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 7, solutions

Exercise 1: (2 points) Use the extended Euclidean algorithm to compute a GCD g of the two polynomials

$$a(u) := u^4 + 3u^3 - 8u^2 - 12u + 16, \quad b(u) := u^4 + u^3 - 7u^2 - u + 6,$$

and two polynomials c, d, such that

$$ac + bd = g$$

holds true.
Hint: You can implement the extended Euclidean algorithm (See Algorithm 3.1 on page 122 of the lecture notes[1]) with the Python package `sympy`. Use `sympy.quo(f, g)` to obtain the quotient of polynomials `f` and `g`. The computed output has rational coefficients and thus has to be scaled.
Use the following skeleton:

```
"""
quo computes for given f,g the quotient q with
f = q*g + r for some rest term r

degree returns the degree of a given polynomial

expand returns a polynomial in standard representation
expand(f) = \sum_{i = 0}^n f_i x^i
"""
from sympy     import quo, degree, expand
from sympy.abc import u

def extended_euclidean_algorithm(a, b):

  """
  TODO Implement the extended Euclidean algorithm
       with sympys div
  """
  pass

if __name__ == '__main__':

  a = u**4 + 3*u**3 - 8*u**2 - 12*u + 16
  b = u**4 + u**3 - 7*u**2 - u + 6

  r, c, d = extended_euclidean_algorithm(a, b)
```

[1]version from 01.11.2021

```
  print('GCD␣=', r)
  print('c␣=', c)
  print('d␣=', d)
  print('Error␣in␣Bezout␣=', expand(a*c + b*d - r))
```

Solution: We use SymPy and the code of the following listing to obtain a GCD g and polynomials c and d.

```
"""
quo computes for given f,g the quotient q with
f = q*g + r for some rest term r

degree returns the degree of a given polynomial

expand returns a polynomial in standard representation
expand(f) = \sum_{i = 0}^n f_i x^i
"""
from sympy     import quo, degree, expand
from sympy.abc import u

def extended_euclidean_algorithm(a, b):

  r_old, r = a, b
  c_old, c = 1, 0
  d_old, d = 0, 1

  while degree(r) > 0:

    q = quo(r_old, r)
    r_old, r = r, expand(r_old - q*r)
    c_old, c = c, expand(c_old - q*c)
    d_old, d = d, expand(d_old - q*d)

  return r_old, c_old, d_old

if __name__ == '__main__':

  a = u**4 + 3*u**3 - 8*u**2 - 12*u + 16
  b = u**4 + u**3 - 7*u**2 - u + 6

  r, c, d = extended_euclidean_algorithm(a, b)
  print('GCD␣=', r)
  print('c␣=', c)
  print('d␣=', d)
  print('Error␣in␣Bezout␣=', expand(a*c + b*d - r))
```

The solution is scaled by 4 here to obtain integer coefficients:

$$g(u) = -3u^2 + 9u - 6, \quad c(u) = -2u - 3, \quad d(u) = 2u + 7.$$

For these polynomials the relation

$$\begin{aligned}(ac + bd)(u) &= (u^4 + 3u^3 - 8u^2 - 12u + 16)(-2u - 3) + (u^4 + u^3 - 7u^2 - u + 6)(2u + 7)\\ &= -3u^2 + 9u - 6\\ &= g(u)\end{aligned}$$

is satisfied.

Exercise 2: (3+1 points)

a) Compute a minimal Winograd algorithm for the full convolution $\mathbf{z} = \mathbf{x} * \mathbf{f}$, where $\mathbf{x} \in \mathbb{R}^3$ and $\mathbf{f} \in \mathbb{R}^2$. Use the four points $g_1 = 0$, $g_2 = 1$, $g_3 = -1$, and $g_4 = \infty$.

b) Give the matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ of the matrix form

$$\mathbf{z} = \mathbf{A}^\mathsf{T}((\mathbf{B}\mathbf{x}) \circ (\mathbf{C}\mathbf{f})).$$

Hint: In the lecture you developed a minimal algorithm for $\mathbf{x} \in \mathbb{R}^2$ and $\mathbf{f} \in \mathbb{R}^2$ (see page 124 of the lecture notes[1]). Extend this example to $\mathbf{x} \in \mathbb{R}^3$ and $\mathbf{f} \in \mathbb{R}^2$.

