EE-559 – Deep learning

4.4. Convolutions

François Fleuret

https://fleuret.org/ee559/

Mar 10, 2020
If they were handled as normal “unstructured” vectors, large-dimension signals such as sound samples or images would require models of intractable size.

For instance a linear layer taking a $256 \times 256$ RGB image as input, and producing an image of same size would require

$$(256 \times 256 \times 3)^2 \simeq 3.87e+10$$

parameters, with the corresponding memory footprint ($\simeq 150$Gb !), and excess of capacity.
Moreover, this requirement is inconsistent with the intuition that such large signals have some “invariance in translation”. A representation meaningful at a certain location can / should be used everywhere.
Moreover, this requirement is inconsistent with the intuition that such large signals have some “invariance in translation”. **A representation meaningful at a certain location can / should be used everywhere.**

A convolution layer embodies this idea. It applies the same linear transformation locally, everywhere, and preserves the signal structure.
Kernel $W$
Input

\[
\begin{array}{cccccccc}
1 & 4 & -1 & 0 & 2 & -2 & 1 & 3 \\
\end{array}
\]

Kernel

\[
\begin{array}{cccc}
1 & 2 & 0 & -1 \\
\end{array}
\]

Output

\[
\begin{array}{cccccccc}
\end{array}
\]

\[W - w + 1\]
\[ W - w + 1 \]
The diagram illustrates the convolution process with an input array and a kernel. The input array is:

\[
\begin{bmatrix}
1 & 4 & -1 & 0 & 2 & -2 & 1 & 3 & 3 & 1
\end{bmatrix}
\]

The kernel is:

\[
\begin{bmatrix}
1 & 2 & 0 & -1
\end{bmatrix}
\]

The output is:

\[
\begin{bmatrix}
9 & 0
\end{bmatrix}
\]

The formula for the output is:

\[W - w + 1\]
The diagram illustrates the convolution operation on a 1D input and a kernel. The input sequence is:

\[ [1, 4, -1, 0, 2, -2, 1, 3, 3, 1] \]

The kernel sequence is:

\[ [1, 2, 0, -1] \]

The output is calculated as follows:

\[ W - w + 1 \]

where \( W \) is the input and \( w \) is the kernel. The output is:

\[ [9, 0, 1] \]
Francois Fleuret

EE-559 – Deep learning / 4.4. Convolutions

\[ W - w + 1 \]

Input

\[
\begin{array}{cccccccc}
1 & 4 & -1 & 0 & 2 & -2 & 1 & 3 & 3 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 0 & -1 \\
\end{array}
\]

\[ W \]

\[ w \]

Output

\[
\begin{array}{cccc}
9 & 0 & 1 & 3 \\
\end{array}
\]

\[ W - w + 1 \]
\[ W - w + 1 \]

Input:

\[
\begin{array}{cccccccc}
1 & 4 & -1 & 0 & 2 & -2 & 1 & 3 & 3 & 1
\end{array}
\]

Kernel:

\[
\begin{array}{cccc}
1 & 2 & 0 & -1
\end{array}
\]

Output:

\[
\begin{array}{cccc}
9 & 0 & 1 & 3 & -5
\end{array}
\]
\( W - w + 1 \)
Formally, in 1d, given

\[ x = (x_1, \ldots, x_W) \]

and a “convolution kernel” (or “filter”) of width \( w \)

\[ u = (u_1, \ldots, u_w) \]
Formally, in 1d, given 

\[ x = (x_1, \ldots, x_W) \]

and a “convolution kernel” (or “filter”) of width \( w \)

\[ u = (u_1, \ldots, u_w) \]

the convolution \( x \otimes u \) is a vector of size \( W - w + 1 \), with

\[
(x \otimes u)_i = \sum_{j=1}^{w} x_{i-1+j} \ u_j \\
= (x_i, \ldots, x_{i+w-1}) \cdot u
\]

for instance

\[
(1, 2, 3, 4) \otimes (3, 2) = (3 + 4, 6 + 6, 9 + 8) = (7, 12, 17).
\]
Formally, in 1d, given
\[ x = (x_1, \ldots, x_W) \]
and a “convolution kernel” (or “filter”) of width \( w \)
\[ u = (u_1, \ldots, u_w) \]
the convolution \( x \circledast u \) is a vector of size \( W - w + 1 \), with
\[
(x \circledast u)_i = \sum_{j=1}^{w} x_{i-1+j} u_j \\
= (x_i, \ldots, x_{i+w-1}) \cdot u
\]
for instance
\[ (1, 2, 3, 4) \circledast (3, 2) = (3 + 4, 6 + 6, 9 + 8) = (7, 12, 17). \]

⚠️ This differs from the usual convolution since the kernel and the signal are both visited in increasing index order.
Convolution can implement in particular differential operators, e.g.

