Deep Learning

5. Convolutional Neural Networks (CNNs)

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Computer Science
University of Hildesheim, Germany

Syllabus

Tue. 21.4.	(1)	1. Supervised Learning (Review 1)
Tue. 28.4.	(2)	2. Neural Networks (Review 2)
Tue. 5.5.	(3)	3. Regularization for Deep Learning
Tue. 12.5.	(4)	4. Optimization for Training Deep Models
Tue. 19.5.	(5)	5. Convolutional Neural Networks
Tue. 26.5.	(6)	6. Recurrent Neural Networks
Tue. 2.6.	—	— *Pentecoste Break* —
Tue. 9.6.	(7)	7. Autoencoders
Tue. 16.6.	(8)	8. Generative Adversarial Networks
Tue. 23.6.	(9)	9. Recent Advances
Tue. 30.6.	(10)	10. Engineering Deep Learning Models
Tue. 7.7.	(11)	tbd.
Tue. 14.7.	(12)	Q & A

Outline

1. Convolutions
2. Ordered vs Unordered Dimensions
3. Convolutional Neural Networks
4. Convolutional Layers vs Fully Connected Layers
5. Reducing Resolutions: Pooling and Striding
6. Outlook

Outline

1. Convolutions

2. Ordered vs Unordered Dimensions

3. Convolutional Neural Networks

4. Convolutional Layers vs Fully Connected Layers

5. Reducing Resolutions: Pooling and Striding

6. Outlook

Convolutions

- given two functions $f, g : \mathbb{R}^N \rightarrow \mathbb{R}$,
  define a third function with the same signature:

$$h := (f * g) : \mathbb{R}^N \rightarrow \mathbb{R},$$

$$h(x) := (f * g)(x) = \int_{\mathbb{R}^N} f(x')g(x - x')dx' = \int_{\mathbb{R}^N} f(x + x')g(-x')dx'$$

- example 1: averaging:
  - $f : \mathbb{R} \rightarrow \mathbb{R}$ a signal in time
  - $g : \mathbb{R} \rightarrow \mathbb{R}$: $g(x) := \frac{1}{2}\mathbb{I}(x \in [-1, 1])$

  $\rightsquigarrow$ $h(x)$ is $f(x')$ averaged over $x' \in [x - 1, x + 1]$

- example 2: correlating:
  - $f : \mathbb{R} \rightarrow \mathbb{R}$ a signal in time
  - $g : \mathbb{R} \rightarrow \mathbb{R}$ a pattern of interest (encoded backwards in time)

  $\rightsquigarrow$ $h(x)$ how similar signal f is at position x to pattern g

Convolutions / Basic Properties

commutative:

$$f * g = g * f$$

associative:

$$f * (g * h) = (f * g) * h$$

distributive:

$$f * (g + h) = (f * g) + (f * h)$$

differentiation:

$$\frac{\partial(f * g)}{\partial x_n} = \frac{\partial f}{\partial x_n} * g = f * \frac{\partial g}{\partial x_n}$$

integration:

$$\int_{\mathbb{R}^N} (f * g)(x) dx = (\int_{\mathbb{R}^N} f(x) dx)(\int_{\mathbb{R}^N} g(x) dx)$$

convolution theorem ($\mathcal{F}$ the Fourier transform):

$$\mathcal{F}(f * g) = \mathcal{F}(f) \cdot \mathcal{F}(g)$$

Discrete Convolutions

- continuous:
  given two functions $f, g : \mathbb{R}^N \to \mathbb{R}$,
  define a third function with the same signature:

$$h := (f * g) : \mathbb{R}^N \to \mathbb{R},$$
$$h(x) := (f * g)(x) = \int_{\mathbb{R}^N} f(x')g(x - x')dx' = \int_{\mathbb{R}^N} f(x + x')g(-x')dx'$$

- discrete:
  given two functions $f, g : \mathbb{Z}^N \to \mathbb{R}$ on a grid,
  define a third function with the same signature:

$$h := (f * g) : \mathbb{Z}^N \to \mathbb{R},$$
$$h(x) := (f * g)(x) = \sum_{x' \in \mathbb{Z}^N} f(x')g(x - x') = \sum_{x' \in \mathbb{Z}^N} f(x + x')g(-x')$$

Discrete Convolutions

- discrete:
  given two functions $f, g : \mathbb{Z}^N \rightarrow \mathbb{R}$ on a grid,
  define a third function with the same signature:

$$h := (f * g) : \mathbb{Z}^N \rightarrow \mathbb{R},$$
$$h(x) := (f * g)(x) = \sum_{x' \in \mathbb{Z}^N} f(x')g(x - x') = \sum_{x' \in \mathbb{Z}^N} f(x + x')g(-x')$$

