section3

September 17, 2023

[146]:
```python
import numpy as np
import torch
import torch.functional as F
import array_to_latex as a2l

'''Solving Q3 using the chain rule and PyTorch, good for sigmoid and ReLU'''

def my_relu(vec):
    return np.array([max(0, v) for v in vec.flatten()]).reshape(-1, 1)

def a2l_short(a):  # prints out nice latex
    return a2l.to_ltx(a, frmt='{:.2f}', arraytype='bmatrix', print_out=False)

def my_relu_derivative(vec):
    ret = np.zeros((len(vec), len(vec)))  # square matrix
    relu_vec = my_relu(vec)
    # if ReLu(vec_i) > 0 then ret_{i,i} = 1, ret_{i,i} = 0 otherwise
    relu_vec[relu_vec > 0] = 1
    relu_vec[relu_vec <= 0] = 0
    np.fill_diagonal(a=ret, val=relu_vec)
    return ret

def my_sigmoid_derivative(vec):  # computes diag(vec)diag(1-vec)
    ret1 = np.zeros((len(vec), len(vec)))
    ret2 = np.zeros((len(vec), len(vec)))
    np.fill_diagonal(a=ret1, val=vec)
    np.fill_diagonal(a=ret2, val=1-vec)
    return np.matmul(ret1, ret2)

def my_run(relu_or_sigmoid, wrt, verbose=False, rand=False):
    if not rand:
        x1 = np.array([0., 1., 2.]).reshape(-1, 1)
        W1 = np.array([[ 3., -1,  1],
                       [-5.,  2, -1]])
        b1 = np.array([1., 2]).reshape(-1, 1)

        W2 = np.array([[ 1., -2],
                    [-3.,  1]])
        b2 = np.array([1., 1]).reshape(-1, 1)
        t = np.array([1., 2]).reshape(-1, 1)
    else:
        shape1 = np.random.randint(2, 12)
        shape2 = np.random.randint(2, 12)
        x1 = np.random.rand(1, shape1).reshape(-1, 1) * np.random.
↪randint(low=-5, high=5)
        W1 = np.random.rand(shape2, shape1) * np.random.randint(low=-5, high=5)
        b1 = np.random.rand(1, shape2).reshape(-1, 1) * np.random.
↪randint(low=-5, high=5)

        shape3 = np.random.randint(2, 12)
        W2 = np.random.rand(shape3, shape2) * np.random.randint(low=-5, high=5)
        b2 = np.random.rand(1, shape3).reshape(-1, 1) * np.random.
↪randint(low=-5, high=5)
        t = np.random.rand(1, shape3).reshape(-1, 1) * np.random.
↪randint(low=-5, high=5)
    if verbose:
        print(x1.shape, W1.shape, b1.shape, W2.shape, b2.shape, t.shape)
    if relu_or_sigmoid == 'relu':
        # print('here', (np.matmul(W1, x1) + b1).shape)
        x2 = my_relu(np.matmul(W1, x1) + b1)
    elif relu_or_sigmoid == 'sigmoid':
        x2 = 1/(1 + np.exp(-(np.matmul(W1, x1) + b1)))
    else:
        raise KeyError('relu or sigmoid')
    x3 = np.matmul(W2, x2) + b2

    mse_here = 0.5 * np.linalg.norm(t - x3) ** 2
    print('mse=' + str(mse_here))

    if relu_or_sigmoid == 'relu':
        step_f_x2_wrt_z = my_relu_derivative(x2)  # = diag(v)
        # dReLU(z)/dz as matrix
    if relu_or_sigmoid == 'sigmoid':
        step_f_x2_wrt_z = my_sigmoid_derivative(x2)
        # dSigma(z)/dz as matrix
    if wrt == 'W1':  # see equation 23
        dL_dW1 = np.matmul(np.matmul(-(t - x3).T, W2),
                           np.matmul(step_f_x2_wrt_z, np.kron(x1.T, np.
↪identity(W1.shape[0]))))
        print('dL/d' + wrt + ', res before re-shaping\n', a2l_short(dL_dW1))
        print('res after re-shaping\n', a2l_short(dL_dW1.reshape(W1.shape[1],␣
↪W1.shape[0])))
    if wrt == 'W2':  # see equation 24
```

```
        print('dL/d' + wrt + ', res before re-shaping\n',
              a2l_short(np.matmul(-(t - x3).T, np.kron(x2.T, np.identity(W2.
