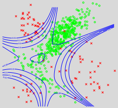
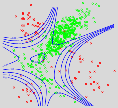


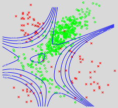
Machine Learning

Blockkurs *Neuronale Netze und Deep Learning* vom 7.6.2018

Mario Stanke
Institut für Mathematik und Informatik
Universität Greifswald

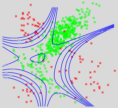


Problems with Fully Connected Artificial Neural Nets



Problems with Fully Connected Artificial Neural Nets

- high number of parameters
- when images are input:



Problems with Fully Connected Artificial Neural Nets

- high number of parameters
- when images are input:
  - no notion of pixel neighborhoods
  - no translation invariance

Convolution (1-dimensional)

Definition 1 (Convolution of vectors)

Let indices start at 0 and let $a = (a_0, a_1, \ldots, a_{m-1})$ and $b = (b_0, b_1, \ldots, b_{n-1})$ be vectors of dimensions m and $n \geq m$, respectively. The $(n + m - 1)$-dimensional vector $c := (c_0, \ldots, c_{n+m-2})$ with

$$c_i := \sum_{k \in \mathbb{Z}} a_k b_{i-k} \qquad (i = 0, \ldots, n + m - 2)$$

is called the convolution of a and b.

(For notational convenience set $a_i = 0$ if $i \neq \{0, \ldots, m-1\}$ and $b_i = 0$ if $i \neq \{0, \ldots, n-1\}$).

We write

$$c = a \otimes b.$$

Convolution (1-dimensional)

Definition 1 (Convolution of vectors)

Let indices start at 0 and let $a = (a_0, a_1, \ldots, a_{m-1})$ and $b = (b_0, b_1, \ldots, b_{n-1})$ be vectors of dimensions m and $n \geq m$, respectively. The $(n+m-1)$-dimensional vector $c := (c_0, \ldots, c_{n+m-2})$ with

$$c_i := \sum_{k \in \mathbb{Z}} a_k b_{i-k} \qquad (i = 0, \ldots, n+m-2)$$

is called the convolution of a and b.

(For notational convenience set $a_i = 0$ if $i \neq \{0, \ldots, m-1\}$ and $b_i = 0$ if $i \neq \{0, \ldots, n-1\}$).

We write

$$c = a \otimes b.$$

Example 2 ($n = m = 3$)

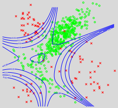
$$\begin{aligned} c_0 &= a_0 b_0 \\ c_1 &= a_0 b_1 + a_1 b_0 \\ c_2 &= a_0 b_2 + a_1 b_1 + a_2 b_0 \\ c_3 &= a_1 b_2 + a_2 b_1 \\ c_4 &= a_2 b_2 \end{aligned}$$

Example 3

$(1, 2, 3) \otimes (-2, 3, 4, 1)$
$= (-2, -1, 4, 18, 14, 3)$

FFT

The convolution c can be computed in time $O(n \log n)$ with the discrete fast Fourier transform (FFT).



Definition 4 (Cross-correlation (1-dimensional))

Call

$$d = (d_0, \dots, d_{n-m}) = a * b$$

with

$$d_j := \sum_{k=0}^{m-1} a_k b_{k+j} \qquad (j = 0, \dots, n-m)$$

the *cross-correlation* of a and b.

Cross-correlation and convolution

- Up to the (quick) operations *reversal*, *shift*, *cropping* and *zero-padding*, cross-correlation and convolution are equivalent.
- In particular,
$$d_j = (\mathrm{rev}(a) \otimes b)_{j+m-1} \,.$$
- Although 'convolution' is eponymous (namensgebend) for CNNs, cross-correlation is a more convenient definition.

Cross-correlation (2-dimensional)

Definition 5

Let $A = (a_{ij})_{0 \leq i,j < m}$ be a square $m \times m$-dimensional matrix and

$$B = (b_{ij})_{\substack{0 \leq i < h \\ 0 \leq j < w}}$$

be another matrix of shape $h \times w$.
The $h - n + 1 \times w - n + 1$-dimensional matrix C with entries

$$c_{i,j} := \sum_{i'=0}^{m-1} \sum_{j'=0}^{m-1} a_{i',j'} \cdot b_{i+i',j+j'}$$

is the 2-dimensional cross-correlation of A and B. We write $C = A * B$.