  - in computer science, reading the second function backwards usually is not done:

$$h(x) := (f * g)(x) = \sum_{x' \in \mathbb{Z}^N} f(x + x')g(x')$$

Finite Discrete Convolutions

- finite discrete:
  given two arrays $f \in \mathbb{R}^{N \times M}, g \in \mathbb{R}^{\tilde{N} \times \tilde{M}}$,
  define a third array with the dimensions:
  $$h := (f * g) \in \mathbb{R}^{N \times M}$$
  $$h_{n,m} := (f * g)_{n,m} = \sum_{n'=1}^{\tilde{N}} \sum_{m'=1}^{\tilde{M}} f(n + \delta n', m + \delta m') g(n', m')$$

  - $\delta n' := \delta(n', \tilde{N}) := n' - \lfloor \frac{\tilde{N}+1}{2} \rfloor$ **index centering**
    - e.g., $\tilde{N} = 5 \quad \rightsquigarrow \quad \delta n' = n' - 3$: $\delta n' = -2, -1, 0, 1, 2$ for $n' = 1, 2, \ldots, 5$.
      $\tilde{N} = 6 \quad \rightsquigarrow \quad \delta n' = n' - 3$: $\delta n' = -2, -1, 0, 1, 2, 3$ for $n' = 1, 2, \ldots,$
  - $f(n, m) := 0$ for $n < 1$, $n \geq N$, $m < 1$ or $m \geq M$ (**zero padding**)

Note: Here for two-dimensional arrays. The same works for any dimensional arrays.

Finite Discrete Convolutions

- finite discrete:
  given two arrays $f \in \mathbb{R}^{N\times M}, g \in \mathbb{R}^{\tilde{N}\times\tilde{M}}$,
  define a third array with the dimensions:

$$h := (f * g) \in \mathbb{R}^{N\times M}$$

$$h_{n,m} := (f * g)_{n,m} = \sum_{n'=1}^{\tilde{N}} \sum_{m'=1}^{\tilde{M}} f(n+\delta n', m+\delta m') g(n', m')$$

$$= \sum_{n'=\alpha(\tilde{N},n)}^{\beta(\tilde{N},N)} \sum_{m'=\alpha(\tilde{M},m)}^{\beta(\tilde{M},M)} f(n+\delta n', m+\delta m') g(n', m')$$

  - $\delta n' := \delta(n', \tilde{N}) := n' - \lfloor \frac{\tilde{N}+1}{2} \rfloor$ **index centering**
  - $f(n,m) := 0$ for $n < 1$, $n \geq N$, $m < 1$ or $m \geq M$ (**zero padding**)
    - $\alpha(\tilde{N}, n) := 1 - \min(0, n - 1 + \delta(1, \tilde{N}))$, i.e., $n + \delta(\alpha(\tilde{N}, n), \tilde{N}) \geq 1$
      $\beta(\tilde{N}, N) := \ldots$, i.e., $n + \delta(\beta(\tilde{N}, n), \tilde{N}) \leq N$

Note: Here for two-dimensional arrays. The same works for any dimensional arrays.

Finite Discrete Convolutions / Shrinking Array Sizes

- finite discrete (alternative definition):
  given two arrays $f \in \mathbb{R}^{N \times M}, g \in \mathbb{R}^{\tilde{N} \times \tilde{M}}$,
  define a third array with the dimensions:

$$h := (f * g) \in \mathbb{R}^{(N - \tilde{N} + 1) \times (M - \tilde{M} + 1)}$$

$$h_{n,m} := (f * g)_{n,m} = \sum_{n'=1}^{\tilde{N}} \sum_{m'=1}^{\tilde{M}} f(n + n' - 1, m + m' - 1) g(n', m')$$

  - avoids zero padding
  - but leads to shrinking array sizes
  - rarely used in ML nowadays

1D convolution

- let $X \in \mathbb{R}^W$ be a sequence of length W (called **input**)
  (e.g., a time series),
  $K \in \mathbb{R}^{\tilde{W}}$ a **pattern** / **filter** / **kernel** / **window** ($\tilde{W} \ll W$):
  - $\tilde{W}$ **pattern size**

$$Z_w := (X * K)_w = \sum_{w'=1}^{\tilde{W}} X_{w+\delta w'} K_{w'}$$

  $Z \in \mathbb{R}^W$ called **feature map**
  - of same type as X
  - uses zero padding convention

1D convolution / Example

$$
\begin{aligned}
X &:= (1, -3, 4, 4, 2) \\
K &:= (-1, 1, 2) \\
X * K &=
\end{aligned}
$$