↪shape[0])))))
        print('res after re-shaping\n',
              a2l_short(np.matmul(-(t - x3).T,
                        np.kron(x2.T, np.identity(W2.shape[0]))).reshape(W2.
↪shape[1], W2.shape[0])))
    if wrt == 'b1':  # see equation 25
        print('dL/d' + wrt, a2l_short(np.matmul(np.matmul(-(t - x3).T, W2),␣
↪step_f_x2_wrt_z)))
    if wrt == 'b2':  # see equation 26
        print('dL/d' + wrt, a2l_short(-(t - x3).T))

    if verbose:  # prints out intermediate steps
        print('dL/dx3 = -(t - x3)^T=\n', a2l_short(-(t - x3).T))
        print('dx3/dW2=x2^T kron I_2x2=\n', a2l_short(np.kron(x2.T, W2.
↪shape[0])))
        print('d(z)/dW1=d(W1x1+b1)/dW1=x1^T kron I_2x2=\n', a2l_short(np.
↪kron(x1.T, W1.shape[0])))
        print('dx3/dx2\n', a2l_short(W2))
        print('dReLU(z)/dz=dReLU(W1x1+b1)/dW1=\n', a2l_short(step_f_x2_wrt_z))
        print('z=W1x1+b1\n', a2l_short(np.matmul(W1, x1) + b1))
        print('x2=ReLU(z)=\n', a2l_short(x2))
        print('x3=\n', a2l_short(x3))
    return x1, W1, b1, W2, b2, t


def compute_grad_pytorch(relu_or_sigmoid, wrt,
                         x1=None, W1=None, b1=None, W2=None, b2=None, t=None,
                         rand=False):
    if not rand:  # uses whatever is given in Q3.2
        x1_ = np.array([0., 1., 2.]).reshape(-1, 1)
        b1_ = np.array([1., 2]).reshape(-1, 1)
        W1_ = np.array([[ 3., -1,  1],
                        [-5.,  2, -1]])

        b2_ = np.array([1., 1]).reshape(-1, 1)
        W2_ = np.array([[ 1., -2],
                        [-3.,  1]])
        t_ = np.array([1., 2]).reshape(-1, 1)
    else:  # these vars come from my_run
        x1_ = x1
        W1_ = W1
        b1_ = b1

        W2_ = W2
        b2_ = b2
        t_ = t

    x1n = torch.tensor(x1_, requires_grad=True)
    b1n = torch.tensor(b1_, requires_grad=True)
    b2n = torch.tensor(b2_, requires_grad=True)
    W1n = torch.tensor(W1_, requires_grad=True)
    W2n = torch.tensor(W2_, requires_grad=True)

    x1_after = torch.matmul(W1n, x1n) + b1n

    if relu_or_sigmoid == 'relu':
        x2n = torch.relu(x1_after)
    else:
        x2n = torch.sigmoid(x1_after)
    x3n = torch.matmul(W2n, x2n) + b2n
    tn = torch.tensor(t_, requires_grad=True)
    mse = 0.5 * F.norm(tn - x3n)**2
    print(mse)
    which_param = {'W1': W1n, 'W2': W2n, 'b1': b1n, 'b2': b2n}
    mse.backward(inputs=which_param[wrt])
    print(relu_or_sigmoid + ' dL/d' + wrt + '\n', which_param[wrt].grad.T, '\n')
    return


my_run(relu_or_sigmoid='relu', wrt='W1')
compute_grad_pytorch(relu_or_sigmoid='relu', wrt='W1')

my_run(relu_or_sigmoid='relu', wrt='W2')
compute_grad_pytorch(relu_or_sigmoid='relu', wrt='W2')

my_run(relu_or_sigmoid='relu', wrt='b1')
compute_grad_pytorch(relu_or_sigmoid='relu', wrt='b1')

my_run(relu_or_sigmoid='relu', wrt='b2', verbose=True)
compute_grad_pytorch(relu_or_sigmoid='relu', wrt='b2')
```

```
mse=14.499999999999998
dL/dW1, res before re-shaping
 \begin{bmatrix}
