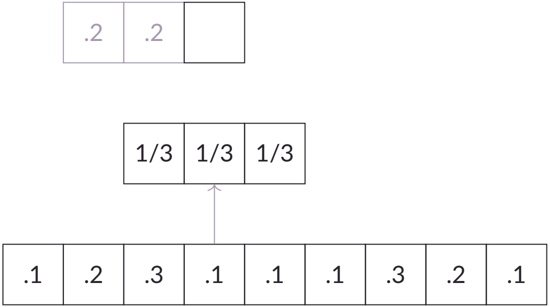
(Example: audio signal processing, time series…)



1-d, single-channel convolution

Simplest case: z_0 is just a sequence of numbers.

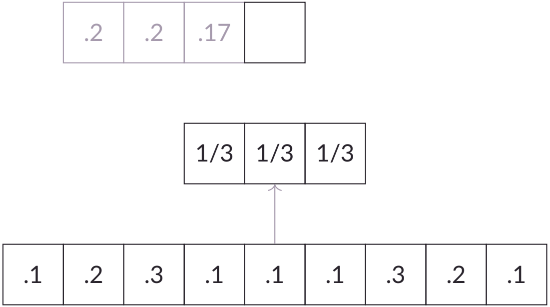
(Example: audio signal processing, time series…)



1-d, single-channel convolution

Simplest case: z_0 is just a sequence of numbers.

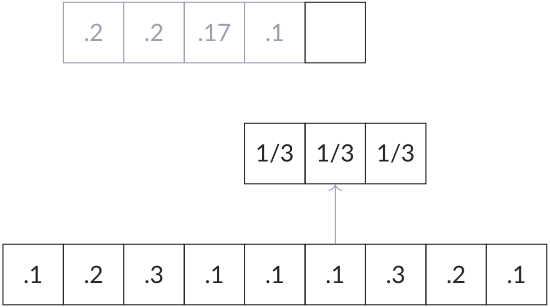
(Example: audio signal processing, time series...)



1-d, single-channel convolution

Simplest case: z_0 is just a sequence of numbers.

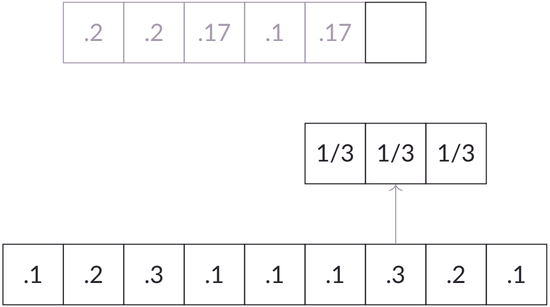
(Example: audio signal processing, time series…)



1-d, single-channel convolution

Simplest case: z_0 is just a sequence of numbers.

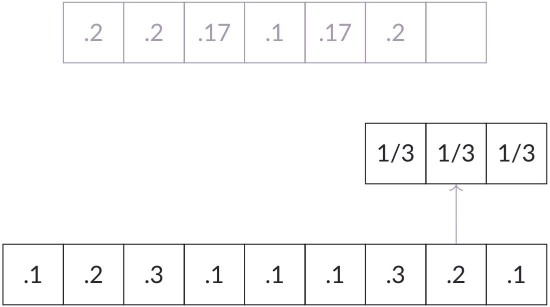
(Example: audio signal processing, time series…)



1-d, single-channel convolution

Simplest case: z_0 is just a sequence of numbers.

(Example: audio signal processing, time series…)



1-d, single-channel convolution

Simplest case: z_0 is just a sequence of numbers.

(Example: audio signal processing, time series…)

.2	.2	.17	.1	.17	.2	.2

.1	.2	.3	.1	.1	.1	.3	.2	.1

1-d, single-channel convolution

Simplest case: z_0 is just a sequence of numbers.

(Example: audio signal processing, time series…)

.2	.2	.17	.1	.17	.2	.2

.1	.2	.3	.1	.1	.1	.3	.2	.1

- Only k parameters, but can apply to seq. of any length.
- Filters are **activated** by patches that match them.
- Since we slide, the **position** of the matching patch doesn't matter. ("translation equivariance")
- Maps an input sequence to an output sequence of (almost) the same length. To make it the same length, we can assume zero padding.