A. $(4, 15, 4)$

B. $(4, 15, 4, -2, -2)$

C. $(-5, 4, 15, 4, -2)$

1D convolution / Example

$$
\begin{aligned}
X &:= (1, -3, 4, 4, 2) \\
K &:= (-1, 1, 2) \\
X * K &=
\end{aligned}
$$

A.	$(4, 15, 4)$	with size shrinking
B.	$(4, 15, 4, -2, -2)$	without centering (unusual)
C.	$(-5, 4, 15, 4, -2)$	default

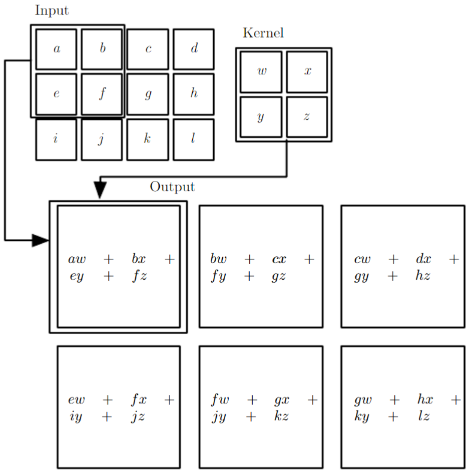
2D convolution

- let $X \in \mathbb{R}^{W \times H}$ be an array of dimensions $W \times H$ (e.g., an image), $K \in \mathbb{R}^{\tilde{W} \times \tilde{H}}$ a **pattern** / **filter** / **kernel** ($\tilde{W} \ll W, \tilde{H} \ll H$):

$$Z_{w,h} := (X * K)_{w,h} = \sum_{w'=1}^{\tilde{W}} \sum_{h'=1}^{\tilde{H}} X_{w+\delta w', h+\delta h'} K_{w',h'}$$

  $Z \in \mathbb{R}^{W \times H}$ called **feature map**
  - of same type as X

2D convolution / Example



Note: This example uses size shrinking. Usually we do not do that.

[source: Goodfellow et al. 2016]

3D convolution

- let $X \in \mathbb{R}^{W \times H \times D}$ be an array of dimensions $W \times H \times D$ (e.g., a 3d image),
  $K \in \mathbb{R}^{\tilde{W} \times \tilde{H} \times \tilde{D}}$ a **pattern** / **filter** / **kernel** $(\tilde{W} \ll W, \tilde{H} \ll H, \tilde{D} \ll D)$:

$$Z_{w,h,d} := (X * K)_{w,h,d} = \sum_{w'=1}^{\tilde{W}} \sum_{h'=1}^{\tilde{H}} \sum_{d'=1}^{\tilde{D}} X_{w+\delta w', h+\delta h', d+\delta d'} K_{w',h',d'}$$

  $Z \in \mathbb{R}^{W \times H \times D}$ called **feature map**
  - of same type as X

convolution for arrays of any order

- let $X \in \mathbb{R}^{M_1 \times M_2 \times \cdots \times M_D}$ be an array of order D,
  $K \in \mathbb{R}^{\tilde{M}_1 \times \tilde{M}_2 \times \cdots \times \tilde{M}_D}$ a **pattern** / **filter** / **kernel**
  $(\tilde{M}_d \ll M_d, \quad d = 1, \ldots, D)$:

$$Z_{m_1,m_2,\ldots,m_D} := (X * K)_{m_1,m_2,\ldots,m_D}$$
$$= \sum_{m'_1=1}^{\tilde{M}_1} \sum_{m'_2=1}^{\tilde{M}_2} \cdots \sum_{m'_D=1}^{\tilde{M}_D} X_{m_1+\delta m'_1, m_2+\delta m'_2, \ldots, m_D+\delta m'_D} K_{m'_1,m'_2,\ldots,m'_D}$$

$Z \in \mathbb{R}^{M_1 \times M_2 \times \cdots \times M_D}$ called **feature map**

- of same type as X

Outline

1. Convolutions

2. Ordered vs Unordered Dimensions

3. Convolutional Neural Networks

4. Convolutional Layers vs Fully Connected Layers

5. Reducing Resolutions: Pooling and Striding

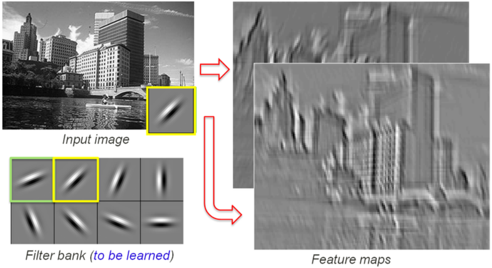
6. Outlook

Multiple Patterns

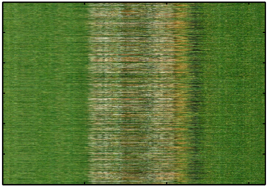
- let $X \in \mathbb{R}^{W \times H}$ be a $W \times H$ image,
  $K_1, \ldots, K_C \in \mathbb{R}^{\tilde{W} \times \tilde{H}}$ **multiple** patterns (**filter bank**):

$$Z_{w,h,c} := (X * K_c)_{w,h} = \sum_{w'=1}^{\tilde{W}} \sum_{h'=1}^{\tilde{H}} X_{w+\delta w', h+\delta h'} K_{c,w',h'}$$

