Convolutional neural networks

Toby Dylan Hocking
toby.hocking@nau.edu
toby.hocking@r-project.org

November 22, 2023

Supervised learning setup

- Have an input $\mathbf{x} \in \mathbb{R}^d$ – a vector of d real numbers.
- And an output y (real number: regression, integer ID: classification).
- Want to learn a prediction function $f(\mathbf{x}) = y$ that will work on a new input.
- In a neural network with $L - 1$ hidden layers the function f is defined using composition of L functions, $f(x) = f^{(L)}[\cdots f^{(1)}[x]] \in \mathbb{R}$.

Each function is matrix multiplication and activation

- Prediction function $f(x) = f^{(L)}[\cdots f^{(1)}[x]] \in \mathbb{R}$.
- Each function $l \in \{1, \dots, L\}$ is a matrix multiplication followed by an activation function: $f^{(l)}[z] = \sigma^{(l)}[W^{(l)}z]$ where $W^{(l)} \in \mathbb{R}^{u^{(l)} \times u^{(l-1)}}$ is a weight matrix to learn, and $z \in \mathbb{R}^{u^{(l-1)}}$ is the input vector to that layer.
- So far we have only discussed fully connected networks, which means that each entry of the weight matrix is unique and non-zero.
- This week we discuss "convolutional" networks which are useful for spatial data and can be interpreted as multiplication by a special kind of matrix (with sparsity and weight sharing).

Convolution is a linear operator for spatial data

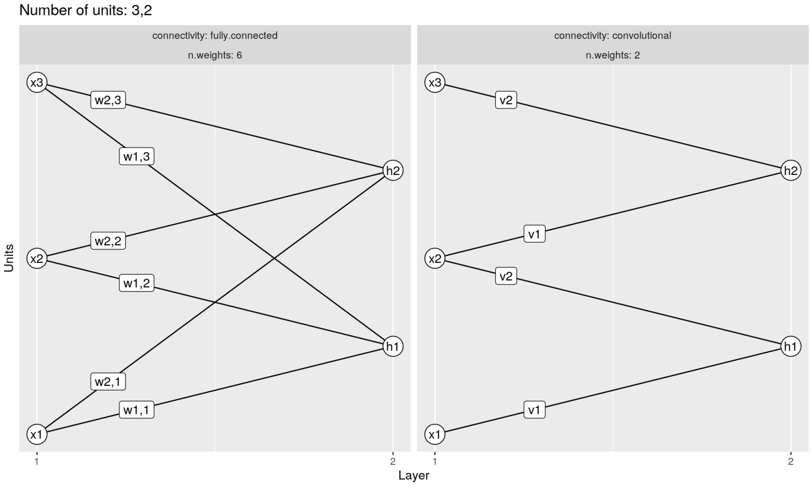
Useful for data which have spatial dimension(s) such as time series (1 dim) or images (2 dim). Simple example with 1 dim:

- $\mathbf{x} = [x_1, \dots, x_D]$ is an input vector (array of D data).
- $\mathbf{v} = [v_1, \dots, v_P]$ is a kernel (array of P parameters / weights to learn), $P < D$.
- $\mathbf{h} = [h_1, \dots, h_U]$ is an output vector of $U = D - P + 1$ hidden units. Convolution (actually cross-correlation) is used to define each hidden unit: $\forall u \in \{1, \dots, U\}, h_u = \sum_{p=1}^{P} v_p x_{u+p-1}$.
- EX: $D = 3$ inputs, $P = 2$ parameters $\Rightarrow U = 2$ output units:

$$
\begin{aligned}
h_1 &= v_1 x_1 + v_2 x_2 \text{ (convolutional=sparse+shared)} \\
h_2 &= v_1 x_2 + v_2 x_3 \text{ (convolutional=sparse+shared)} \\
h_1 &= w_{1,1} x_1 + w_{1,2} x_2 + w_{1,3} x_3 \text{ (fully connected/dense)} \\
h_2 &= w_{2,1} x_1 + w_{2,2} x_2 + w_{2,3} x_3 \text{ (fully connected/dense)}
\end{aligned}
$$

- Compare with fully connected – convolutional means weights are shared among outputs, and some are zero/sparse.

