Technische Universität Hamburg
Institut für Mathematik
Lehrstuhl Numerische Mathematik

Jens-Peter M. Zemke
Jonas Grams

Mathematics of Neural Networks
winter semester 2021/2022
exercise sheet 4

Exercise 1: (2 points) Let $\mathbf{i} = (i_1, \dots, i_d)$ be a multi-index. Prove that multi-dimensional convolution

$$(\mathbf{X} * \mathbf{F})(n_1, \dots, n_d) = \sum_{\mathbf{i}} \mathbf{X}(n_1 - i_1, \dots, n_d - i_d) \cdot \mathbf{F}(i_1, \dots, i_d)$$

is commutative, i.e., $\mathbf{X} * \mathbf{F} = \mathbf{F} * \mathbf{X}$.

Exercise 2: (4 points) In Lecture 4 we showed that

$$\frac{\partial C}{\partial \mathbf{f}} = \widetilde{(\mathbf{a} * \widetilde{\boldsymbol{\delta}})} = \widetilde{\mathbf{a}} * \boldsymbol{\delta},$$

for the case of a 1D kernel $\mathbf{f}$ (see lecture notes, p. 93[1]), where the tilde represents a flipping of the entries. We now consider a 2D kernel $\mathbf{F} \in \mathbb{R}^{2\times 2}$ with input $\mathbf{A} \in \mathbb{R}^{3\times 3}$. The convolution $\mathbf{A} * \mathbf{F}$ is the given by

$$\begin{pmatrix} y_{1,1} & y_{1,2} \\ y_{2,1} & y_{2,2} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} * \begin{pmatrix} f_{1,1} & f_{1,2} \\ f_{2,1} & f_{2,2} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \star \begin{pmatrix} f_{2,2} & f_{2,1} \\ f_{1,2} & f_{1,1} \end{pmatrix},$$

or in vector notation by

$$\left(\begin{array}{c} y_{1,1} \\ y_{1,2} \\ \hline y_{2,1} \\ y_{2,2} \end{array}\right) = \left(\begin{array}{ccc|ccc|ccc} f_{2,2} & f_{2,1} & 0 & f_{1,2} & f_{1,1} & 0 & 0 & 0 & 0 \\ 0 & f_{2,2} & f_{2,1} & 0 & f_{1,2} & f_{1,1} & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & f_{2,2} & f_{2,1} & 0 & f_{1,2} & f_{1,1} & 0 \\ 0 & 0 & 0 & 0 & f_{2,2} & f_{2,1} & 0 & f_{1,2} & f_{1,1} \end{array}\right) \left(\begin{array}{c} a_{1,1} \\ a_{1,2} \\ a_{1,3} \\ \hline a_{2,1} \\ a_{2,2} \\ a_{2,3} \\ \hline a_{3,1} \\ a_{3,2} \\ a_{3,3} \end{array}\right).$$

a) How does the $\boldsymbol{\Delta}$ from the backpropagation look like in this case?
b) Prove the equality

$$\widetilde{(\mathbf{A} * \widetilde{\boldsymbol{\Delta}})} = \widetilde{\mathbf{A}} * \boldsymbol{\Delta}.$$

[1] Version from 26.10.2021

Exercise 3: (4 points) In preparation for convolutional networks change the current implementation of `DenseLayer` to treat *derivatives* in place of *gradients*:

- Initialize with transposed weights and biases given by `np.zeros(no)`.
- Evaluate using $\mathbf{z} = \mathbf{aW} + \mathbf{b}$, $a(\mathbf{z})$.
- Backpropagate using

$$\frac{\partial C}{\partial \mathbf{W}} = \mathbf{a}^{\mathsf{T}}\boldsymbol{\delta}, \quad \frac{\partial C}{\partial \mathbf{b}} = \boldsymbol{\delta},$$

  now by abuse of notation denoting the *derivatives*, and output $\boldsymbol{\delta}\mathbf{W}^{\mathsf{T}}$.
- Which other parts do you have to change accordingly?

Exercise 4: (4 points)

a) Add a class `DropoutLayer` to `layers.py`:
   (i) Initialize with the following attributes:
   - `p`: the probability `p` $\in [0.5, 0.8]$ with which neurons are kept, default: `p = .5` .
   - `mask`: a scaled version of a vector with 0's and 1's, specifiying the dropped neurons, set to `None` on initialization.
   - `trainable`: a Boolean value given on initialization (`True`: the network is trained, neurons are dropped, and the output is scaled; `False`: the network is not trained and the weights remain unchanged).

   (ii) Add the following methods:
   - `evaluate(self, a)`: If the network is trained, i.e., `trainable` is `True`, neurons are dropped and the input gets scaled. If the network is not trained, i.e., `trainable` is `False`, the input is returned.
     Hint: Implement the dropping of neurons by setting `self.mask` using
     `numpy.random.binomial(1, self.p, size=shape)`.
     The parameter `shape` is a tuple $(1, m)$ with m being the number of given input vectors. This ensures the same mask for the whole minibatch. The scaling factor is chosen as $1/(1 - p)$.
   - `backprop(self, delta)`: backpropagation by dropping neurons as given in `self.mask`.
   - `update`: perform the update after backpropagation. It shouldn't do anything since there are no parameters to update.

b) Test your code by expanding exercise 5c) from sheet 2. Use the logistic function as activation function. Compare the result for different probabilities `p` $\in [0.5, 0.8]$ to a network without dropout.

   Note that a higher learning rate is suggested for networks with dropout. Try different sizes for minibatches and different learning rates as well. Note that a higher learning rate is recommended for layers with dropout, use, e.g., `10*eta` as learning rate for layers w/ dropout, if `eta` is the learning rate for a layer w/o dropout.