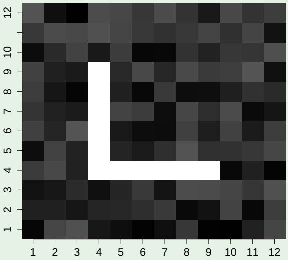
Example 6

$m - 2$, $h = 4$, $w = 5$.

$$A = \begin{pmatrix} 1 & -1 \\ 2 & -3 \end{pmatrix} \qquad B = \begin{pmatrix} 2 & -3 & 0 & 2 & -1 \\ 0 & 1 & 4 & 0 & 1 \\ 2 & -2 & 7 & 3 & 0 \\ -1 & 0 & 1 & 0 & 4 \end{pmatrix} \qquad C = \begin{pmatrix} 2 & -13 & 6 & 0 \\ 9 & -28 & 9 & 5 \\ 2 & -12 & 6 & -9 \end{pmatrix}$$

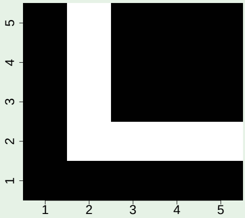
Cross-Correlation of an Image

$$\begin{pmatrix}
0.56 & 0.54 & 0.59 & 0.41 & 0.56 & 0.54 & 0.45 & 0.50 & 0.44 & 0.58 & 0.44 & 0.54 \\
0.48 & 0.42 & 0.53 & 0.48 & 0.53 & 0.46 & 0.42 & 0.52 & 0.60 & 0.52 & 0.56 & 0.52 \\
0.40 & 0.56 & 0.56 & 0.42 & 0.51 & 0.55 & 0.58 & 0.47 & 0.55 & 0.53 & 0.51 & 0.57 \\
0.40 & 0.44 & 0.58 & 1.00 & 0.51 & 0.47 & 0.51 & 0.43 & 0.54 & 0.48 & 0.56 & 0.46 \\
0.53 & 0.42 & 0.58 & 1.00 & 0.59 & 0.55 & 0.56 & 0.43 & 0.57 & 0.52 & 0.53 & 0.52 \\
0.44 & 0.53 & 0.45 & 1.00 & 0.51 & 0.43 & 0.43 & 0.53 & 0.49 & 0.42 & 0.52 & 0.57 \\
0.41 & 0.51 & 0.53 & 1.00 & 0.46 & 0.43 & 0.54 & 0.42 & 0.57 & 0.42 & 0.53 & 0.53 \\
0.43 & 0.49 & 0.49 & 1.00 & 0.48 & 0.46 & 0.56 & 0.48 & 0.50 & 0.54 & 0.56 & 0.57 \\
0.45 & 0.49 & 0.44 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 0.46 & 0.59 & 0.58 \\
0.59 & 0.45 & 0.50 & 0.48 & 0.48 & 0.60 & 0.46 & 0.41 & 0.46 & 0.55 & 0.57 & 0.40 \\
0.56 & 0.48 & 0.45 & 0.57 & 0.55 & 0.49 & 0.48 & 0.45 & 0.47 & 0.50 & 0.58 & 0.43 \\
0.41 & 0.48 & 0.44 & 0.54 & 0.43 & 0.55 & 0.52 & 0.54 & 0.55 & 0.43 & 0.53 & 0.59
\end{pmatrix}$$

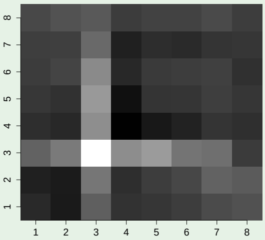


input image B

“filter” A



$$\begin{pmatrix}
-1 & 4 & -1 & -1 & -1 \\
-1 & 4 & -1 & -1 & -1 \\
-1 & 4 & -1 & -1 & -1 \\
-1 & 4 & 4 & 4 & 4 \\
-1 & -1 & -1 & -1 & -1
\end{pmatrix}$$