Solution: The elements of $\mathbf{x} \in \mathbb{R}^3$ and $\mathbf{f} \in \mathbb{R}^2$ and $\mathbf{z} := \mathbf{x} * \mathbf{f} \in \mathbb{R}^4$ are denoted by

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}, \quad \mathbf{f} = \begin{pmatrix} f_0 \\ f_1 \end{pmatrix}, \quad \mathbf{z} = \begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix}.$$

a) We use the $4 = 3 + 2 - 1$ points $g_1 = 0$, $g_2 = 1$, $g_3 = -1$, and $g_4 = \infty$. We have

$$\prod_{j=1}^{3}(u - g_j) = (u - 0)(u - 1)(u - (-1)) = u^3 - u,$$

thus we need to compute

$$z(u) = x(u)f(u) \pmod{u^3 - u} + x_2 f_1(u^3 - u) = \widehat{z}(u) + x_2 f_1(u^3 - u), \qquad m_4 := x_2 f_1.$$

We compute

$$\begin{aligned} Z_1(u) &= x(u)f(u) \pmod{u} = x_0 f_0 =: m_1, \\ Z_2(u) &= x(u)f(u) \pmod{u - 1} = (x_0 + x_1 + x_2)(f_0 + f_1) =: m_2, \\ Z_3(u) &= x(u)f(u) \pmod{u + 1} = (x_0 - x_1 + x_2)(f_0 - f_1) =: m_3. \end{aligned}$$

The polynomial $\widehat{z}$ has a degree less than 3 and can be reconstructed using Lagrange's interpolation formula,

$$\widehat{z}(u) = \sum_{j=1}^{3} Z_j(u) \prod_{i \neq j} \frac{u - g_i}{g_j - g_i} = m_1(1 - u^2) + m_2 \frac{u^2 + u}{2} + m_3 \frac{u^2 - u}{2},$$

since

$$\frac{(u+1)(u-1)}{(0+1)(0-1)} = \frac{u^2 - 1}{-1}, \quad \frac{(u-0)(u+1)}{(1-0)(1+1)} = \frac{u^2 + u}{2}, \quad \frac{(u-0)(u-1)}{(-1-0)(-1-1)} = \frac{u^2 - u}{2}.$$

Thus we can compute the entries of $\mathbf{z}$ using four multiplications,

$$\begin{aligned} z(u) &= m_1(1 - u^2) + m_2 \frac{u^2 + u}{2} + m_3 \frac{u^2 - u}{2} + m_4(u^3 - u) \\ &= m_1 + \frac{m_2 - m_3 - 2m_4}{2} u + \frac{m_2 + m_3 - 2m_1}{2} u^2 + m_4 u^3. \end{aligned}$$

Explicitely computing the convolution as polynomial multiplication gives

$$\begin{aligned} z(u) = x(u)f(u) &= (x_0 + x_1 u + x_2 u^2)(f_0 + f_1 u) \\ &= x_0 f_0 + (x_0 f_1 + x_1 f_0)u + (x_1 f_1 + x_2 f_0)u^2 + x_2 f_1 u^3. \end{aligned}$$

It is obvious that $z_0 = m_1 = x_0 f_0$ and $z_3 = m_4 = x_2 f_1$. We focus on $m_2 \pm m_3$:

$$\begin{aligned} m_2 \pm m_3 &= (x_0 + x_1 + x_2)(f_0 + f_1) \pm (x_0 - x_1 + x_2)(f_0 - f_1) \\ &= \begin{cases} 2(x_0 + x_2)f_0 + 2x_1 f_1, & + \\ 2(x_0 + x_2)f_1 + 2x_1 f_0. & - \end{cases} \end{aligned}$$