Difference in connectivity and weight sharing



Matrix interpretation of convolution

$$\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \text{ (fully connected)}$$

$$\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & 0 \\ 0 & v_1 & v_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \text{ (convolutional)}$$

- Weight sharing: same weights used to compute different output units.
- Sparsity: zeros in weight matrix.

```
torch.nn.Conv1d(
  in_channels=1,
  out_channels=1,
  kernel_size=2)
```

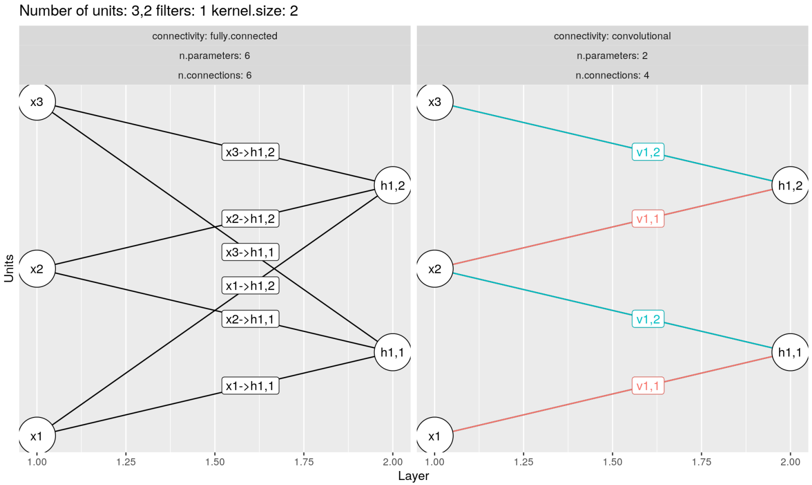
Multiple kernels/filters or sets of weights

- $x = [x_1, \ldots, x_D]$ is an input vector (array of D data).
- $v = \begin{bmatrix} v_{1,1} & \cdots & v_{1,P} \\ \vdots & \ddots & \vdots \\ v_{K,1} & \cdots & v_{K,P} \end{bmatrix}$ is a matrix of K kernels, each row is an array of P parameters / weights to learn, $P < D$.
- $h = \begin{bmatrix} h_{1,1} & \ldots & h_{1,U} \\ \vdots & \ddots & \vdots \\ h_{K,1} & \cdots & h_{K,U} \end{bmatrix}$ is an output matrix of hidden units. Each row is computing by applying a kernel to the input:

$$\forall u \in \{1, \ldots, U\}, \forall k \in \{1, \ldots, K\}, h_{k,u} = \sum_{p=1}^{P} v_{k,p} x_{u+p-1}$$

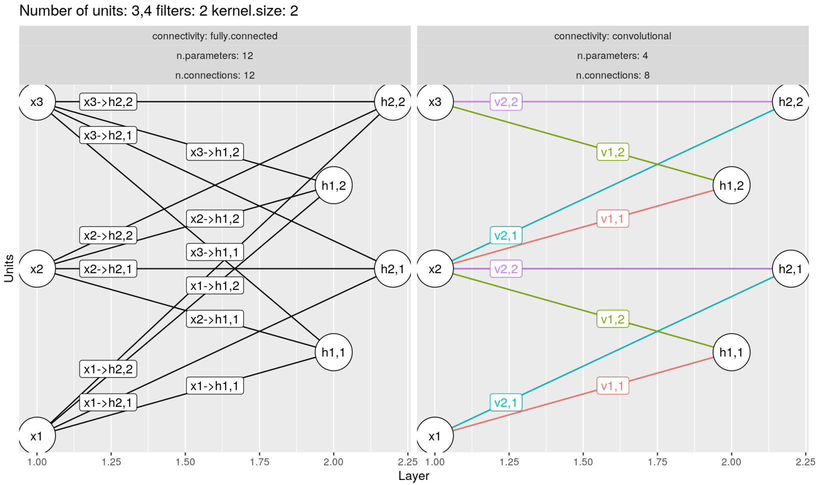
- EX in previous slide: $D = 3$ inputs, $P = 2$ parameters per kernel, $K = 2$ kernels $\Rightarrow U = 2$ output units per kernel, 4 output units total.