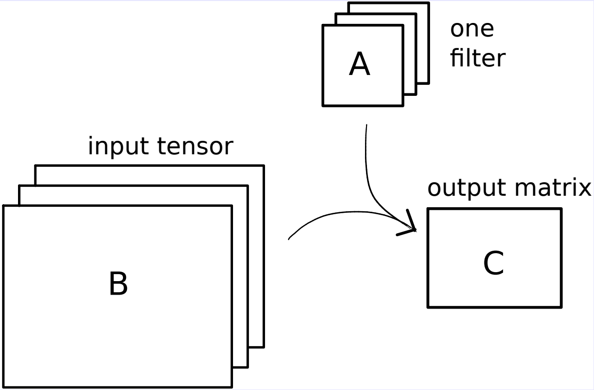
output image

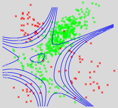
$C = A * B$

$$\begin{pmatrix}
6.6 & 7.4 & 5.0 & 4.19 & 4.7 & 4.3 & 4.7 & 5.2 \\
6.2 & 7.9 & 8.8 & 4.56 & 5.3 & 5.4 & 4.6 & 6.2 \\
5.2 & 5.9 & 9.2 & 3.27 & 4.6 & 5.2 & 3.8 & 5.6 \\
4.7 & 5.2 & 12.1 & 2.57 & 4.6 & 5.0 & 4.0 & 5.5 \\
4.4 & 4.1 & 11.0 & 0.85 & 2.0 & 3.7 & 3.1 & 5.1 \\
7.7 & 9.3 & 19.3 & 11.13 & 11.9 & 10.8 & 8.4 & 7.2 \\
2.7 & 2.7 & 9.6 & 3.69 & 4.4 & 5.7 & 5.4 & 6.7 \\
3.7 & 3.1 & 7.9 & 4.14 & 4.8 & 5.2 & 5.3 & 6.0
\end{pmatrix}$$

3-dimensional input

- Want to
  1. use multiple filters in parallel and
  2. stack several (convolutional) layers.
- Also, color images are naturally encoded as 3-dimensional (each pixel has a red, green and blue value).
- Solution: Define convolution for 3-dimensional tensor input as well.





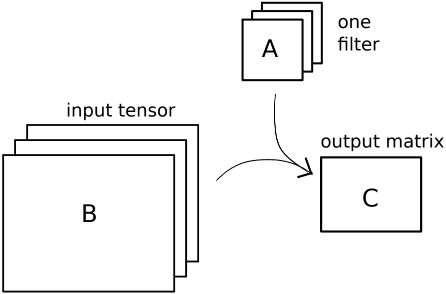
3-dimensional cross-correlation

Let $B = (b_{ijk})_{\substack{0 \le i < h \\ 0 \le j < w \\ 0 \le k < d}}$ be a tensor of shape $h \times w \times d$ and

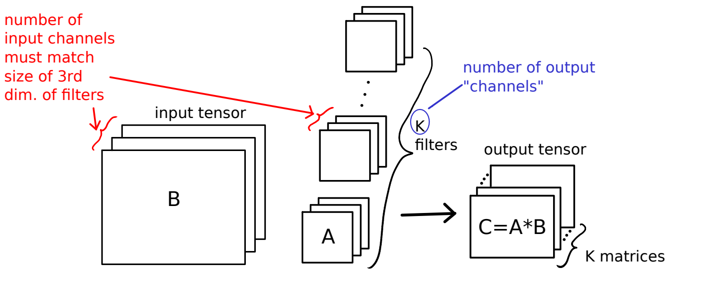
let $A = (a_{ijk})_{\substack{0 \le i,j < m \\ 0 \le k < d}}$ be another tensor (“filter”).