Thus, also z_1 and z_2 are correctly computed,

$$\begin{aligned} z_1 &= \frac{m_2 - m_3 - 2m_4}{2} = (x_0 + x_2)f_1 + x_1 f_0 - x_2 f_1 = x_0 f_1 + x_1 f_0, \\ z_2 &= \frac{m_2 + m_3 - 2m_1}{2} = (x_0 + x_2)f_0 + x_1 f_1 - x_0 f_0 = x_2 f_0 + x_1 f_1. \end{aligned}$$

b) By inspection of the above formulae,

$$\mathbf{z} = \mathbf{A}^{\mathsf{T}}\mathbf{m} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/2 & -1/2 & -1 \\ -1 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{pmatrix} = \begin{pmatrix} m_1 \\ \frac{m_2 - m_3 - 2m_4}{2} \\ \frac{m_2 + m_3 - 2m_1}{2} \\ m_4 \end{pmatrix},$$

$$\mathbf{Bx} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}, \quad \mathbf{Cf} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \end{pmatrix}.$$

We could move the 1/2-terms in the second and third column of $\mathbf{A}^{\mathsf{T}}$, i.e., second and third row of $\mathbf{A}$, into the second and third row of $\mathbf{B}$ or $\mathbf{C}$ (here Winograd collects them, which makes sense, as this transforms the filters, which can be carried out once for all signals).

Exercise 3: (2+1+2* points)

a) Let $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{f} \in \mathbb{R}^k$, and $\mathbf{y} \in \mathbb{R}^{n+k-1}$ be given. Prove that

$$\sum_{i=0}^{n-1} x_i \sum_{j=0}^{k-1} y_{i+j} f_j = \sum_{p=0}^{n+k-2} y_p \sum_{\ell=0}^{p} x_\ell f_{p-\ell},$$

where undefined entries x_ℓ and $f_{p-\ell}$ are zero.

b) Show that the full convolution of $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{f} \in \mathbb{R}^k$ is related to the computation of the valid cross-correlation of $\mathbf{y} \in \mathbb{R}^{n+k-1}$ and $\mathbf{f} \in \mathbb{R}^k$ or FIR filter $F(n, k)$

$$\begin{pmatrix} y_0 & y_1 & \cdots & y_{k-1} \\ y_1 & y_2 & \cdots & y_k \\ \vdots & \vdots & & \vdots \\ y_{n-1} & y_n & \cdots & y_{n+k-2} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{k-1} \end{pmatrix}$$

by computing the scalar product of the resulting vector with $\mathbf{x} \in \mathbb{R}^n$ and using part a).

c) Use this result to derive a minimal algorithm for $F(3,2)$ from a minimal algorithm for the full convolution of $\mathbf{x} \in \mathbb{R}^3$ and $\mathbf{f} \in \mathbb{R}^2$.

Hint: A change of the summation order may be helpful for part c)

Solution: We solve parts a) and b) simultaneously. We write the inner sum on the left-hand side as matrix times vector and the outer sum as row vector times column vector to obtain

$$T(x,y,f) = \begin{pmatrix} x_0 & x_1 & \cdots & x_{n-1} \end{pmatrix} \begin{pmatrix} y_0 & y_1 & \cdots & y_{k-1} \\ y_1 & y_2 & \cdots & y_k \\ \vdots & \vdots & & \vdots \\ y_{n-1} & y_n & \cdots & y_{n+k-2} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{k-1} \end{pmatrix}.$$

We split the matrix into $n+k-1$ parts consisting of matrices $\mathbf{E}_p$, $p = 0, \dots, n-+k-2$, that are zero apart from the pth antidiagonal comprising ones that are multiplied by y_p,

$$\begin{pmatrix} y_0 & y_1 & \cdots & y_{k-1} \\ y_1 & y_2 & \cdots & y_k \\ \vdots & \vdots & & \vdots \\ y_{n-1} & y_n & \cdots & y_{n+k-2} \end{pmatrix} = y_0 \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} + \cdots + y_{n+k-2} \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}.$$