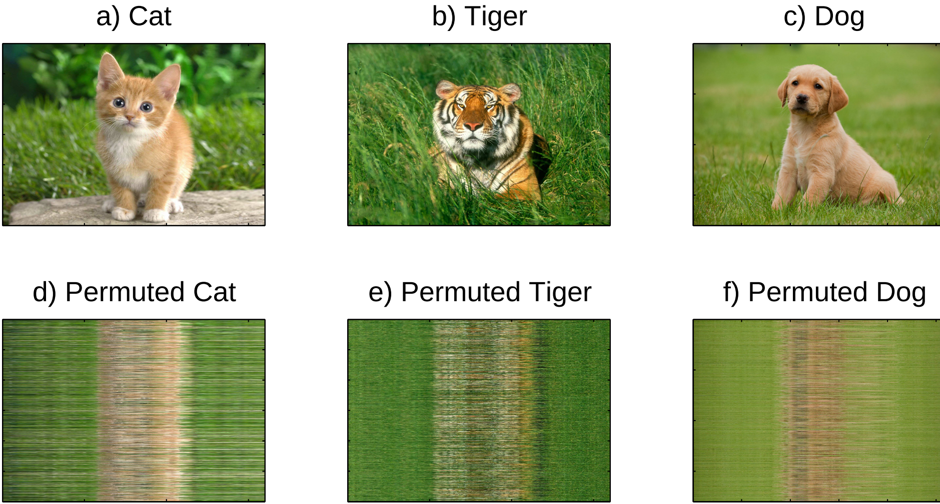
  $Z \in \mathbb{R}^{W \times H \times C}$ called **feature map array**
  - with dimensions $\dim(X) \times C$



What do you see?



What do you see?



Ordered vs Unordered Dimensions / Example

- let input $X \in \mathbb{R}^{W \times H \times C}$ have multiple variables measured for each position (w, h):

$$x_{w,h,1}, \quad x_{w,h,2}, \quad \ldots, \quad x_{w,h,C}$$

  - e.g., red/green/blue intensities of pixels in images: $C = 3$
  - each such variable often is called a **channel**
  - lets assume their order does not contain any information:
    - the indices of dimension C are unordered.
    - I will call dimension C **unordered**.

- **ordered dimensions**: first / width (W) and second / height (H).)
- **unordered dimensions**: third / color (C).

Ordered vs Unordered Dimensions

- **ordered dimensions**:
  - re-ordering the indices destroys information
  - e.g., positions, times, generally bins of a continuous variable
  - consider convolutions with patterns
    - pattern size usually way smaller than input size ($\tilde{W} \ll W$)
- **unordered dimensions**:
  - re-ordering the indices does not destroy any information
  - e.g., color channels, different attributes measured of an entity
  - convolutions with patterns over some indices make no sense
- but patterns can stretch over all indices of an unordered dimension and drop it in the output.

2D convolution with Channels

- let $X \in \mathbb{R}^{W \times H \times C}$ be an array with
  - ordered dimensions W and H and
  - unordered dimension C

  (e.g., an image with C channels),
  $K \in \mathbb{R}^{\tilde{W} \times \tilde{H} \times C}$ a **pattern** / **filter** / **kernel** ($\tilde{W} \ll W, \tilde{H} \ll H$):

$$Z_{w,h} := (X * K)_{w,h,c_0} = \sum_{w'=1}^{\tilde{W}} \sum_{h'=1}^{\tilde{H}} \sum_{c'=1}^{C} X_{w+\delta w', h+\delta h', c'} K_{w',h',c'}$$

  $Z \in \mathbb{R}^{W \times H}$ called **feature map**
  - with all dimensions of X but the unordered one.

