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 3

Exercise 1: (4 points)

a) Add different weight initialization schemes to `initializers.py` as treated in the lecture:

(i) A class `RandnAverage` that provides the method `wfun(self, m, n)` that returns an $m \times n$ random matrix with normally distributed entries and variance $2/(m+n)$.
Hint: Use `numpy.random.randn` and multiply the result by the square root of the desired variance.

(ii) A class `RandAverage` that provides the method `wfun(self, m, n)` that returns an $m \times n$ random matrix with uniformly distributed entries on the interval $[r, r]$, where the radius is given by $r := \sqrt{6/(m+n)}$.
Hint: Use `numpy.random.rand`. The method `rand` generates random numpy arrays with entries uniformly distributed in the interval $[0, 1]$. Shift and scale the results in an appropriate way.

(iii) A class `RandnOrthonormal` that provides the method `wfun(self, m, n)` that returns an $m \times n$ random orthonormal matrix.
Hint: Use `numpy.linalg.svd` or `numpy.linalg.qr` as sketched in the lecture to obtain a random orthonormal matrix from a random matrix.

b) On initialization of a `DenseLayer` add the attribute `initializer`, which is given as argument on initialization and should be `RandnAverage()` by default.
The weight matrix should be initialized by `wfun` from the initializer and scaled by a factor depending on the activation function. This factor can be obtained from the attribute `factor` of the activation function `afun`.

Exercise 2: (4 points)

a) Add as many other activation functions to `activations.py` as you like. e.g., those mentioned in the lecture (See lecture notes, section 1.3[1]). Use classes and subclasses to group activation functions. See the following listing for an example using the already implemented activation functions:

```
1  import numpy as np
2
3  # Heaviside family activation functions
4  class HeavisideLike:
5
6    def __init__(self):
7
8      self.factor = 1.0
9
10 """
```

[1] Version from October 20th, 2021

```
TODO Implement the following activation functions from the
     Heaviside activation function family:
      - Heaviside function
      - Modified Heaviside function
      - Logistic function
      - Exp function (as mentioned on the exercise sheet)
"""

# Sign family activation functions
class SignLike:

  def __init__(self):

    self.factor = 1.0

"""
TODO Implement the following activation functions from the
     Sign activation function family:
      - Sign function
      - TanH function
      - SoftSign function
"""
# ReLU family activation functions
class ReLULike:

  def __init__(self):

    self.factor = np.sqrt(2)

class ReLU(ReLULike):

  def __init__(self):

    super().__init__()
    self.data = None
    self.name = 'ReLU'

  def evaluate(self, x):

    self.data = x
    return x.clip(min = 0)

  def backprop(self, delta):

    return (self.data >= 0) * delta

"""
TODO Add the following activation functions from the
     ReLU activation function family:
      - leaky ReLU function
      - ELU function
      - SoftPlus function
      - Swish function
"""
# Abs family activation functions
class AbsLike:
```

```
67
68    def __init__(self):
69
70      self.factor = 1.0
71
72  class Abs(AbsLike):
73
74    def __init__(self, alpha=0.0):
75
76      super().__init__()
77      self.data  = None
78      self.name  = 'Abs'
79      self.alpha = alpha
80
81    def evaluate(self, x):
82
83      self.data = x
84      if(self.alpha == 0.0):
85        return np.abs(x)
86      else:
87        return np.sqrt(x*x + self.alpha**2) - self.alpha
88
89    def backprop(self, delta):
90
91      if(self.alpha == 0.0):
92        return np.sign(self.data)*delta
93      else:
94        return self.data/np.sqrt(self.data**2 + self.alpha**2) * delta
95
96  """
97  TODO Add the following activation functions from the
98       Abs activation function family:
99        - LOCo function
100       - Twist function
101       - SoftAbs function
102 """
```

Do you have ideas for activation functions different from the ones mentioned in lecture? You could search the internet or take a look at those gathered in Wikipedia.
Hint: The initialization factor for activation functions of one family can be set by calling `super().__init__()`

b) Expand example 5c) of exercise sheet 2 to use different activation functions. What do you observe? What happens when you use a function that is generally not used as an activation function like $\mathsf{Exp}(x) := e^x$?

Exercise 3: (4 points)

a) Add the optimizer Adam (Lecture notes, section 2.5.7[2]) as class `Adam` to the `optimizers.py` script. It should work similar to the `SGD` class.

b) Add momentum to the `SGD` class (See lecture notes, section 2.5.2).

Exercise 4: (4 points) You can add a penalty term for the size of weights by using L^2 regularization to change the cost function. The resulting effect is often called *weight decay*.

[2]Version from October 20th, 2021

a) Compare plain SGD using L^2 regularization with SGD using momentum by calculating the first three respective updates of a single weight matrix by hand.
b) What differences do you observe?
c) Without directly calculating it, what do you expect for other optimizers like Adam?

Exercise 5: (4 points)

a) Implement L^2 regularization in our code:
   (i) Add a class `L2Regularizer` to `layers.py`:
      - It is initialized with a regularization parameter `l2`.
      - It provides a method `update(self, p, dp)` which adds `p` scaled by the regularization parameter `l2` to the derivatives computed with backpropagation.
   (ii) Add an (optional) attribute `kernel_regularizer` to the `DenseLayer` class which is initialized as `None` by default.
   (iii) Add the regularization update as additional step in the backpropagation method `backprop` of `DenseLayer`.
b) Compare the effect of the regularization on plain SGD, SGD with momentum, and Adam by expanding example 5c) of exercise sheet 2. What do you observe?