  0.00 &  0.00 &  13.00 & -1.00 &  26.00 & -2.00
\end{bmatrix}
res after re-shaping
 \begin{bmatrix}
  0.00 &  0.00\\
  13.00 & -1.00\\
  26.00 & -2.00
\end{bmatrix}
```

```
tensor(14.5000, dtype=torch.float64, grad_fn=<MulBackward0>)
relu dL/dW1
 tensor([[ 0.0000,  0.0000],
        [13.0000, -1.0000],
        [26.0000, -2.0000]], dtype=torch.float64)

mse=14.499999999999998
dL/dW2, res before re-shaping
 \begin{bmatrix}
 -4.00 & -10.00 & -4.00 & -10.00
\end{bmatrix}
res after re-shaping
 \begin{bmatrix}
 -4.00 & -10.00\\
 -4.00 & -10.00
\end{bmatrix}
tensor(14.5000, dtype=torch.float64, grad_fn=<MulBackward0>)
relu dL/dW2
 tensor([[ -4.0000, -10.0000],
        [ -4.0000, -10.0000]], dtype=torch.float64)

mse=14.499999999999998
dL/db1 \begin{bmatrix}
  13.00 & -1.00
\end{bmatrix}
tensor(14.5000, dtype=torch.float64, grad_fn=<MulBackward0>)
relu dL/db1
 tensor([[13.0000, -1.0000]], dtype=torch.float64)

(3, 1) (2, 3) (2, 1) (2, 2) (2, 1) (2, 1)
mse=14.499999999999998
dL/db2 \begin{bmatrix}
 -2.00 & -5.00
\end{bmatrix}
dL/dx3 = -(t - x3)^T=
 \begin{bmatrix}
 -2.00 & -5.00
\end{bmatrix}
dx3/dW2=x2^T kron I_2x2=
 \begin{bmatrix}
  4.00 &  4.00
\end{bmatrix}
d(z)/dW1=d(W1x1+b1)/dW1=x1^T kron I_2x2=
 \begin{bmatrix}
  0.00 &  2.00 &  4.00
\end{bmatrix}
dx3/dx2
 \begin{bmatrix}
```

```
  1.00 & -2.00\\
 -3.00 &  1.00
\end{bmatrix}
dReLU(z)/dz=dReLU(W1x1+b1)/dW1=
 \begin{bmatrix}
  1.00 &  0.00\\
  0.00 &  1.00
\end{bmatrix}
z=W1x1+b1
 \begin{bmatrix}
  2.00\\
  2.00
\end{bmatrix}
x2=ReLU(z)=
 \begin{bmatrix}
  2.00\\
  2.00
\end{bmatrix}
x3=
 \begin{bmatrix}
 -1.00\\
 -3.00
\end{bmatrix}
tensor(14.5000, dtype=torch.float64, grad_fn=<MulBackward0>)
relu dL/db2
 tensor([[-2.0000, -5.0000]], dtype=torch.float64)
```

[147]:
```
my_run(relu_or_sigmoid='sigmoid', wrt='W1')
compute_grad_pytorch(relu_or_sigmoid='sigmoid', wrt='W1')

my_run(relu_or_sigmoid='sigmoid', wrt='W2')
compute_grad_pytorch(relu_or_sigmoid='sigmoid', wrt='W2')

my_run(relu_or_sigmoid='sigmoid', wrt='b1')
compute_grad_pytorch(relu_or_sigmoid='sigmoid', wrt='b1')

my_run(relu_or_sigmoid='sigmoid', wrt='b2')
compute_grad_pytorch(relu_or_sigmoid='sigmoid', wrt='b2')
```

```
mse=4.201102887391705
dL/dW1, res before re-shaping
 \begin{bmatrix}
  0.00 &  0.00 &  0.78 & -0.10 &  1.55 & -0.21
\end{bmatrix}
res after re-shaping
 \begin{bmatrix}
  0.00 &  0.00\\
  0.78 & -0.10\\
  1.55 & -0.21
\end{bmatrix}
tensor(4.2011, dtype=torch.float64, grad_fn=<MulBackward0>)
sigmoid dL/dW1
 tensor([[ 0.0000,  0.0000],
        [ 0.7774, -0.1050],
        [ 1.5547, -0.2100]], dtype=torch.float64)

mse=4.201102887391705
dL/dW2, res before re-shaping
 \begin{bmatrix}
 -0.78 & -2.43 & -0.78 & -2.43
\end{bmatrix}
res after re-shaping
 \begin{bmatrix}
 -0.78 & -2.43\\
 -0.78 & -2.43
\end{bmatrix}
tensor(4.2011, dtype=torch.float64, grad_fn=<MulBackward0>)
sigmoid dL/dW2
 tensor([[-0.7758, -2.4324],
        [-0.7758, -2.4324]], dtype=torch.float64)

mse=4.201102887391705