Colors for unique weight parameters to learn




```
torch.nn.Conv1d(in_channels=1,
  out_channels=1, kernel_size=2)
```

Colors for unique weight parameters to learn




```
torch.nn.Conv1d(in_channels=1,
  out_channels=2, kernel_size=2)
```

Matrix interpretation of convolution

$$\begin{bmatrix} h_{1,1} \\ h_{1,2} \\ h_{2,1} \\ h_{2,2} \end{bmatrix} = \begin{bmatrix} v_{1,1} & v_{1,2} & 0 \\ 0 & v_{1,1} & v_{1,2} \\ v_{2,1} & v_{2,2} & 0 \\ 0 & v_{2,1} & v_{2,2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \text{ (convolutional)}$$

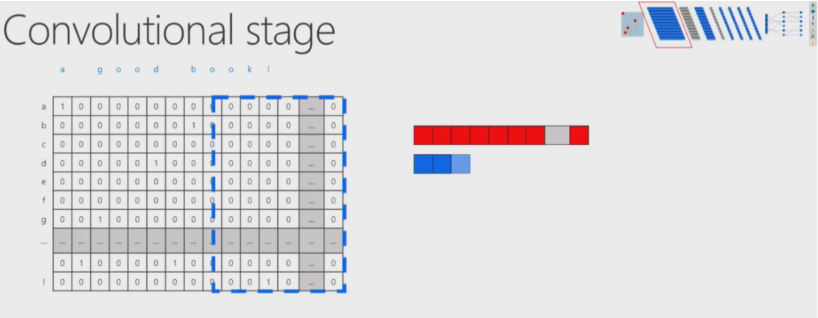
- Weight sharing: same weights used to compute different output units.
- Sparsity: zeros in weight matrix.

```
torch.nn.Conv1d(
  in_channels=1,
  out_channels=2,
  kernel_size=2)
```

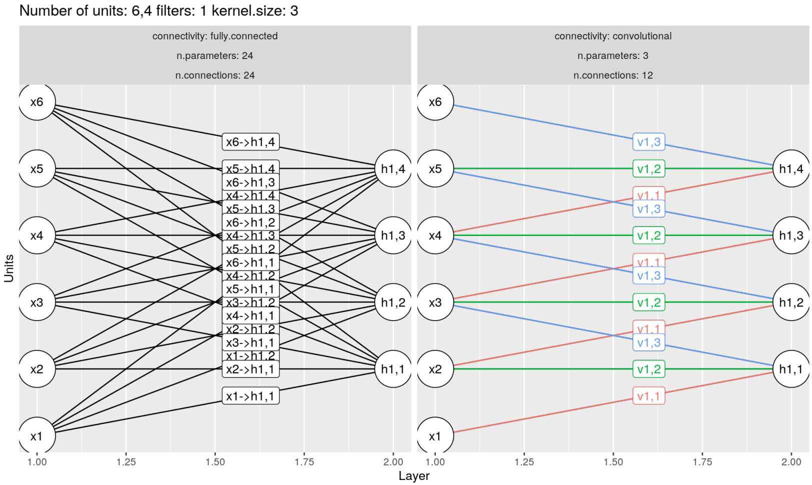
Video about convolution

`https://github.com/tdhock/useR2017-debrief`

Angus Taylor's talk at useR 2017.

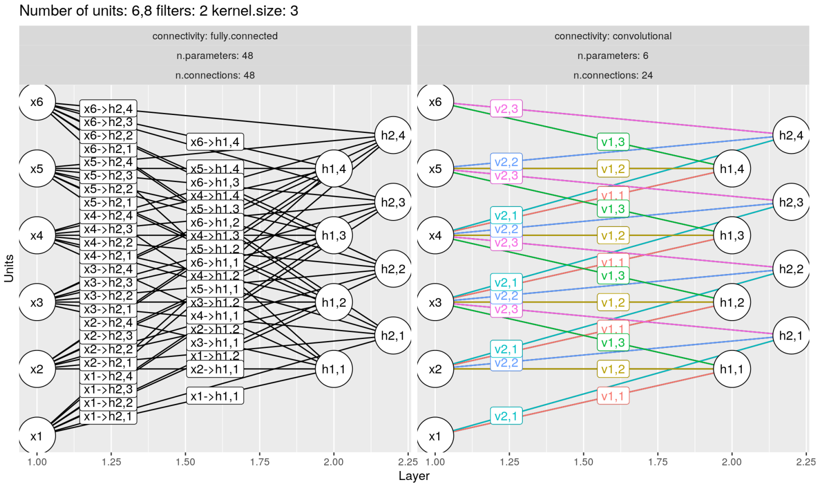


A more complex example (one filter)




```
torch.nn.Conv1d(in_channels=1,
  out_channels=1, kernel_size=3)
```

A more complex example (two filters)




```
torch.nn.Conv1d(in_channels=1,
  out_channels=2, kernel_size=3)
```

Architecture exercises

1D Convolution: if there are $D = 10$ inputs and $U = 5$ outputs,

- how many parameters to learn in a fully connected layer?
- a single kernel in a convolutional layer,
    1. how many parameters are there to learn?
    2. how many connections in the network diagram representation?
    3. how many zeros in the weight matrix representation?

