sheet02-programming

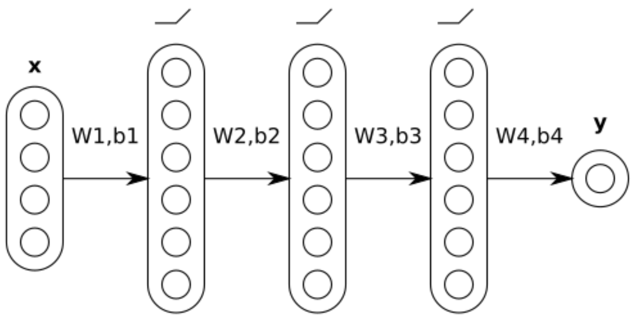
November 7, 2022

Exercises for the course Deep Learning 1 Winter Semester 2022/23

Machine Learning Group Faculty IV – Electrical Engineering and Computer Science Technische Universität Berlin

Exercise Sheet 1-2 (programming part)

In this homework, our goal is to test different approaches to implement neural networks. Here, we will be focusing on programming forward and backward computations. Training neural networks will be done in the next homework. The neural network we consider is depicted below:



0.1 Part 1: Implementing of Backpropagation (10 P)

The following code implements the forward pass of this network in numpy. Here, you are asked to implement the backward pass, and obtain the gradient with respect to the weight and bias parameters.

[1]:
```
import numpy,utils

# 1. Get the data and parameters

X,T = utils.getdata()
W,B = utils.getparams()
A = [X]

# 2. Run the forward pass
for i in range(3): A.append(numpy.maximum(0,A[-1].dot(W[i])+B[i]))
Y = A[-1].dot(W[3])+B[3]
```

```
# 3. Compute the error
err = ((Y-T)**2).mean()

# 4. Error backpropagation (TODO: replace by your code)
# * dE/dY
def err_derivative(Y, T):
    return 2*(Y - T)

# * A_i = max(0, Z_i) => dA_i/dZ_i = 1 if Z_i > 0 else 0
def reLu_derivative(A):
    return (A > 0)*1
def back_propagation(W,B,A,Y,T):
    dY = err_derivative(Y,T)
    deltas = [dY] # dErr/dY
    DW = []
    DB = []
    for i in reversed(range(len(W))):
        if i == len(W)-1 :
            dW_i = deltas[-1].dot(A[i]) # dW_4
            dB_i = deltas[-1] # dB_4
            DW.insert(0,dW_i)
            DB.insert(0,dB_i)
            dA_i = W[i].dot(deltas[-1])
            deltas.insert(0, dA_i)
        else:
            derivative_activation = reLu_derivative(A[i+1])
            dW_i = A[i].T.dot(derivative_activation * deltas[0].T)
            DW.insert(0,dW_i)
            dB_i = derivative_activation * deltas[0].T
            DB.insert(0, dB_i)
            dA_i = W[i].dot((derivative_activation * deltas[0].T).T)
            deltas.insert(0, dA_i)

    return DW
DW = back_propagation(W,B,A,Y,T)
# 5. Show error gradient w.r.t. the 1st weight parameter
print(numpy.linalg.norm(DW[0][0,0]))
```

```
1.5422821523392451
```

0.2 Part 2: Using Automatic Differentiation (10 P)

Because gradient computation can be error-prone, we often rely on libraries that incorporate automatic differentiation. In this exercise, we make use of the PyTorch library. You are then asked to compute the error of the neural network within that framework, which will then be automatically differentiated.

[2]:
```python
import torch
import torch.nn as nn

# 1. Get the data and parameters

X,T = utils.getdata()
W,B = utils.getparams()

# 2. Convert to PyTorch objects

X = torch.Tensor(X)
T = torch.Tensor(T)
W = [nn.Parameter(torch.Tensor(w)) for w in W]
B = [nn.Parameter(torch.Tensor(b)) for b in B]

# 3. Compute the forward pass and the error (TODO: replace by your code)
input_from_prev = X
relu = nn.ReLU()

for i in range(3):
    output = relu(input_from_prev @ W[i] + B[i])
    input_from_prev = output

err = ((input_from_prev @ W[3] + B[3] - T)**2).mean()

# 4. Apply automatic differentiation

err.backward()

# 5. Show error gradient w.r.t. the 1st weight parameter

print(numpy.linalg.norm(W[0].grad[0,0]))
```

```
1.5422821
```

```
c:\users\87290\appdata\local\programs\python\python37\lib\site-
packages\tqdm\auto.py:22: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

0.3 Part 3: Object-Oriented Implementation (10 P)

As a last exercise, we would like to make use of existing neural network objects of the PyTorch library. Here, most of the code is already implemented for you. You are only asked to find where the error gradient of the first weight parameter has been stored, and to print it.

[4]:
```
import torch
import torch.nn as nn
import numpy as np


# 1. Get the data and parameters

X,T = utils.getdata()
W,B = utils.getparams()

# 2. Convert to PyTorch objects

X = torch.Tensor(X)
T = torch.Tensor(T)
W = [torch.nn.Parameter(torch.Tensor(w.T)) for w in W]
B = [torch.nn.Parameter(torch.Tensor(b)) for b in B]

# 3. Build the neural network

net = torch.nn.Sequential(
          nn.Linear(4,6),nn.ReLU(),
          nn.Linear(6,6),nn.ReLU(),
          nn.Linear(6,6),nn.ReLU(),
          nn.Linear(6,1))

for l,w,b in zip(list(net)[::2],W,B):
    l.weight = w
    l.bias = b

# 4. Compute the forward pass and the error gradient

Y = net.forward(X)
err = ((Y-T)**2).mean()
err.backward()

# 5. Show error gradient w.r.t. the 1st weight parameter (TODO: replace by your␣
 ↪code)
print(np.linalg.norm(net[0].weight.grad[0,0]))
```

```
1.5422821
```