The cross-correlation of A and B is then the $h - n + 1 \times w - n + 1$-dimensional matrix $C = A * B$ with entries

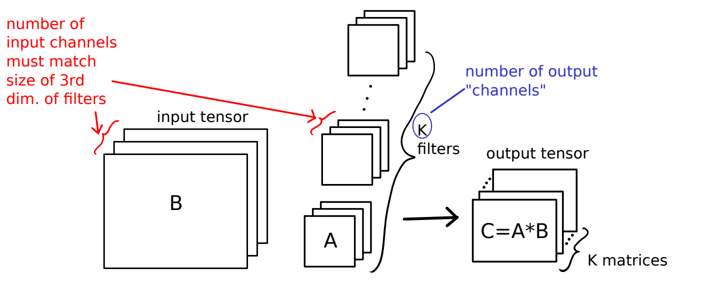
$$c_{i,j} := \sum_{i'=0}^{m-1} \sum_{j'=0}^{m-1} \sum_{k=0}^{d} a_{i',j',k} \cdot b_{i+i',j+j',k}.$$



Convolutional Layer

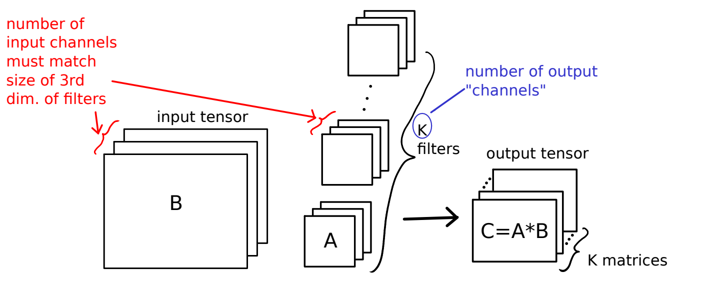


Convolutional Layer



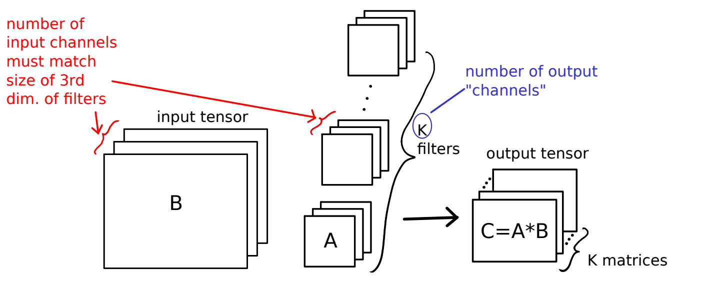
- The input width and height can be conserved in the ouput layer by zero-padding of input.

Convolutional Layer



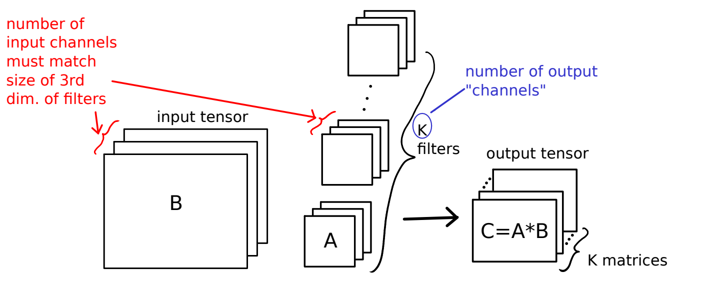
- The input width and height can be conserved in the ouput layer by zero-padding of input.
- Stride (Schrittweite) s: Skip $s - 1$ positions in each direction when 'sliding' A over $B \Rightarrow$ decreases output layer size up to a factor of s^2.

Convolutional Layer



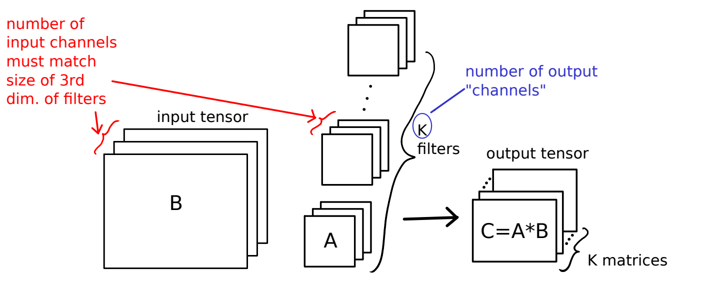
- The input width and height can be conserved in the ouput layer by zero-padding of input.
- Stride (Schrittweite) s: Skip $s - 1$ positions in each direction when 'sliding' A over $B \Rightarrow$ decreases output layer size up to a factor of s^2.
- The matrices A are learned, not set manually. The derivative wrt. to the filter matrix parameters needs to be computed for BackProp (omitted).