2D Convolution: if you have a 10 x 10 pixel input image, and you apply 5 x 5 kernel,

1. How many parameters are there to learn in each filter?
2. How many parameters total if there are 20 filters?
3. How many output units per filter?
4. How many output units total using 20 filters?

Computation exercises (forward propagation)

$$
\begin{aligned}
\mathbf{x} &= \begin{bmatrix} 0 \\ 3 \\ 10 \end{bmatrix} \\
\mathbf{w} &= \begin{bmatrix} -1 & 2 & -3 \\ 4 & -5 & 6 \end{bmatrix} \\
\mathbf{v} &= \begin{bmatrix} -2 \\ 1 \end{bmatrix}
\end{aligned}
$$

If $\mathbf{x}$ is an input vector,

1. what is the output vector when using $\mathbf{w}$ as the weight matrix in a fully connected layer?
2. what is the output vector when using $\mathbf{v}$ as the kernel in a 1d convolutional layer?

Pooling

Typical order of application in a layer is

1. Weight matrix multiplication (learned via gradient descent).
2. Activation, nonlinear function (not learned).
3. Pooling, reducing the number of units (not learned).

What is pooling?

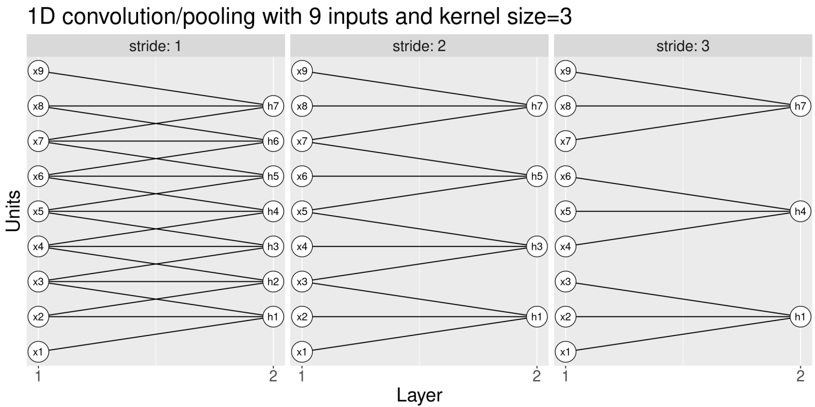
- Main purpose: reducing time/space during learning/prediction.
- Like convolution in that you apply some operation in a window over inputs; each window creates a single output unit.
- In convolution the operation is **multiplication** of inputs in window and corresponding weights, then **addition** to combine results in window.
- In pooling the operation is **mean or max** over all inputs in window.
- Pooling typically used over spatial dimension (independently for each channel/filter).

Stride

- This is the offset between windows on which the convolution/pooling is computed.
- Another technique for reducing number of output units and therefore time/space required during learning/prediction.
- In previous slides we were using stride of 1 (adjacent windows overlap each other and have redundant information).
- Often stride is set to kernel size (no overlapping windows) — this is the default in torch.

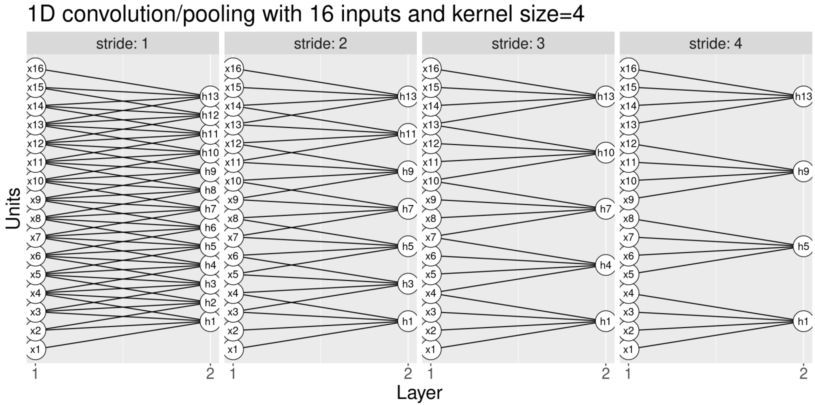
```
torch.nn.MaxPool1d(kernel_size=2, stride=2)
```

