COMPSCI 371 Homework 8

Group Members: Mayur Sekhar, Jai Kasera, Rithvik Neti

Problem 0 (3 points)

Part 1: MLP Back-Propagation: Mathematics

Problem 1.1 (Exam Style)

Mathematical expressions:

$l_q(y, z) = \frac{1}{2} \sum_{i=0}^{n} (z_i - y_i)^2$

$\frac{\partial l_q}{\partial z_i} = \frac{\partial}{\partial z_i} (\frac{1}{2} \sum_{i=0}^{n} (z_i - y_i)^2) = 2 * \frac{1}{2} (z_i - y_i) = z_i - y_i$

Thus, $\frac{\partial l_q}{\partial z_i} = z_i - y_i$

$l_h(y, z) = log(\sum_{j=0}^{n-1} e^{z_j}) - z_y$

$\frac{\partial l_h}{\partial z_i} = \frac{e^{z_j}}{log(\sum_{j=0}^{n-1} e^{z_j})} - \delta_{i=y}$

Numerical values:

When $y = (-2, 1)$ and $z = (4, 1)$

$l_q(y, z) = \frac{1}{2} * 36 = 18$

$g_q(y, z) = (6, 0)$

When $y = 1$ and $z = (-1, 0, 3)$

$l_h(y, z) = log(e^{-1} + e^0 + e^3) - 0 = log(e^{-1} + 1 + e^3) = 3.066$

$g_{h,0}(y, z) = \frac{e^{-1}}{(e^{-1} + e^0 + e^3)} = 0.01715$

$g_{h,1}(y, z) = \frac{e^{0}}{(e^{-1} + e^0 + e^3)} - 1 = -0.9534$

$g_{h,2}(y, z) = \frac{e^{3}}{(e^{-1} + e^0 + e^3)} = 0.9362$

$g_h(y, z) = (0.01715, -0.9534, 0.9362)$

In [13]:
```
import math
l_h = math.log(1/math.e + 1 + math.e**3)

g_h0 = ((math.e**-1)/((1/math.e + 1 + math.e**3)))
g_h1 = ((math.e**0)/((1/math.e + 1 + math.e**3))) - 1
g_h2 = ((math.e**3)/((1/math.e + 1 + math.e**3)))
print(f'l_h = {l_h:.4g}, g_h0 = {g_h0:.4g}, g_h1 = {g_h1:.4g}, g_h2 = {g_h2:.4g}')
```

```
l_h = 3.066, g_h0 = 0.01715, g_h1 = -0.9534, g_h2 = 0.9362
```

Problem 1.2 (Exam Style)

$\frac{\partial y_i}{\partial x_j} = 1$ if $i == j$ and $x_i > 0$, 0 otherwise

When $x = (-2, 1, 3)$,

$p(x) = (0, 1, 3)$

$$J_x = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Problem 1.3 (Exam Style)

$\frac{\partial y_i}{\partial x_j} = v_{ij}$

$\frac{\partial y_i}{\partial v_{jk}} = \delta_{i=j} x_k$

$\frac{\partial y_i}{\partial b_j} = \delta_{i=j}$

$$y = Vx + b = \begin{pmatrix} 3 & 0 \\ -1 & 4 \\ 2 & 5 \end{pmatrix} \begin{pmatrix} -2 \\ 1 \end{pmatrix} + \begin{pmatrix} 3 \\ -2 \\ 1 \end{pmatrix} = \begin{pmatrix} -6+3 \\ 6-2 \\ 1+1 \end{pmatrix} = \begin{pmatrix} -3 \\ 4 \\ 2 \end{pmatrix}$$

$$J_x = \begin{pmatrix} 3 & 0 \\ -1 & 4 \\ 2 & 5 \end{pmatrix}$$

$$J_v = \left[\begin{pmatrix} -2 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ -2 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ -2 & 1 \end{pmatrix} \right]$$

$$J_b = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Problem 1.4 (Exam Style)

$$\begin{aligned}
\mathbf{g}_{V_2} &= \mathbf{g}_{\mathbf{z}} \cdot J_{V_2} \\
\mathbf{g}_{\mathbf{b}_2} &= \mathbf{g}_{\mathbf{z}} \cdot J_{b_2} \\
\mathbf{g}_{\mathbf{q}} &= \mathbf{g}_{\mathbf{z}} \cdot J_q \\
\mathbf{g}_{\mathbf{p}} &= \mathbf{g}_{\mathbf{q}} \cdot J_p \\
\mathbf{g}_{V_1} &= \mathbf{g}_{\mathbf{p}} \cdot J_{V_1} \\
\mathbf{g}_{\mathbf{b}_1} &= \mathbf{g}_{\mathbf{p}} \cdot J_{b_1} \\
\mathbf{g}_{\mathbf{x}} &= \mathbf{g}_{\mathbf{p}} \cdot J_x
\end{aligned}$$