- by abuse of notation, this is also often written as convolution $X * K$.
  - correct: use $c_0 := \lfloor \frac{C+1}{2} \rfloor$ to select just the center slice w.r.t. C

Outline

1. Convolutions

2. Ordered vs Unordered Dimensions

3. Convolutional Neural Networks

4. Convolutional Layers vs Fully Connected Layers

5. Reducing Resolutions: Pooling and Striding

6. Outlook

Nonlinear Activation of Feature Maps

- Q: why is stacking purely convolutional layers not useful?

$$Z^2 = Z^1 * W^2 = (X * W^1) * W^2$$

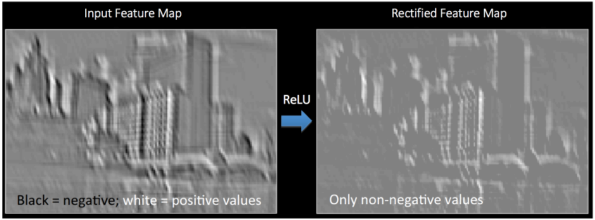
Nonlinear Activation of Feature Maps

- Q: why is stacking purely convolutional layers not useful?

$$Z^2 = Z^1 * W^2 = (X * W^1) * W^2$$

- use non-linear activation functions such as ReLU to avoid weight array collapsing:

$$Z^{\text{next}}_{w,h} := a((Z * W)_{w,h,c_0}) = a\left(\sum_{w'=1}^{\tilde{W}} \sum_{h'=1}^{\tilde{H}} \sum_{c'=1}^{C} Z_{w+\delta w', h+\delta h', c'} W_{w',h',c'}\right)$$



[source: Rob Fergus]

Fully Connected vs Convolutional Neural Networks

fully connected layers
(L hidden layers):

$$x \in \mathbb{R}^M, y \in \mathbb{R}^O$$

$$z^\ell := a_\ell(W^\ell z^{\ell-1} + b^\ell),$$

$$\in \mathbb{R}^{M_\ell}, \quad \ell = 1, \ldots, L+1$$

$$z^0 {:=} x, \quad M_0 {:=} M, \quad z^{L+1} {=:} \hat{y}, \quad M_{L+1} {:=} O$$

$$W^\ell \in \mathbb{R}^{M_\ell \times M_{\ell-1}}$$

$$b^\ell \in \mathbb{R}^{M_\ell}$$

$$a_\ell : \mathbb{R} \rightarrow \mathbb{R}$$

$$a_{L+1} : \mathbb{R}^{M^{\ell+1}} \rightarrow \mathbb{R}^{M^{\ell+1}} \text{ e.g., softmax}$$

Note: More precise: $W^\ell * z^{\ell-1}$ here denotes $((W^\ell_{m,.,.,.} * z^{\ell-1})_{m'_0})_{m=1:M^\ell}$. W is used twice!

Fully Connected vs Convolutional Neural Networks

fully connected layers
(L hidden layers):

$$x \in \mathbb{R}^M, y \in \mathbb{R}^O$$

$$z^\ell := a_\ell(W^\ell z^{\ell-1} + b^\ell),$$
$$\in \mathbb{R}^{M_\ell}, \quad \ell = 1, \dots, L+1$$

$$z^0 := x, \quad M_0 := M, \quad z^{L+1} =: \hat{y}, \quad M_{L+1} := O$$

$$W^\ell \in \mathbb{R}^{M_\ell \times M_{\ell-1}}$$

$$b^\ell \in \mathbb{R}^{M_\ell}$$

$$a_\ell : \mathbb{R} \to \mathbb{R}$$

$$a_{L+1} : \mathbb{R}^{M^{\ell+1}} \to \mathbb{R}^{M^{\ell+1}} \text{ e.g., softmax}$$

convolutional layers (2D, images):
(L hidden layers):

$$x \in \mathbb{R}^{W \times H \times C}, y \in \mathbb{R}^{W \times H \times O}$$

$$z^\ell := a_\ell(W^\ell * z^{\ell-1})$$
$$\in \mathbb{R}^{W \times H \times M_\ell}, \quad \ell = 1, \dots, L+1$$

$$z^0 := x, \quad M_0 := C, \quad z^{L+1} =: \hat{y}, \quad M_{L+1} := O$$

$$W^\ell \in \mathbb{R}^{M_\ell \times \tilde{W} \times \tilde{H} \times M_{\ell-1}}, \quad \tilde{W} \ll W, \tilde{H} \ll H$$

$$a_\ell : \mathbb{R} \to \mathbb{R}$$

Note: More precise: $W^\ell * z^{\ell-1}$ here denotes $((W^\ell_{m,.,.,.} * z^{\ell-1})_{m'_0})_{m=1:M^\ell}$. W is used twice!

