TU München
Fakultät für Informatik
PD Dr. Rudolph Triebel
John Chiotellis, Maximilian Denninger

Machine Learning for Computer Vision
Winter term 2018

December 21, 2018
Topic: Neural Networks and Deep Learning

Exercise 1: Neural Networks

a) *Write the general formula of the transformation that a fully-connected layer performs on its inputs.*

The transformation is

$$x^{(\ell)} = s(W^{(\ell)} x^{(\ell-1)} + b^{(\ell)})$$

where $x^{(\ell-1)} \in \mathbb{R}^{d_{\ell-1}}$ is an input vector, $W^{(\ell)} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ is a weight matrix, $b^{(\ell)} \in \mathbb{R}^{d_\ell}$ is the bias vector, s is a non-linear function, such as ReLU or tanh applied elementwise to the vector input, resulting in an output vector $x^{(\ell)} \in \mathbb{R}^{d_\ell}$.

b) *Write the general formula of the transformation that three fully-connected layers perform on their inputs.*

$$x^{(3)} = s_3(W^{(3)} s_2(W^{(2)} s_1(W^{(1)} x^{(0)} + b^{(1)}) + b^{(2)}) + b^{(3)})$$

where $W^{(\ell)}, b^{(\ell)}$ are the weights and biases and s_ℓ is the non-linear activation function of the ℓ^{th} layer.

c) *What would be the disadvantage of not having a nonlinearity? What would the output of a three-layer perceptron be?*

Without nonlinearities we can only express affine transformations of the inputs:

$$\begin{aligned} x^{(3)} &= W^{(3)}(W^{(2)}(W^{(1)} x^{(0)} + b^{(1)}) + b^{(2)}) + b^{(3)} \\ &= W^{(3)} W^{(2)} W^{(1)} x^{(0)} + W^{(3)} W^{(2)} b^{(1)} + W^{(3)} b^{(2)} + b^{(3)} \\ &= \tilde{W} x^{(0)} + \tilde{b} \end{aligned}$$

The transformation can be directly rewritten as one layer, the advantages of *deep* learning disappear. We can as well solve a linear system instead of training a neural network.

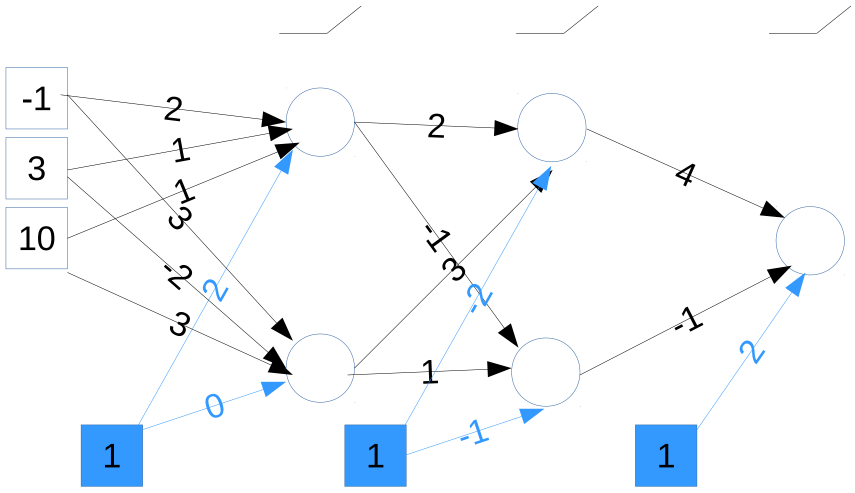

Figure 1: A 3-layer perceptron with ReLU activations.

d) *Draw a three-layer perceptron with arbitrary numbers as input values and weights, and choose arbitrary nonlinearities. Write the weights as matrices, and also write them next to the respective arrows in the drawing. Compute the outputs.*

$$W^{(1)} = \begin{pmatrix} 2 & 1 & 1 \\ 3 & -2 & 3 \end{pmatrix} \qquad b^{(1)} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

$$W^{(2)} = \begin{pmatrix} 2 & -1 \\ -1 & 1 \end{pmatrix} \qquad b^{(2)} = \begin{pmatrix} -2 \\ -1 \end{pmatrix}$$

$$W^{(3)} = \begin{pmatrix} 4 & -1 \end{pmatrix} \qquad b^{(3)} = 2$$

The input vector $x^{(0)}$ is first multiplied with the weight matrix of the first layer $W^{(1)}$ and added to the bias vector $b^{(1)}$. After applying the ReLU, we have the output of the first layer which is the input to the second layer, $x^{(1)}$.