Convolutional Layer



- The input width and height can be conserved in the ouput layer by zero-padding of input.
- Stride (Schrittweite) s: Skip $s - 1$ positions in each direction when 'sliding' A over $B \Rightarrow$ decreases output layer size up to a factor of s^2.
- The matrices A are learned, not set manually. The derivative wrt. to the filter matrix parameters needs to be computed for BackProp (omitted).
- Convolution is a special case of a fully-connected layer, in which certain parameters are shared (parameter sharing).

Convolutional Layer



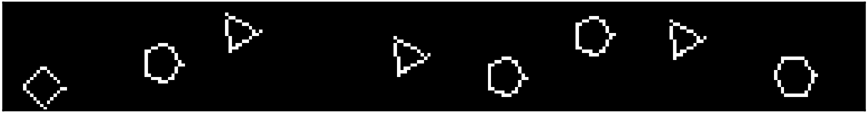
- The input width and height can be conserved in the ouput layer by zero-padding of input.
- Stride (Schrittweite) s: Skip $s - 1$ positions in each direction when 'sliding' A over $B \Rightarrow$ decreases output layer size up to a factor of s^2.
- The matrices A are learned, not set manually. The derivative wrt. to the filter matrix parameters needs to be computed for BackProp (omitted).
- Convolution is a special case of a fully-connected layer, in which certain parameters are shared (parameter sharing).
- Output neurons of convolution could represent lower-level features like ("lower left corner", "pupil") and be combined in deeper layers.

Example Application

Distinguish images of {3, 4, 5, 6}-gons

- inputs $\mathbf{x} \in \{0, \dots, 255\}^{32\times 32}$
  32×32 grayscale images (1 byte) which contain a triange, a quadrilateral, a pentagon or a hexagon with probability 1/4 each
- *n*-gons (*n*-Ecke) of the same type have the same size and shape, just the position in the image is random
- labels $y \in \{3, 4, 5, 6\}$

Examples:



4 5 3 3 5 5 3 6

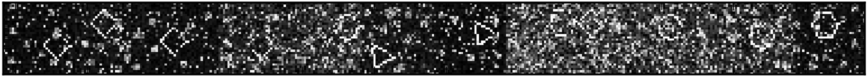
This application is tailored to work well with convolution layers.

n-gon Classification

Making it harder: noise

- randomly change pixels
- for some images more than for others (details in code)

Examples:

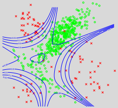


4 4 4 4 5 3 3 6 4 5 5 6

The program `createNGonExamples.py` to generate

- 50000 training examples and
- 2000 test examples

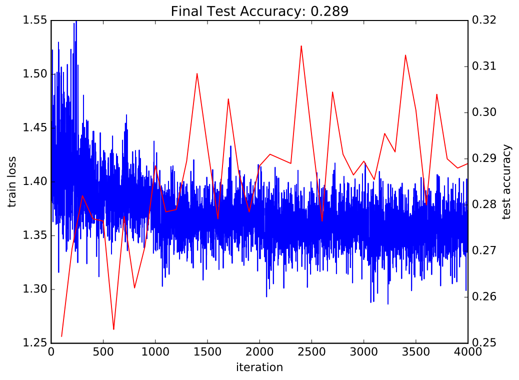
and to store in LMDBs (Lightning Memory-Mapped Databases) is on the class web site.



A Classical NN Performs Poorly

ANN architecture

- $L = 2$ layers:
  1 hidden layer of size 3200 and output layer of size 4
- ReLU activation function in hidden layer
- barely surpasses random-guessing-accuracy (0.25)
- (improvements by adding layers conceivable, but that is not the point, here)



A Convolutional Neural Net (CNN)

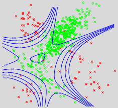
CNN architecture

layers:

1. convolutional with 8 filters of size 13 x 13
2. obtain a single maximum from each of the 8 'channels'
3. ReLU
4. 1 fully connected layer