dL/db1 \begin{bmatrix}
  0.78 & -0.10
\end{bmatrix}
tensor(4.2011, dtype=torch.float64, grad_fn=<MulBackward0>)
sigmoid dL/db1
 tensor([[ 0.7774, -0.1050]], dtype=torch.float64)

mse=4.201102887391705
dL/db2 \begin{bmatrix}
 -0.88 & -2.76
\end{bmatrix}
tensor(4.2011, dtype=torch.float64, grad_fn=<MulBackward0>)
sigmoid dL/db2
 tensor([[-0.8808, -2.7616]], dtype=torch.float64)
```

[148]:
```
# testing with random inputs
x1, W1, b1, W2, b2, t = my_run(relu_or_sigmoid='relu', wrt='W1', rand=True)
compute_grad_pytorch(relu_or_sigmoid='relu', wrt='W1', rand=True,
                     x1=x1, W1=W1, b1=b1, W2=W2, b2=b2, t=t)

x1, W1, b1, W2, b2, t = my_run(relu_or_sigmoid='relu', wrt='W2', rand=True)
```

```
compute_grad_pytorch(relu_or_sigmoid='relu', wrt='W2', rand=True,
                     x1=x1, W1=W1, b1=b1, W2=W2, b2=b2, t=t)

x1, W1, b1, W2, b2, t = my_run(relu_or_sigmoid='relu', wrt='b1', rand=True)
compute_grad_pytorch(relu_or_sigmoid='relu', wrt='b1', rand=True,
                     x1=x1, W1=W1, b1=b1, W2=W2, b2=b2, t=t)
```

```
mse=4284.1927808565515
dL/dW1, res before re-shaping
 \begin{bmatrix}
 -59.28 & -75.72 & -209.31 & -290.40 & -370.93 & -1025.27 & -358.42 & -457.81 &
-1265.43 & -80.43 & -102.73 & -283.96 & -430.61 & -550.02 & -1520.29 & -321.76 &
-410.99 & -1136.00
\end{bmatrix}
res after re-shaping
 \begin{bmatrix}
 -59.28 & -75.72 & -209.31\\
 -290.40 & -370.93 & -1025.27\\
 -358.42 & -457.81 & -1265.43\\
 -80.43 & -102.73 & -283.96\\
 -430.61 & -550.02 & -1520.29\\
 -321.76 & -410.99 & -1136.00
\end{bmatrix}
tensor(4284.1928, dtype=torch.float64, grad_fn=<MulBackward0>)
relu dL/dW1
 tensor([[  -59.2843,   -75.7240,  -209.3059],
        [ -290.3996,  -370.9283, -1025.2691],
        [ -358.4225,  -457.8142, -1265.4272],
        [  -80.4290,  -102.7323,  -283.9585],
        [ -430.6100,  -550.0196, -1520.2886],
        [ -321.7636,  -410.9897, -1136.0013]], dtype=torch.float64)

mse=32.47118572498797
dL/dW2, res before re-shaping
 \begin{bmatrix}
  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &
0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &
0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &
0.00 &  0.00
\end{bmatrix}
res after re-shaping
 \begin{bmatrix}
  0.00 &  0.00 &  0.00 &  0.00\\
  0.00 &  0.00 &  0.00 &  0.00\\
  0.00 &  0.00 &  0.00 &  0.00\\
  0.00 &  0.00 &  0.00 &  0.00\\
  0.00 &  0.00 &  0.00 &  0.00\\
  0.00 &  0.00 &  0.00 &  0.00\\
  0.00 &  0.00 &  0.00 &  0.00\\
  0.00 &  0.00 &  0.00 &  0.00
\end{bmatrix}
tensor(32.4712, dtype=torch.float64, grad_fn=<MulBackward0>)
relu dL/dW2
 tensor([[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]], dtype=torch.float64)

mse=23.41035200715275
dL/db1 \begin{bmatrix}
  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00 &  0.00
\end{bmatrix}
tensor(23.4104, dtype=torch.float64, grad_fn=<MulBackward0>)
relu dL/db1
 tensor([[0., 0., 0., 0., 0., 0., 0.]], dtype=torch.float64)
```

[148]:

```
```