1-d, single-channel convolution

Simplest case: z_0 is just a sequence of numbers.

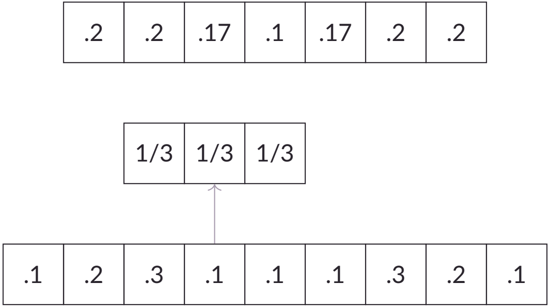
(Example: audio signal processing, time series…)

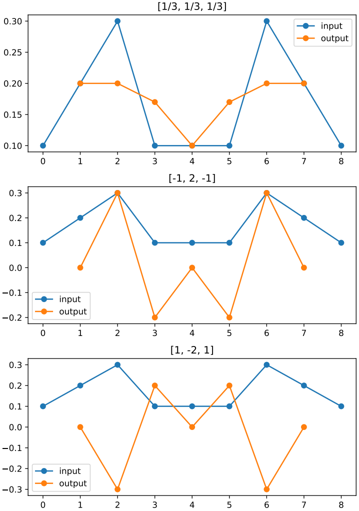
.1	.2	.2	.17	.1	.17	.2	.2	.1

0	.1	.2	.3	.1	.1	.1	.3	.2	.1	0

- Only k parameters, but can apply to seq. of any length.
- Filters are **activated** by patches that match them.
- Since we slide, the **position** of the matching patch doesn't matter. ("translation equivariance")
- Maps an input sequence to an output sequence of (almost) the same length. To make it the same length, we can assume zero padding.

Effects of different convolutional filters





Convolution is sparkling matrix multiplication

- Turns out, convolution is multiplication with a “special” matrix, $\boldsymbol{z}_1 = \boldsymbol{W}\boldsymbol{z}_0$.
- But this $\boldsymbol{W}$ has a very special form that allows it to “stretch” to any size!
- This happens implicitly: such a matrix is never actually built in memory.

$$\begin{bmatrix} .1 \\ .2 \\ .2 \\ .17 \\ .1 \\ .17 \\ .2 \\ .2 \\ .1 \end{bmatrix} = \boldsymbol{W} \begin{bmatrix} .1 \\ .2 \\ .3 \\ .1 \\ .1 \\ .1 \\ .3 \\ .2 \\ .1 \end{bmatrix}$$

Convolution is sparkling matrix multiplication

- Turns out, convolution is multiplication with a "special" matrix, $\boldsymbol{z}_1 = \boldsymbol{W}\boldsymbol{z}_0$.
- But this $\boldsymbol{W}$ has a very special form that allows it to "stretch" to any size!
- This happens implicitly: such a matrix is never actually built in memory.

$$
\begin{bmatrix} .1 \\ .2 \\ .2 \\ .17 \\ .1 \\ .17 \\ .2 \\ .2 \\ .1 \end{bmatrix}
=
\begin{bmatrix}
1/3 & 1/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1/3 & 1/3 & 1/3 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1/3 & 1/3 & 1/3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1/3 & 1/3 & 1/3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1/3 & 1/3 & 1/3 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1/3 & 1/3 & 1/3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1/3 & 1/3 & 1/3 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 1/3 & 1/3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 1/3
\end{bmatrix}
\begin{bmatrix} .1 \\ .2 \\ .3 \\ .1 \\ .1 \\ .1 \\ .3 \\ .2 \\ .1 \end{bmatrix}
$$

Lecture 4

Representation Learning with Convolutions

Part 3: Embedding Discrete Data

Machine Learning for Structured Data

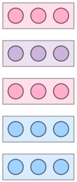
Vlad Niculae · LTL, UvA · `https://vene.ro/mlsd`

Embeddings of Discrete Tokens

Neural networks perform continuous operations.