Pycaffe code snippet

```
n = caffe.NetSpec()
n.data, n.label = L.Data([...]
          transform_param=dict(scale=1./255))
n.conv1 = L.Convolution(n.data, kernel_size=13,
                        num_output=8 [...])
n.pool1 = L.Pooling(n.conv1, kernel_size=32,
                    stride=32, pool=P.Pooling.MAX)
n.relu1 = L.ReLU(n.pool1, in_place=True)
n.score = L.InnerProduct(n.relu1, num_output=4 [....])
n.loss  = L.SoftmaxWithLoss(n.score, n.label)
```



ANN vs CNN

parameter numbers

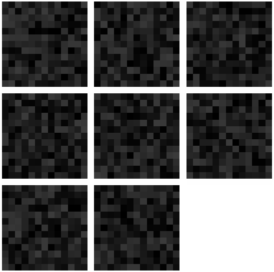
- ANN: $32 \cdot 32 \cdot 3200 + 3200 \cdot 4 = 3\,289\,600$
- CNN: $8 \cdot 13 \cdot 13 + 8 \cdot 4 = 1\,384$

(bias term parameters not included)

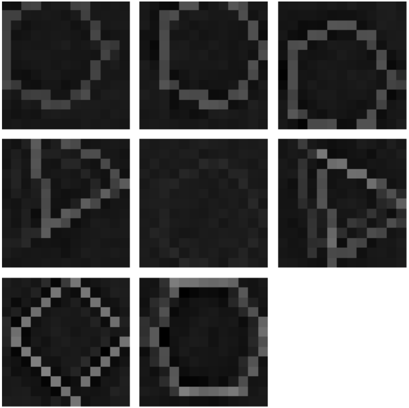
training times (CPU)

- ANN: 388 seconds
- CNN: 98 seconds

Filters



a the start (random init.)

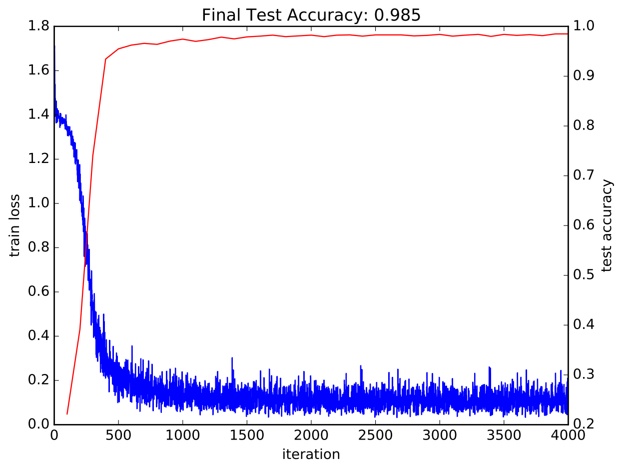


at end of training

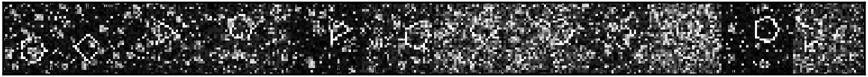
Pycaffe program

`train-angle-CNN.py` producing the images from this set of slides available on class web site.

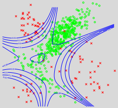
CNN performance on *n*-gon classification



Example predictions (CNN prediction / actual label)



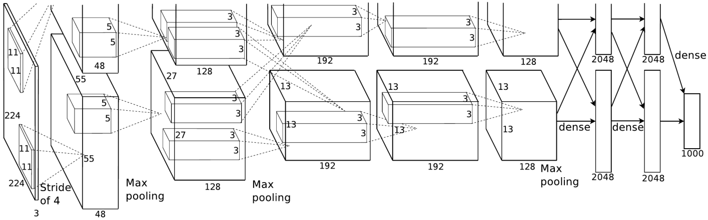
5 / 5   4 / 4   3 / 3   5 / 5   3 / 3   5 / 5   4 / 4   5 / 5   4 / 4   5 / 5   5 / 5   3 / 3



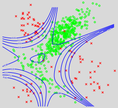
Multi-Layered CNN example

Demo: photo classification

- CNN from 2012 ("AlexNet")
- classification into 1000 categories



Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", *NIPS*, 2012



Convolutional NNs

Caveats

- filters are often less clearly interpretable than in above tailored example
- recognize only translation-invariant patterns, not from other transformations (e.g. scale, rotation)