Problem 1.5 (Exam Style)

Forward pass:

$$\begin{aligned}
\mathbf{x} &= (-1, 2) \\
\mathbf{p} &= (-2, 5, 3) \\
\mathbf{q} &= (0, 5, 3) \\
\mathbf{z} &= (0, 3) \\
\lambda &= \frac{1}{2}((0+4)^2 + (3-1)^2) = \frac{1}{2}(16+4) = 10
\end{aligned}$$

Local Jacobians:

$$\begin{aligned}
J_{V_2} &= \left[\begin{pmatrix} 0 & 5 & 3 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 5 & 3 \end{pmatrix} \right] \\
J_{\mathbf{b}_2} &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\
J_{\mathbf{q}} &= \begin{pmatrix} 2 & 1 & -3 \\ 3 & -1 & 2 \end{pmatrix} \\
J_{\mathbf{p}} &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
J_{V_1} &= \left[\begin{pmatrix} -1 & 2 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ -1 & 2 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ -1 & 2 \end{pmatrix} \right] \\
J_{\mathbf{b}_1} &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}
\end{aligned}$$

Gradients:

$$
\begin{aligned}
\mathbf{g_z} &= z - y = (4, 2) \\
\mathbf{g}_{V_2} &= g_z \cdot J_{V_2} = \begin{pmatrix} 0 & 20 & 12 \\ 0 & 10 & 6 \end{pmatrix} \\
\mathbf{g}_{\mathbf{b}_2} &= g_z \cdot J_{b_2} = (4, 2) \\
\mathbf{g_q} &= g_z \cdot J_q = (14, 2, -8) \\
\mathbf{g_p} &= (0, 2, -8) \\
\mathbf{g}_{V_1} &= \begin{pmatrix} 0 & 0 \\ -2 & 4 \\ 8 & -16 \end{pmatrix} \\
\mathbf{g}_{\mathbf{b}_1} &= (0, 2, -8)
\end{aligned}
$$

Part 2: MLP Back-Propagation: Code

In [110…

```
import numpy as np
from types import SimpleNamespace

tests = [
    SimpleNamespace(
        v1=np.array(((2, -1), (0, 3), (2, 1)), dtype=float),
        b1=np.array((2, -1, 3), dtype=float),
        v2=np.array(((2, 1, -3), (3, -1, 2)), dtype=float),
        b2=np.array((4, 2), dtype=float),
        x=np.array((-1, 2), dtype=float),
        y=np.array((-4, 1), dtype=float),
        loss='quadratic'
    ),
    SimpleNamespace(
        v1=np.array(((2, -1, 0), (0, 1, 3)), dtype=float),
        b1=np.array((2, -1), dtype=float),
        v2=np.array((2, -3), dtype=float),
        b2=np.array(4, dtype=float),
        x=np.array((-1, 2, 0), dtype=float),
        y=np.array(3, dtype=float),
        loss='quadratic'
    ),
    SimpleNamespace(
        v1=np.array(((1, 2), (1, 0)), dtype=float),
        b1=np.array((-5, 2), dtype=float),
        v2=np.array(((2, -3), (1, 0), (-1, -1)), dtype=float),
        b2=np.array((1, 3, -2), dtype=float),
        x=np.array((-1, 2), dtype=float),
        y=np.array(2, dtype=int),
        loss='hinge'
    )
]
```

Problem 2.1

In [111…

```
def array_string(a):
    if np.isscalar(a) or a.ndim == 0:
```

```
        return '{:.4g}'.format(a)
    return '(' + ', '.join([array_string(x) for x in a]) + ')'
```

In [112…

```
def dot(a, b):
    return a * b if np.isscalar(a) or np.isscalar(b)\
        else np.einsum('...i,i...', a, b)
```