For sequential **discrete** data, (language, DNA, etc), we must first represent each token as a continuous “embedding” vector.

$$\underbrace{\begin{bmatrix} 3 \\ 2 \\ 3 \\ 1 \\ 1 \end{bmatrix}}_{\in \mathbb{R}^{L}} \rightarrow \underbrace{\begin{bmatrix} \boldsymbol{e}(3) \\ \boldsymbol{e}(2) \\ \boldsymbol{e}(3) \\ \boldsymbol{e}(1) \\ \boldsymbol{e}(1) \end{bmatrix}}_{\in \mathbb{R}^{L \times d}}$$

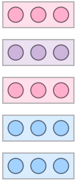


Embeddings of Discrete Tokens

Neural networks perform continuous operations.

For sequential **discrete** data, (language, DNA, etc), we must first represent each token as a continuous "embedding" vector.

$$\underbrace{\begin{bmatrix} 3 \\ 2 \\ 3 \\ 1 \\ 1 \end{bmatrix}}_{\in \mathbb{R}^{L}} \rightarrow \underbrace{\begin{bmatrix} \boldsymbol{e}(3) \\ \boldsymbol{e}(2) \\ \boldsymbol{e}(3) \\ \boldsymbol{e}(1) \\ \boldsymbol{e}(1) \end{bmatrix}}_{\in \mathbb{R}^{L \times d}}$$

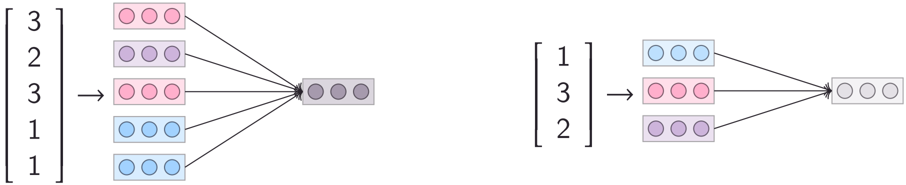


The function $\boldsymbol{e}(i)$ retrieves the ith row from an *embedding matrix* $\mathbf{E} \in \mathbb{R}^{|V| \times d}$.

The embeddings could be fixed or learned as model parameters.

Continuous Bag Of Words

Different-length sequences can be encoded by pooling their embeddings.



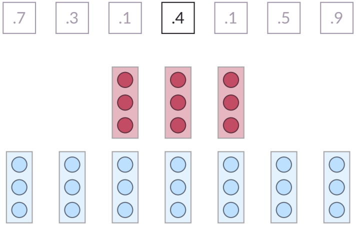
- average pooling: $\boldsymbol{z} = \frac{1}{L}(\boldsymbol{z}_1 + \ldots + \boldsymbol{z}_L)$
- max pooling: $[\boldsymbol{z}]_j = \max([\boldsymbol{z}_1]_j, \ldots, [\boldsymbol{z}_L]_j$ (coordinate-wise)

Just like in the standard bag of words, word order doesn't matter.

Sequence convolutions

aka 1-d convolution with *d* channels

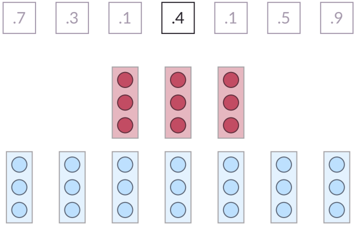
- Denote L=sequence length, d=embedding size, k=window size.



To reduce visual noise on slides, we now use the same color for all words, even if they're different words in general.

Sequence convolutions

aka 1-d convolution with d channels

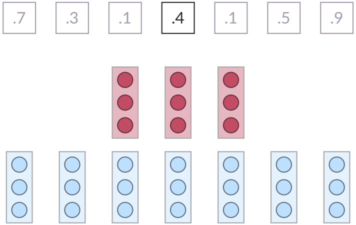


- Denote L=sequence length, d=embedding size, k=window size.
- In the single-channel case, a filter was a dim-k vector. Now, a filter is a $d \times k$ matrix.

To reduce visual noise on slides, we now use the same color for all words, even if they're different words in general.

Sequence convolutions

aka 1-d convolution with d channels

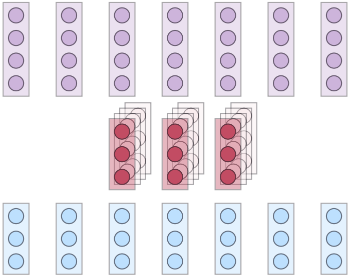


- Denote L=sequence length, d=embedding size, k=window size.
- In the single-channel case, a filter was a dim-k vector. Now, a filter is a $d \times k$ matrix.
- Output is still a single number per window.