Antidiagonals are defined by indices that have a sum that is constant, i.e., the pth antidiagonal has indices $(\ell, p-\ell)$. Thus,

$$T(x,y,f) = \sum_{p=0}^{n+k-2} y_p \sum_{\ell=0}^{p} x_\ell f_{p-\ell}.$$

Part c) of the exercise is based on these considerations. We use the result of exercise 1) where we obtained a minimal algorithm for the convolution of $\mathbf{x} \in \mathbb{R}^3$ and $\mathbf{f} \in \mathbb{R}^2$, but define m_2 and m_3 scaled by 2, i.e.,

$$m_1 = x_0 f_0, \quad m_2 = (x_0 + x_1 + x_2)\frac{f_0 + f_1}{2}, \quad m_3 = (x_0 - x_1 + x_2)\frac{f_0 - f_1}{2}, \quad m_4 = x_2 f_1.$$

By exercise 1) the coefficients z_0, z_1, z_2, and z_3 are given by

$$z_0 = m_1, \quad z_1 = m_2 - m_3 - m_4, \quad z_2 = m_2 + m_3 - m_1, \quad z_3 = m_4.$$

Thus,

$$\begin{aligned} T(x,y,f) &= \sum_{p=0}^{n+k-2} y_p \sum_{\ell=0}^{p} x_\ell f_{p-\ell} = \sum_{p=0}^{n+k-2} y_p z_p \\ &= y_0 m_1 + y_1(m_2 - m_3 - m_4) + y_2(m_2 + m_3 - m_1) + y_3 m_4 \\ &= y_0 x_0 f_0 + y_1(x_0 + x_1 + x_2)\frac{f_0 + f_1}{2} - y_1(x_0 - x_1 + x_2)\frac{f_0 - f_1}{2} - y_1 x_2 f_1 \\ &\qquad + y_2(x_0 + x_1 + x_2)\frac{f_0 + f_1}{2} + y_2(x_0 - x_1 + x_2)\frac{f_0 - f_1}{2} - y_2 x_0 f_0 + y_3 x_2 f_1. \end{aligned}$$

Grouping differently gives

$$\begin{aligned} T(x,y,f) = (y_0 - y_2)x_0 f_0 + (y_1 + y_2)(x_0 + x_1 + x_2)\frac{f_0 + f_1}{2} \\ + (y_2 - y_1)(x_0 - x_1 + x_2)\frac{f_0 - f_1}{2} + (y_3 - y_1)x_2 f_1. \end{aligned} \tag{1}$$

Equating the terms in front of the coefficients x_0, x_1, and x_2 in equation (1) we obtain

$$\begin{pmatrix} y_0 & y_1 \\ y_1 & y_2 \\ y_2 & y_3 \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \end{pmatrix} = \begin{pmatrix} \mathsf{m}_1 + \mathsf{m}_2 + \mathsf{m}_3 \\ \mathsf{m}_2 - \mathsf{m}_3 \\ \mathsf{m}_2 + \mathsf{m}_3 + \mathsf{m}_4 \end{pmatrix},$$

where

$$\mathsf{m}_1 = (y_0 - y_2)f_0, \quad \mathsf{m}_2 = (y_1 + y_2)\frac{f_0 + f_1}{2}, \quad \mathsf{m}_3 = (y_2 - y_1)\frac{f_0 - f_1}{2}, \quad \mathsf{m}_4 = (y_3 - y_1)f_1,$$

which is a minimal algorithm for $F(3, 2)$. Just for the sake of completeness we derive a minimal algorithm for $F(2, 3)$: Equating the terms in front of the coefficients f_0 and f_1 in equation (1) we obtain

$$\begin{pmatrix} y_0 & y_1 & y_2 \\ y_1 & y_2 & y_3 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} M_1 + M_2 + M_3 \\ M_2 - M_3 + M_4 \end{pmatrix},$$

where

$$M_1 := (y_0 - y_2)x_0, \qquad M_2 := (y_1 + y_2)\frac{x_0 + x_1 + x_2}{2},$$
$$M_3 := (y_2 - y_1)\frac{x_0 - x_1 + x_2}{2}, \qquad M_4 := (y_3 - y_1)x_2,$$

which is a minimal algorithm for $F(2, 3)$.