Outline

1. Convolutions

2. Ordered vs Unordered Dimensions

3. Convolutional Neural Networks

4. Convolutional Layers vs Fully Connected Layers

5. Reducing Resolutions: Pooling and Striding

6. Outlook

A Convolutional Layer as Fully Connected Layer

- fully connected layer:
  - connected every layer input neuron $z_{w',h',m'}$
    with every layer output neuron $z_{w,h,m}$:

$$z^{\text{next}}_{w,h,m} := a(\sum_{w',h',m'} W_{w,h,m,w',h',m'} z_{w',h',m'})$$

  - # parameters: $W^2 H^2 M_\ell M_{\ell-1}$, # operations: $\mathcal{O}(W^2 H^2 M_\ell M_{\ell-1})$

A Convolutional Layer as Fully Connected Layer

- fully connected layer:
  - connected every layer input neuron $z_{w',h',m'}$ with every layer output neuron $z_{w,h,m}$:

$$z^{\text{next}}_{w,h,m} := a\Big(\sum_{w',h',m'} W_{w,h,m,w',h',m'} z_{w',h',m'}\Big)$$

  - # parameters: $W^2 H^2 M_\ell M_{\ell-1}$, # operations: $\mathcal{O}(W^2 H^2 M_\ell M_{\ell-1})$
- convolutional layer as fully connected layer:

$$W_{w,h,m,w',h',m'} := \begin{cases} W^{\text{conv}}_{m,w'-w,h'-h,m'}, & \text{if } w'-w < \tilde{W} \,\&\; h'-h < \tilde{H} \\ 0, & \text{else} \end{cases}$$

  - # parameters: # operations:

$$Z^{\text{next}}_{w,h} := a((Z * W)_{w,h,c_0}) = a\Big(\sum_{w'=1}^{\tilde{W}} \sum_{h'=1}^{\tilde{H}} \sum_{c'=1}^{C} Z_{w+\delta w',h+\delta h',c'} W_{w',h',c'}\Big)$$

Note: Here we use non-centered convolutions for ease of notation.

A Convolutional Layer as Fully Connected Layer

- fully connected layer:
  - connected every layer input neuron $z_{w',h',m'}$ with every layer output neuron $z_{w,h,m}$:

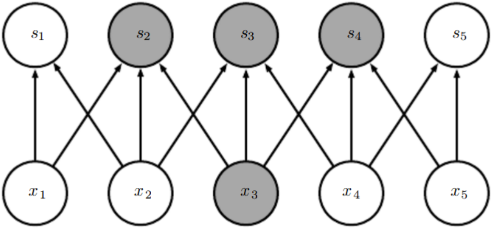
$$z^{\text{next}}_{w,h,m} := a(\sum_{w',h',m'} W_{w,h,m,w',h',m'} z_{w',h',m'})$$

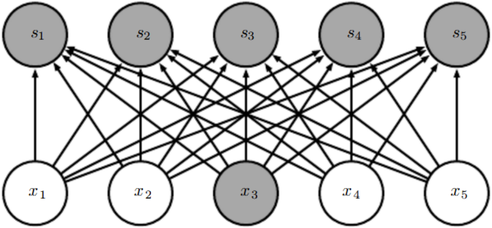
  - # parameters: $W^2 H^2 M_\ell M_{\ell-1}$, # operations: $\mathcal{O}(W^2 H^2 M_\ell M_{\ell-1})$
- convolutional layer as fully connected layer:

$$W_{w,h,m,w',h',m'} := \begin{cases} W^{\text{conv}}_{m,w'-w,h'-h,m'}, & \text{if } w'-w < \tilde{W} \& \ h'-h < \tilde{H} \\ 0, & \text{else} \end{cases}$$

  - # parameters: $\tilde{W}\tilde{H} M_\ell M_{\ell-1}$, # operations: $\mathcal{O}(WH\tilde{W}\tilde{H} M_\ell M_{\ell-1})$
  - convolutions have **sparse parameters**: most are 0.
    - **local interaction**
  - convolutions **share parameters** across positions: e.g., $W_{w,h,3,w+5,h+7,11} = W^{\text{conv}}_{3,5,7,11}$ are the same for all w, h
    - **translation invariant patterns**

Sparse Parameters, Local Interaction / Example

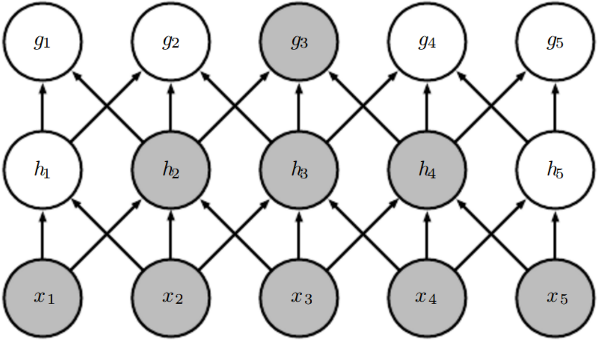




[source: Goodfellow et al., 2016]