\[(0, 0, 0, 0, 1, 2, 3, 4, 4, 4) \ast (-1, 1) = (0, 0, 0, 1, 1, 1, 0, 0, 0)\].
Convolution can implement in particular differential operators, e.g.

$$(0, 0, 0, 0, 1, 2, 3, 4, 4, 4) \ast (−1, 1) = (0, 0, 0, 1, 1, 1, 0, 0, 0).$$
Convolution can implement in particular differential operators, e.g.

\[(0, 0, 0, 0, 1, 2, 3, 4, 4, 4) \ast (-1, 1) = (0, 0, 0, 1, 1, 1, 0, 0, 0).\]

or crude “template matcher”, e.g.
Convolution can implement in particular differential operators, e.g.

\[(0, 0, 0, 0, 1, 2, 3, 4, 4, 4) \odot (-1, 1) = (0, 0, 0, 1, 1, 1, 0, 0, 0)\].

or crude "template matcher", e.g.

Both of these computation examples are indeed "invariant by translation".
It generalizes naturally to a multi-dimensional input, although specification can become complicated.
It generalizes naturally to a multi-dimensional input, although specification can become complicated.

Its most usual form for “convolutional networks” processes a 3d tensor as input (i.e. a multi-channel 2d signal) to output a 2d tensor. The kernel is not swiped across channels, just across rows and columns.

In this case, if the input tensor is of size $C \times H \times W$, and the kernel is $C \times h \times w$, the output is $(H - h + 1) \times (W - w + 1)$. 
It generalizes naturally to a multi-dimensional input, although specification can become complicated.

Its most usual form for “convolutional networks” processes a 3d tensor as input (i.e. a multi-channel 2d signal) to output a 2d tensor. The kernel is not swiped across channels, just across rows and columns.

In this case, if the input tensor is of size $C \times H \times W$, and the kernel is $C \times h \times w$, the output is $(H - h + 1) \times (W - w + 1)$.

⚠️ We say “2d signal” even though it has $C$ channels, since it is a feature vector indexed by a 2d location without structure on the feature indexes.
It generalizes naturally to a multi-dimensional input, although specification can become complicated.

Its most usual form for “convolutional networks” processes a 3d tensor as input (i.e. a multi-channel 2d signal) to output a 2d tensor. The kernel is not swiped across channels, just across rows and columns.

In this case, if the input tensor is of size $C \times H \times W$, and the kernel is $C \times h \times w$, the output is $(H - h + 1) \times (W - w + 1)$.

We say “2d signal” even though it has $C$ channels, since it is a feature vector indexed by a 2d location without structure on the feature indexes.

In a standard convolution layer, $D$ such convolutions are combined to generate a $D \times (H - h + 1) \times (W - w + 1)$ output.
Kernel $K$:

- $D - h + 1$
- $W - w + 1$

Output:

Input

- $W$
- $H$
- $C$
\[ D - h + 1 \times W - w + 1 \]
The diagram illustrates the process of convolution in deep learning. The **Input** is represented on the left, showing a 3D tensor with dimensions $H$, $W$, and $C$. The **Kernel** is depicted in the middle, with dimensions $h$, $w$, and $C$. The **Output** is shown on the right. The convolution operation involves sliding the kernel across the input tensor and performing elementwise multiplication and summing the results.
Kernel

\[ D - h + 1 \times W - w + 1 \]

Input

Output

Kernel

\[ w \times h \times C \]

\[ C \]
\[ Kernels = D - h + 1 \times W - w + 1 \]

Input

Output
The formula for the output size of a convolution operation is given by:

\[ D - h + 1 \times W - w + 1 \]

where:
- \( D \) is the input depth,
- \( W \) is the input width,
- \( h \) is the kernel height,
- \( w \) is the kernel width.

This formula is used to determine the size of the output volume after applying a convolution operation.
\[ Kernels \]

\[ D \times H - h + 1 \times W - w + 1 \]
Kernel $K$:

$D 	imes H 	imes W 	imes C$

$1 	imes 1 	imes 1 	imes C$

Input

Kernel

Output
The convolution operation can be expressed as:

\[
(D - h + 1) \times (W - w + 1)\]

where:
- \(D\) and \(W\) are the dimensions of the input tensor.
- \(h\) and \(w\) are the dimensions of the kernel.
- \(C\) is the number of channels in the input and output tensors.
Françcois Fleuret

EE-559 – Deep learning / 4.4. Convolutions
Kernels

$$D - h + 1 \quad W - w + 1$$

Input

Output

Kernel

Françoise Fleuret
Kernel

\[
D - h + 1 \quad W - w + 1
\]

\[
H - h + 1
\]
Kernel:

\[ Kernels \]

Input:

\[ \text{Input} \]

\[ \begin{align*}
  D & \quad H - h + 1 \\
  W & \quad W - w + 1 \\
  C & \quad C
\end{align*} \]

Output:

\[ \text{Output} \]

\[ \begin{align*}
  D & \quad D \\
  W & \quad W - w + 1 \\
  H & \quad H - h + 1 \\
  C & \quad C
\end{align*} \]
Input

Linear

Output

\[ W - w + 1 \]

\[ H - h + 1 \]
Note that a convolution preserves the signal support structure.