$$x^{(1)} = ReLU(W^{(1)}x^{(0)} + b^{(1)}) \quad = ReLU(\begin{pmatrix} 11 \\ 21 \end{pmatrix} + \begin{pmatrix} 2 \\ 0 \end{pmatrix}) = ReLU(\begin{pmatrix} 13 \\ 21 \end{pmatrix}) = \begin{pmatrix} 13 \\ 21 \end{pmatrix}$$

$$x^{(2)} = ReLU(W^{(2)}x^{(1)} + b^{(2)}) \quad = ReLU(\begin{pmatrix} 5 \\ 8 \end{pmatrix} + \begin{pmatrix} -2 \\ -1 \end{pmatrix}) = ReLU(\begin{pmatrix} 3 \\ 7 \end{pmatrix}) = \begin{pmatrix} 3 \\ 7 \end{pmatrix}$$

$$x^{(3)} = ReLU(W^{(3)}x^{(2)} + b^{(3)}) = ReLU(5 + 2) = ReLU(7) = 7$$

Exercise 2: Deep Learning

a) *Write the general formula of the transformation that a 1D convolutional layer performs on its inputs.*

Assuming an input vector x and a filter w of length m, the convolution will output a vector h where

$$h_i = \sum_{j=1}^{m} w_j x_{j+i-\frac{m}{2}}$$

b) *Compute the result of a 1D convolutional layer with several input channels and several filters. Write arbitrary numbers as inputs and weights. Consider also the 2D case.*

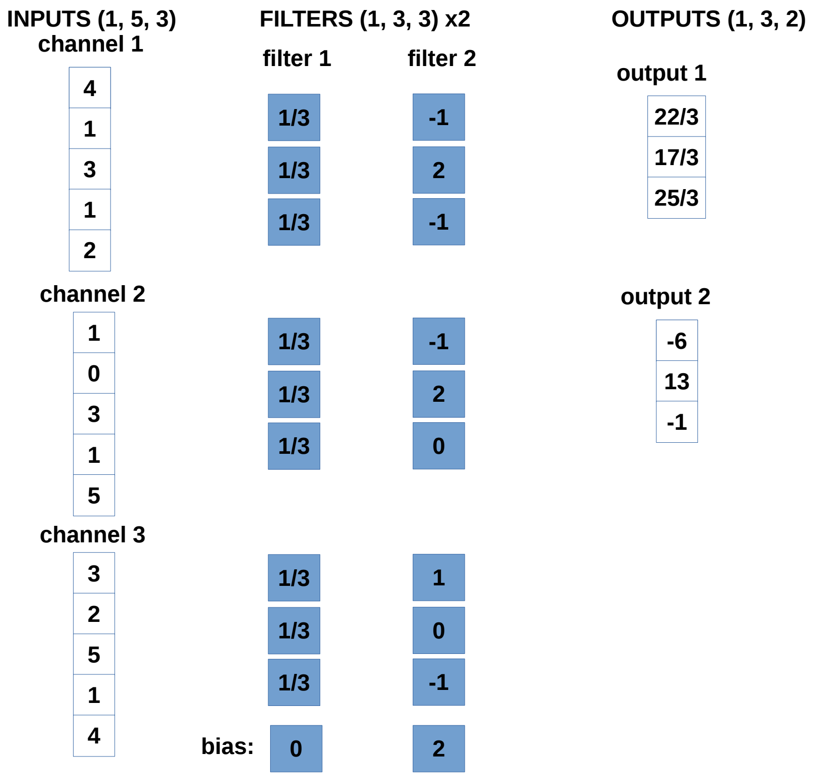

Figure 2: A convolutional layer. The numbers in parentheses indicate the volumes in the form (width, height, number of channels).

c) *Design an edge-detecting filter.*

Example: $\begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ reacts to vertical dark-to-bright intensity transitions (horizontal edges). After a ReLU, patches without this property would yield output 0.

What should its output be in regions with constant input image intensity?

With the value 0 usually representing the absence of a feature, the output should be 0 for regions without edges.

What property should an edge-detecting filter therefore have?

Its weights should sum up to 0.

Note that weights in neural networks are learned, rather than handcrafted. But if edge detection is useful for optimizing the loss function in question, something like edge detecting filters will emerge through training.

d) *Compute the receptive field of a three-layer 1D convolutional network with filter size 3 in the first layer, filter size 7 in the second layer, and filter size 7 in the third layer.*

$1 + (3 - 1) + (7 - 1) + (7 - 1) = 15$

If the input to this network has length 100 and 10 channels, what can be said about the length and number of channels of the output?

The length shrinks as $100 - 15 + 1 = 86$ due to the receptive window size (if the input is not padded with "fictional" (unknown/inexistent) values). The number of output channels can be arbitrary (equal to the chosen number of filters). What this choice should be depends on the application.

What happens with the last entry of the output, if the first entry of the input is increased by 1*? Why?*

The last entry of the output remains unchanged because the first entry of the input is beyond its receptive window. (Feature extraction in convolutional layers is *local*.)

e) *What two properties of feature extraction does a convolutional layer have?*

Locality (input values that are "far away" in the "spatial" dimensions do not affect a given local output) and translation-equivariance (shifting (translating) the input in the "spatial" dimensions yield the same output shifted in the same way). Use convolutional layers when this is what you want.

f) *Why do deep neural networks yield better results on certain datasets than shallow neural networks?*

They can learn more complicated transformations than shallow networks of comparable size. Each layer of a deep network learns a simple data transformation (while all layers together perform a complicated transformation), this resembles a "divide-and-conquer" strategy to learn the transformation. Besides, this yields simple meaningful feature extractors in every layer that are often useful for good generalization.