Stride diagram



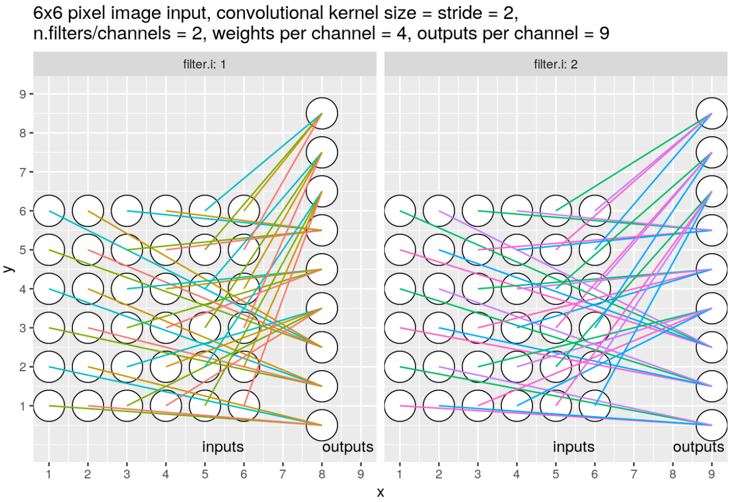

```
torch.nn.MaxPool1d(kernel_size=3, stride=1)
torch.nn.MaxPool1d(kernel_size=3, stride=2)
torch.nn.MaxPool1d(kernel_size=3, stride=3)
```

Stride diagram



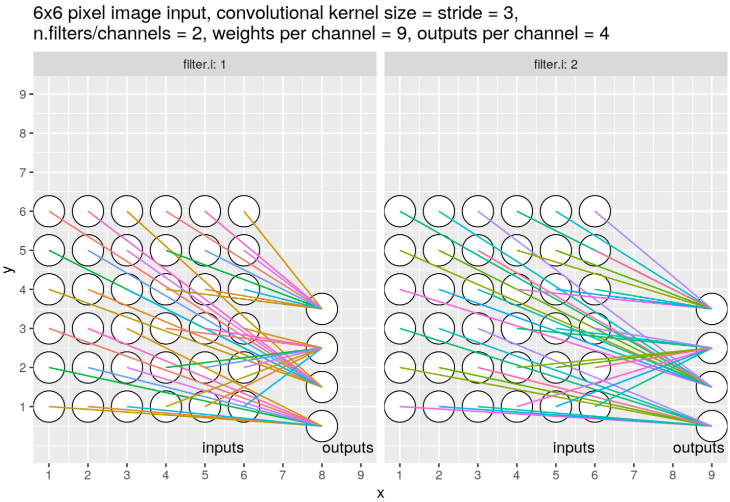

```
torch.nn.MaxPool1d(kernel_size=4, stride=1)
torch.nn.MaxPool1d(kernel_size=4, stride=2)
torch.nn.MaxPool1d(kernel_size=4, stride=3)
torch.nn.MaxPool1d(kernel_size=4, stride=4)
```

2D convolutional kernel for 6x6 pixel image, kernel size=2




```
torch.nn.MaxPool2d(kernel_size=2, stride=2)
torch.nn.Conv2d(kernel_size=2, stride=2)
```

2D convolutional kernel for 6x6 pixel image, kernel size=3




```
torch.nn.MaxPool2d(kernel_size=3, stride=3)
torch.nn.Conv2d(kernel_size=3, stride=3)
```

Architecture exercises

1D Convolution: if there are $D = 20$ inputs and you have a kernel of size 5 with stride 5,

1. how many parameters are there to learn?
2. how many output units are there?
3. how many connections in the network diagram representation?
4. how many zeros in the weight matrix representation?

2D Pooling: if you have a 10 x 10 pixel input image, and you apply a 5 x 5 max pooling kernel with stride 5, how many output units are there?

Computation exercises

If $\mathbf{x} = [0, 3, 10, -2, 5, 1]$ is an input vector,
and $\mathbf{k} = [-2, 1]$ is a kernel,

1. what is the output vector when doing mean pooling with a stride of 2 and kernel of size 2?
2. what is the output vector when doing max pooling with a stride of 3 and kernel of size 3?
3. what is the output vector when using **k** as the kernel in a 1d convolutional layer with a stride of 2?