A 1d signal is converted into a 1d signal, a 2d signal into a 2d, and neighboring parts of the input signal influence neighboring parts of the output signal.
Note that a convolution preserves the signal support structure.

A 1d signal is converted into a 1d signal, a 2d signal into a 2d, and neighboring parts of the input signal influence neighboring parts of the output signal.

A 3d convolution can be used if the channel index has some metric meaning, such as time for a series of grayscale video frames. Otherwise swiping across channels makes no sense.
We usually refer to one of the channels generated by a convolution layer as an **activation map**.

The sub-area of an input map that influences a component of the output as the **receptive field** of the latter.
We usually refer to one of the channels generated by a convolution layer as an activation map.

The sub-area of an input map that influences a component of the output as the receptive field of the latter.

In the context of convolutional networks, a standard linear layer is called a **fully connected layer** since every input influences every output.
F.conv2d(input, weight, bias=None, stride=1, padding=0, dilation=1, groups=1)

Implements a 2d convolution, where \texttt{weight} contains the kernels, and is \(D \times C \times h \times w\), \texttt{bias} is of dimension \(D\), \texttt{input} is of dimension \(N \times C \times H \times W\)

and the result is of dimension

\(N \times D \times (H - h + 1) \times (W - w + 1).\)
F.conv2d(input, weight, bias=None, stride=1, padding=0, dilation=1, groups=1)

Implements a 2d convolution, where \( \text{weight} \) contains the kernels, and is \( D \times C \times h \times w \), \( \text{bias} \) is of dimension \( D \), \( \text{input} \) is of dimension \( N \times C \times H \times W \)

and the result is of dimension

\[
N \times D \times (H - h + 1) \times (W - w + 1).
\]

```python
>>> weight = torch.empty(5, 4, 2, 3).normal_()
>>> bias = torch.empty(5).normal_()
>>> input = torch.empty(117, 4, 10, 3).normal_()
>>> output = F.conv2d(input, weight, bias)
>>> output.size()
torch.Size([117, 5, 9, 1])
```
F.conv2d(input, weight, bias=None, stride=1, padding=0, dilation=1, groups=1)

Implements a 2d convolution, where `weight` contains the kernels, and is $D \times C \times h \times w$, `bias` is of dimension $D$, `input` is of dimension

$$N \times C \times H \times W$$

and the result is of dimension

$$N \times D \times (H - h + 1) \times (W - w + 1).$$

```python
>>> weight = torch.empty(5, 4, 2, 3).normal_(
>>> bias = torch.empty(5).normal_(
>>> input = torch.empty(117, 4, 10, 3).normal_(
>>> output = F.conv2d(input, weight, bias)
>>> output.size()  
torch.Size([117, 5, 9, 1])
```

Similar functions implement 1d and 3d convolutions.
```python
x = mnist_train.data[12].float().view(1, 1, 28, 28)

weight = torch.empty(5, 1, 3, 3)

weight[0, 0] = torch.tensor(
    [[0., 0., 0.],
     [0., 1., 0.],
     [0., 0., 0.]]
)

weight[1, 0] = torch.tensor(
    [[1., 1., 1.],
     [1., 1., 1.],
     [1., 1., 1.]]
)

weight[2, 0] = torch.tensor(
    [[-1., 0., 1.],
     [-1., 0., 1.],
     [-1., 0., 1.]]
)

weight[3, 0] = torch.tensor(
    [[-1., -1., -1.],
     [0., 0., 0.],
     [1., 1., 1.]]
)

weight[4, 0] = torch.tensor(
    [[0., -1., 0.],
     [-1., 4., -1.],
     [0., -1., 0.]]
)

y = F.conv2d(x, weight)
```
Français Fleuret

EE-559 – Deep learning / 4.4. Convolutions
Francois Fleuret

EE-559 – Deep learning / 4.4. Convolutions
class torch.nn.Conv2d(in_channels, out_channels,
    kernel_size, stride=1, padding=0, dilation=1,
    groups=1, bias=True)

Wraps the convolution into a Module, with the kernels and biases as Parameter properly randomized at creation.

The kernel size is either a pair \((h, w)\) or a single value \(k\) interpreted as \((k, k)\).
class torch.nn.Conv2d(in_channels, out_channels,
    kernel_size, stride=1, padding=0, dilation=1,
    groups=1, bias=True)

Wraps the convolution into a Module, with the kernels and biases as Parameter properly randomized at creation.