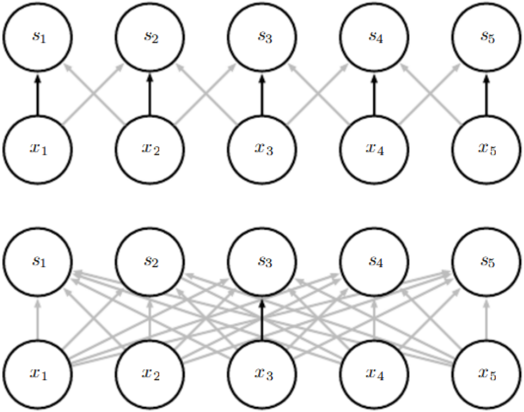
Local Interaction over Multiple Layers

- stacked convolutions increase the interaction area (**receptive field**)



[source: Goodfellow et al., 2016]

Shared Parameters / Example



[source: Goodfellow et al., 2016]

Outline

1. Convolutions

2. Ordered vs Unordered Dimensions

3. Convolutional Neural Networks

4. Convolutional Layers vs Fully Connected Layers

5. Reducing Resolutions: Pooling and Striding

6. Outlook

Reducing Resolutions

- convolutional layers retain the resolution of their inputs.
  - OK, if the output has the same resolution, e.g., for image segmenation tasks
- but what do we do if the output does not have any/some of the ordered input dimensions?
  - add a last fully connected layer
    - could lead to a large number of parameters for high resolutions
  - just average latent features over the ordered dimensions (**pooling**)
    - has no parameters
    - is it too simple?

Pooling

- reduce resolution by aggregating neighborhoods of a position:

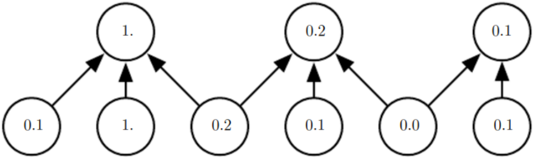
$$
\begin{aligned}
z^{\text{next}} &:= \text{poolmax}(z) \\
\text{poolmax}_{v,s} &: \mathbb{R}^{W\times H\times M} \rightarrow \mathbb{R}^{\lceil \frac{W}{s} \rceil \times \lceil \frac{H}{s} \rceil \times M} \\
z^{\text{next}}_{w',h',m} &:= \max(z_{w,h,m} \mid \begin{array}{l} w := w's, w's+1, \ldots, w's+v-1, \\ h := h's, h's+1, \ldots, h's+v-1) \end{array}
\end{aligned}
$$

  - **pool width** $v > 1$
  - **pool stride** s, $s \leq v$ (otherwise parts are skipped), often $s = v$

- **max pooling**: as above (using max)
- **average pooling**: use avg instead of max to aggregate neighborhoods

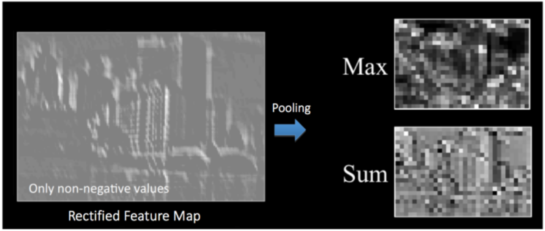
Pooling / Example 1D

- pool width $v = 3$, pool stride $s = 2$



[source: Goodfellow et al., 2016]

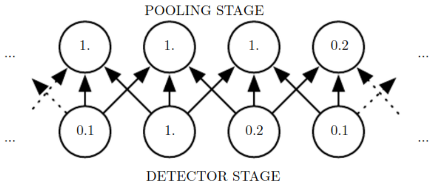
Pooling / Example 2D

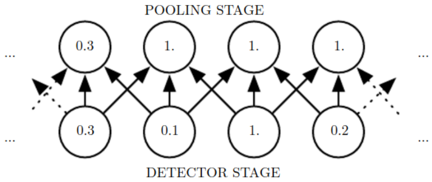


[source: Goodfellow et al., 2016]

Pooling / Smoothing

- pooling also can be used for smoothing the latent features e.g., for reduced sensitivity to small translations of the input:





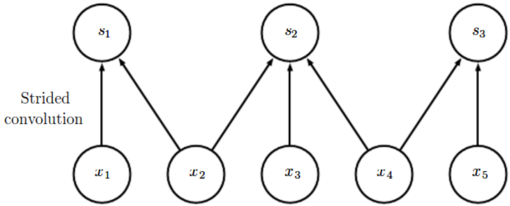
[source: Goodfellow et al., 2016]