Exercise 4: (3 points) Compare the computation time for the different implementations of the evaluation of a convolutional layer. In addition to the already implemented methods `evaluate_scipy()`, `evaluate_fft()` and `evaluate_im2col()` you can find an implementation of the Winograd approach for filters $F \in \mathbb{R}^{3,3}$ called `evaluate_winograd()` in `layers.py`. Compare them using a different number of images, corresponding height and width and channels for randomly generated input. For example you can try

- 100 and 20 for the number of inputs,
- 3 and 50 for the number of channels,
- 28 and 100 for the height/width of the input.

What can you oberserve?
You can use the following skeleton.

```python
from time   import process_time
from layers import *

"""
TODO Test the evaluations for different numbers of inputs,
     channels, and different input dimensions
"""
n, c, h, w = None

m, fh, fw = 32, 3, 3
tensor = (c, h, w)
fshape = (m, fh, fw)
conv = Conv2DLayer(tensor, fshape)

# generate random input
```

```
x = np.random.randint(0, 10, (n, c, h, w))

# test winograd
print('---------------winograd----------------')
start = process_time()
"""
TODO Add the evaluation of the convolutional layer with
     the wingrad minimal algorithm
"""
time_winograd = process_time() - start
print('Time␣used␣by␣winograd:', time_winograd, 'seconds')

# test im2col
print('---------------im2col----------------')
start = process_time()
"""
TODO Add the evaluation of the convolutional layer with
     the im2col method
"""
time_im2col = process_time() - start
print('Time␣used␣by␣im2col:', time_im2col, 'seconds')

# test
print('---------------fft-----------------')
start = process_time()
"""
TODO Add the evaluation of the convolutional layer with
     the the fft method
"""
time_fft = process_time() - start
print('Time␣used␣by␣fft:', time_fft, 'seconds')

# test
print('---------------scipy-----------------')
start = process_time()
"""
TODO Add the evaluation of the convolutional layer with
     the scipy convolutions
"""
time_scipy = process_time() - start
print('Time␣used␣by␣scipy:', time_scipy, 'seconds')
```

Solution: We implemented the code in the following listing.

```
from time import process_time
from layers import *

# set realistic parameters
#n, c, h, w = 100, 3, 28, 28 # MNIST, im2col wins
#n, c, h, w = 100, 3, 100, 100 # larger images, winograd wins
#n, c, h, w = 20, 50, 28, 28
n, c, h, w = 20, 50, 100, 100 # winograd wins, Scipy faster than fft
m, fh, fw = 32, 3, 3
tensor = (c, h, w)
fshape = (m, fh, fw)
conv = Conv2DLayer(tensor, fshape)
```

```

# generate random input
x = np.random.randint(0, 10, (n, c, h, w))

# test winograd
print('---------------winograd----------------')
start = process_time()
for i in range(1):
    print(i)
    y = conv.evaluate_winograd(x)
time_winograd = process_time() - start
print('Time␣used␣by␣winograd:', time_winograd, 'seconds')

# test im2col
print('---------------im2col----------------')
start = process_time()
for i in range(1):
    print(i)
    y = conv.evaluate_im2col(x)
time_im2col = process_time() - start
print('Time␣used␣by␣im2col:', time_im2col, 'seconds')

# test
print('---------------fft-----------------')
start = process_time()
for i in range(1):
    print(i)
    y = conv.evaluate_fft(x)
time_fft = process_time() - start
print('Time␣used␣by␣fft:', time_fft, 'seconds')

# test
print('---------------scipy-----------------')
start = process_time()
for i in range(1):
    print(i)
    y = conv.evaluate_scipy(x)
time_scipy = process_time() - start
print('Time␣used␣by␣scipy:', time_scipy, 'seconds')
```

For a situation that resembles the MNIST example the im2col approach is fastest. When using either more channels or larger images at one point the Winograd approach is faster. FFT and Scipy are both significantly slower, although the FFT approach is faster in most cases.