The kernel size is either a pair \((h, w)\) or a single value \(k\) interpreted as \((k, k)\).

```python
>>> f = nn.Conv2d(in_channels = 4, out_channels = 5, kernel_size = (2, 3))
>>> for n, p in f.named_parameters(): print(n, p.size())
...
weight torch.Size([5, 4, 2, 3])
bias torch.Size([5])
```
Padding, stride, and dilation
Convolutions have three additional parameters:

- The **padding** specifies the size of a zeroed frame added around the input,
- the **stride** specifies a step size when moving the kernel across the signal,
- the **dilation** modulates the expansion of the filter without adding weights.
Here with $C \times 3 \times 5$ as input

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
\hline
& & & & & & & & \\
\hline & & & & & & & & \\
\hline & & & & & & & & \\
\hline & & & & & & & & \\
\hline & & & & & & & & \\
\hline & & & & & & & & \\
\hline & & & & & & & & \\
\hline & & & & & & & & \\
\hline
\end{array}
\]
Here with $C \times 3 \times 5$ as input, a padding of $(2, 1)$
Here with \( C \times 3 \times 5 \) as input, a padding of \( (2, 1) \), a stride of \( (2, 2) \)
Here with $C \times 3 \times 5$ as input, a padding of $(2, 1)$, a stride of $(2, 2)$, and a kernel of size $C \times 3 \times 3$.
Here with $C \times 3 \times 5$ as input, a padding of $(2, 1)$, a stride of $(2, 2)$, and a kernel of size $C \times 3 \times 3$. 

![Image of convolution process](image-url)
Here with $C \times 3 \times 5$ as input, a padding of $(2, 1)$, a stride of $(2, 2)$, and a kernel of size $C \times 3 \times 3$. 

![Diagram of convolution process]
Here with $C \times 3 \times 5$ as input, a padding of $(2,1)$, a stride of $(2,2)$, and a kernel of size $C \times 3 \times 3$.
Here with $C \times 3 \times 5$ as input, a padding of $(2,1)$, a stride of $(2,2)$, and a kernel of size $C \times 3 \times 3$. 

![Input and Output Diagram]

Input

Output
Here with $C \times 3 \times 5$ as input, a padding of $(2, 1)$, a stride of $(2, 2)$, and a kernel of size $C \times 3 \times 3$. 

Input 

Output
Here with $C \times 3 \times 5$ as input, a padding of $(2,1)$, a stride of $(2,2)$, and a kernel of size $C \times 3 \times 3$. 

![Diagram showing convolution process]
Here with $C \times 3 \times 5$ as input, a padding of $(2, 1)$, a stride of $(2, 2)$, and a kernel of size $C \times 3 \times 3$
Here with $C \times 3 \times 5$ as input, a padding of (2, 1), a stride of (2, 2), and a kernel of size $C \times 3 \times 3$, the output is $1 \times 3 \times 3$.
A convolution with a stride greater than 1 may not cover the input map completely, hence may ignore some of the input values.
The dilation modulates the expansion of the filter support by adding rows and columns of zeros between coefficients (Yu and Koltun, 2015).

It is 1 for standard convolutions, but can be greater, in which case the resulting operation can be envisioned as a convolution with a regularly sparsified filter.
The dilation modulates the expansion of the filter support by adding rows and columns of zeros between coefficients (Yu and Koltun, 2015).

It is 1 for standard convolutions, but can be greater, in which case the resulting operation can be envisioned as a convolution with a regularly sparsified filter.

This notion comes from signal processing, where it is referred to as *algorithm à trous*, hence the term sometime used of “convolution à trous”.
Dilation = 1
Dilation = 1
Dilation = 1

Input

Output
Dilation = 1
Dilation = 1

Input

Output
Dilation = 1

Input

Output
Dilation $= 2$

Input

Output
Dilation = 2

Input

Output
Dilation = 2

Input

Output
Dilation = 2
Dilation = 2

Input

Output
Dilation = 2

Input

Output
Dilation = 2

Input

Output
Dilation = 2
A convolution with a kernel of size $k$ and dilation $d$ can be interpreted as a convolution with a filter of size $1 + (k - 1)d$ with only $k$ non-zero coefficients.

For example with $k = 3$ and $d = 4$, the difference between the input map size and the output map size is $1 + (3 - 1)4 - 1 = 8$.

```python
>>> x = torch.empty(1, 1, 20, 30).normal_()
>>> l = nn.Conv2d(1, 1, kernel_size = 3, dilation = 4)
>>> l(x).size()
torch.Size([1, 1, 12, 22])
```
Having a dilation greater than one increases the units’ receptive field size without increasing the number of parameters.

Convolutions with stride or dilation strictly greater than one reduce the activation map size, for instance to make a final classification decision.
The end
References