Strided Convolutions

- instead of first computing high-resolution convolutions and
  and then aggregating with pooling,
  one also can use **strided convolutions**:

$$Z^{\text{next}}_{w,h,m} := (Z *_{\text{stride } s} W_m)_{w,h,m'_0}$$

$$= \sum_{w'=1}^{\tilde{W}} \sum_{h'=1}^{\tilde{H}} \sum_{m'=1}^{M'} Z_{ws+\delta w', hs+\delta h', m'} W_{m,w',h',m'}$$



[Goodfellow et al., 2016]

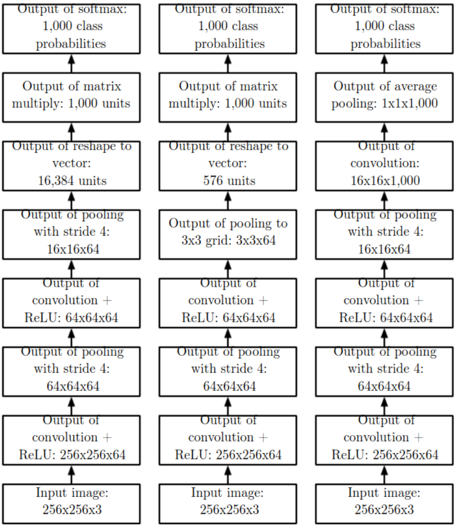
Reshaping and Fully Connected Layers

- finally add fully connected layers
- **reshape** the D-dimensional array $Z \in \mathbb{R}^{M_1 \times M_2 \times \cdots \times M_D}$ to a vector:

$$\text{reshape}(Z) := (Z_{\text{index(i)}})_{i=1,\ldots,M'} \in \mathbb{R}^{M'}, \quad M' := M_1 M_2 \cdots M_D$$

$$\text{index}(i)_d := (i - \sum_{d'=d+1}^{D} \text{index}(i)_{d'} M_1 M_2 \cdots M_{d'}) \text{ div } M_1 M_2 \cdots M_d$$

Example CNN Architectures



[source: Goodfellow et al., 2016]

Outline

1. Convolutions
2. Ordered vs Unordered Dimensions
3. Convolutional Neural Networks
4. Convolutional Layers vs Fully Connected Layers
5. Reducing Resolutions: Pooling and Striding
6. Outlook

Gradients and Backpropagation

- gradients for convolutions are easy to compute.
- backpropagation as learning algorithm works seamlessly.

Convolutional Neural Network Architectures

- **AlexNet**: deep CNNs.– 2012
  - Alex = First name of first author.
- **VGG**: networks using blocks – 2014
  - VGG = Visual Geometry Group.
- **NiN**: Network in Network – 2013
- **GoogleLeNet** – 2015: parallel concatenations; Inception
- **ResNet**: Residual Networks – 2016
- **DenseNet**: densely connected networks – 2016

Summary

- In multidimensional data, dimensions can be **ordered** or **unordered**.
  - information in ordered dimensions is destroyed if indices are shuffled.
    - images
    - time series
    - any indices representing binned continuous variables
- **Convolutions** allow to learn patterns in data with ordered dimensions.
- **Finite discrete convolutions** for arrays need to take care of **index centering** and **zero padding**.
- To reduce resolution, **pooling** and **striding** are used.
  - **max pooling** and **average pooling**.
- For unordered targets (e.g., classification), CNNs feature final fully connected layers (**reshaping** the last latent array to a vector).

Further Readings

- Goodfellow et al. 2016, ch. 9
- Zhang et al. 2020, ch. 6 & 7

Acknowledgement: An earlier version of the slides for this lecture have been written by my former postdoc Dr Josif Grabocka.

convolution for arrays of any order

- convolutions for arrays of any order can be written more compactly as follows:
- let $X \in \mathbb{R}^M, M \in \mathbb{N}^D$ be an array of order D,
  $K \in \mathbb{R}^{\tilde{M}}, \tilde{M} \in \mathbb{N}^D$ a **pattern** / **filter** / **kernel**
  ($\tilde{M} \ll M$ elementwise):

$$Z_m := (X * K)_m = \sum_{m' \in \rho(\tilde{M})} X_{m+\delta m'} K_{m'}, \quad m \in \rho(M)$$

  $Z \in \mathbb{R}^M$ called **feature map**
  - of same type as X
  - grid $\rho(\tilde{M}) := \bigtimes_{d=1}^{|\tilde{M}|} \{1, 2, \ldots, \tilde{M}_d\}$
  - index centering $\delta m' := \delta(m', \tilde{M}) := m' - (\lfloor \frac{\tilde{M}_d + 1}{2} \rfloor)_{d=1,\ldots,D}$

References

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The Mit Press, Cambridge, Massachusetts, November 2016. ISBN 978-0-262-03561-3.

Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander Smola. *Dive into Deep Learning*. https://d2l.ai/, 2020.