To reduce visual noise on slides, we now use the same color for all words, even if they're different words in general.

Sequence convolutions

aka 1-d convolution with d channels



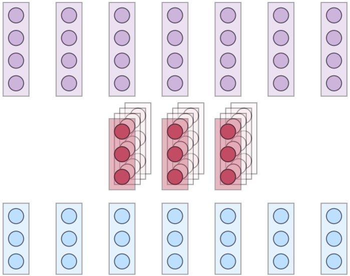
- Denote L=sequence length, d=embedding size, k=window size.
- In the single-channel case, a filter was a dim-k vector. Now, a filter is a $d \times k$ matrix.
- Output is still a single number per window.
- Apply m filters in parallel: output is a dim-m vector per window:

  a “layer” maps $(L, d) \rightarrow (L, m)$, for any L.

To reduce visual noise on slides, we now use the same color for all words, even if they’re different words in general.

Sequence convolutions

aka 1-d convolution with d channels

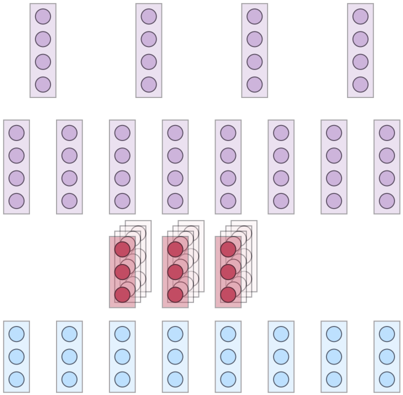


- Denote L=sequence length, d=embedding size, k=window size.
- In the single-channel case, a filter was a dim-k vector. Now, a filter is a $d \times k$ matrix.
- Output is still a single number per window.
- Apply m filters in parallel: output is a dim-m vector per window:

  a "layer" maps $(L, d) \to (L, m)$, for any L.
- Kind of like "continuous" n-grams!

To reduce visual noise on slides, we now use the same color for all words, even if they're different words in general.

Stacking convolutions



- Many different architectures are possible.
- One successful idea: (also in your assignment 1)
  alternate convolutions with (max-)pooling over small windows: hidden representations go from finer (local) to coarser (more global).
- Each (conv + max-pool) layer reduces sequence length by the size of the pooling window.
- After enough layers, pool globally to get a d_{out}-dimensional representation independent of L.

Lecture 4

Representation Learning with Convolutions

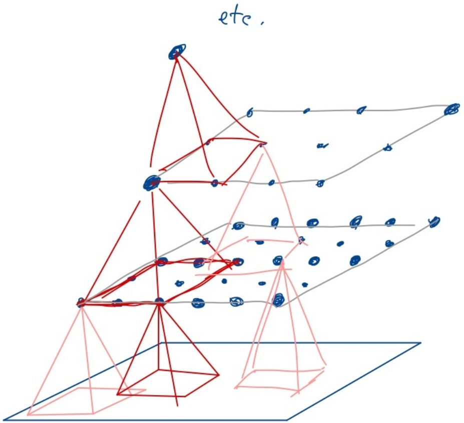
Part 4: 2-d convolution

Machine Learning for Structured Data

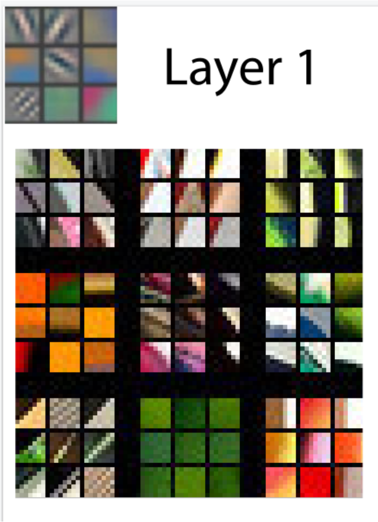
Vlad Niculae · LTL, UvA · `https://vene.ro/mlsd`

Representation Learning with Convolutions

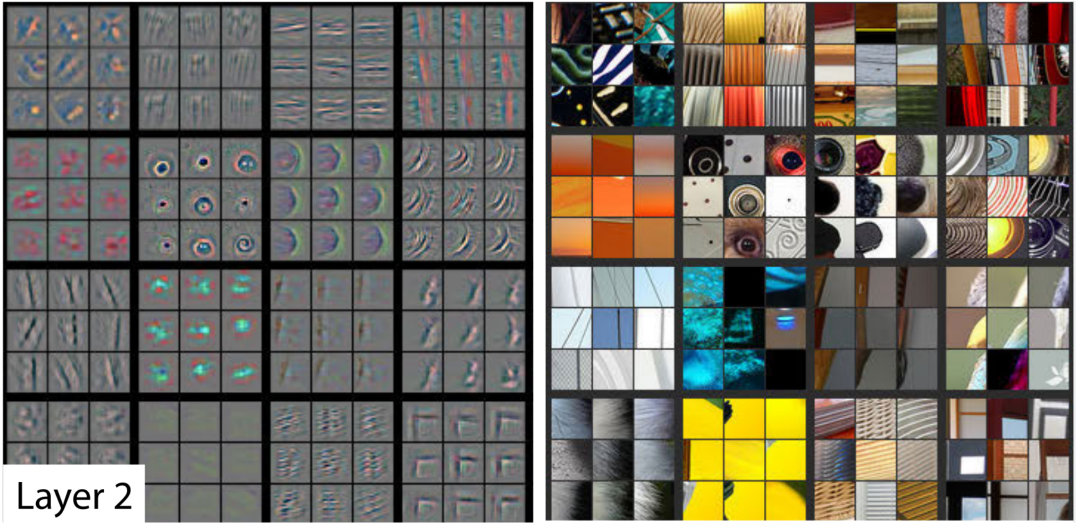
Convolutions for images: 2-d convolution



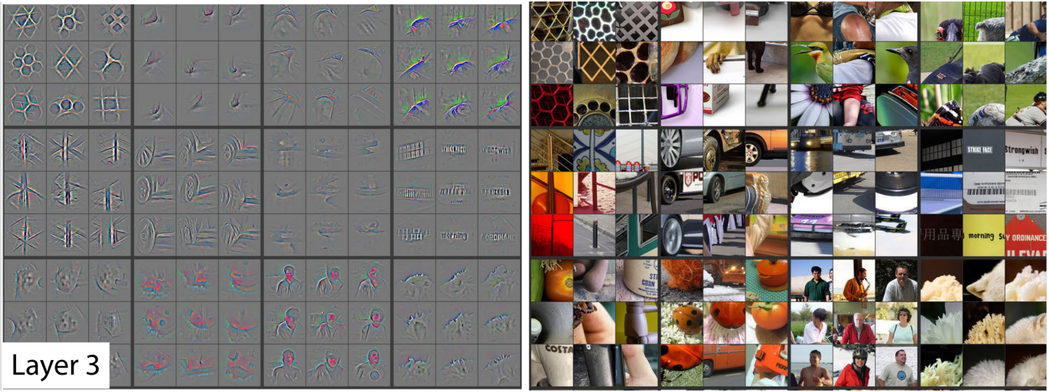
- Instead of just left-to-right, we slide the filter left-to-right top-to-bottom.
- As we go deeper, learned filters become more global/abstract.



From Zeiler and Fergus (2014), “Visualizing and Understanding Convolutional Networks”. ECCV, ©Springer.



From Zeiler and Fergus (2014), “Visualizing and Understanding Convolutional Networks”. ECCV, ©Springer.



From Zeiler and Fergus (2014), “Visualizing and Understanding Convolutional Networks”. ECCV, ©Springer.

Some more practical considerations of convolutions

- Convolutions work great when the phenomenon of interest is fairly local.
- *Strided* convolution: when sliding, skip over a few positions.
  (As long as stride $<$ kernel size $/$ 2, no input positions are ignored.)
- If we want to compute representations of every position (word/pixel) rather than a global representation, there are two options:
  1. no pooling and no striding,
  2. down-sample and then up-sample again ("transpose convolutions"), e.g. "U-net" (Ronneberger et al, 2015).

Convolutions:

1. Representation Learning
2. 1-d convolutions
3. Embedding Discrete Data
4. 2-d convolution