In [118…

```
class Layer():
    def __init__(self):
        self.x = None

    def forward(self, x):
        pass

    def backward(self, g):
        pass

    def training(self):
        self.keep = True

    def inference(self):
        self.keep = False

class AffineLayer(Layer):
    def __init__(self, v, b, name):
        self.v = v
        self.b = b
        self.name = name

    def forward(self, x):
        if self.keep:
            self.x = x

        y = (self.v @ x) + self.b
        return y

    def backward(self, g):
        n = 1 if len(self.b.shape)==0 else self.b.shape[0]
        jv = np.eye(n)[:, :, np.newaxis] * self.x
        jb = np.eye(n)
        jx = self.v
        self.jacobian = [jv,jb,jx]

        gv = dot(g,jv)
        gb = dot(g,jb)
        gx = dot(g,jx)
        self.gradient = [gv,gb,gx]

        return gx

class ReLULayer(Layer):
    def __init__(self, name):
        self.name = name

    def forward(self, x):
```

```
        if self.keep:
            self.x = x

        y = np.maximum(0,x)
        return y

    def backward(self, g):
        n = 1 if len(self.x.shape)==0 else self.x.shape[0]
        jx = np.eye(n) * (self.x > 0).astype(float)
        self.jacobian = [jx]

        gx = dot(g,jx)
        self.gradient = [gx]

        return gx

class QuadraticLoss():
    def forward(self, z, y):
        l = np.sum((z-y)**2)/2
        return l

    def backward(self, z, y):
        gz = z-y
        return gz

class HingeLoss():
    def forward(self, z, y):
        l = np.log(np.sum(np.e**z))-z[y]
        return l

    def backward(self, z, y):
        z_ast = np.zeros(z.shape[0])
        z_ast[y] = 1
        gz = (np.e**z)/np.sum(np.e**z) - z_ast
        return gz

class MLP():
    def __init__(self, layers, loss, decision):
        self.decision = decision
        self.layers = layers
        self.loss = loss
        self.z = None
        self.l = None
        self.gz = None

    def inference(self):
        self.keep = False
        for layer in self.layers:
            layer.inference()

    def training(self):
        self.keep = True
        for layer in self.layers:
            layer.training()

    def forward(self, x, y=None):
```

```python
        if y is None:
            self.inference()
        else:
            self.training()

        for layer in self.layers:
            x = layer.forward(x)
        z = x

        if y is None:
            return self.decision(z)
        else:
            self.z = z
            self.l = self.loss.forward(z,y)
            return self.l

    def backward(self, y):
        g = self.loss.backward(self.z,y)
        self.gz = g

        for layer in reversed(self.layers):
            g = layer.backward(g)

def run_test(t):
    layers = [
        AffineLayer(t.v1, t.b1, "Affine Layer 1"),
        ReLULayer("ReLU Layer"),
        AffineLayer(t.v2, t.b2, "Affine Layer 2")
    ]

    if t.loss == "quadratic":
        loss = QuadraticLoss()
        decision = lambda x: x
    else:
        loss = HingeLoss()
        decision = np.argmax

    mlp = MLP(layers, loss, decision)

    y_hat = mlp.forward(t.x)
    l = mlp.forward(t.x, t.y)
    mlp.backward(t.y)

    return y_hat, mlp

def snapshot(t):
    y_hat, mlp = run_test(t)

    layers = mlp.layers
    res = {'inputs':{}, 'jacobians':{}, 'gradients':{}}

    inputs = res['inputs']
    inputs['x'] = layers[0].x
    inputs['p'] = layers[1].x
    inputs['q'] = layers[2].x
    inputs['z'] = mlp.z
```

```python
    inputs['lambda'] = mlp.l

    jacobians = res['jacobians']
    jacobians['jv2'] = layers[2].jacobian[0]
    jacobians['jb2'] = layers[2].jacobian[1]
    jacobians['jq'] = layers[2].jacobian[2]
    jacobians['jp'] = layers[1].jacobian[0]
    jacobians['jv1'] = layers[0].jacobian[0]
    jacobians['jb1'] = layers[0].jacobian[1]

    gradients = res['gradients']
    gradients['gz'] = mlp.gz
    gradients['gv2'] = layers[2].gradient[0]
    gradients['gb2'] = layers[2].gradient[1]
    gradients['gq'] = layers[2].gradient[2]
    gradients['gp'] = layers[1].gradient[0]
    gradients['gv1'] = layers[0].gradient[0]
    gradients['gb1'] = layers[0].gradient[1]

    return res

def print_snapshot(n):
    print(f"Network {n}\n")
    res = snapshot(tests[n])

    print("\tForward Pass\n")
    for key in res['inputs'].keys():
        print(f"\t\t{key}: {array_string(res['inputs'][key])}\n")
    print("\n")

    print("\tLocal Jacobians\n")
    for key in res['jacobians'].keys():
        print(f"\t\t{key}: {array_string(res['jacobians'][key])}\n")
    print("\n")

    print("\tGradients\n")
    for key in res['gradients'].keys():
        print(f"\t\t{key}: {array_string(res['gradients'][key])}\n")
```

In [119…

```python
print_snapshot(0)
```

```
Network 0

        Forward Pass

                x: [-1.  2.]

                p: [-2.  5.  3.]

                q: [0. 5. 3.]

                z: [0. 3.]

                lambda: 10.0



        Local Jacobians

                jv2: [[[0. 5. 3.]
  [0. 0. 0.]]

 [[0. 0. 0.]
  [0. 5. 3.]]]

                jb2: [[1. 0.]
 [0. 1.]]

                jq: [[ 2.  1. -3.]
 [ 3. -1.  2.]]

                jp: [[0. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

                jv1: [[[-1.  2.]
  [-0.  0.]
  [-0.  0.]]

 [[-0.  0.]
  [-1.  2.]
  [-0.  0.]]

 [[-0.  0.]
  [-0.  0.]
  [-1.  2.]]]

                jb1: [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]



        Gradients

                gz: [4. 2.]
```

```
               gv2: [[ 0. 20. 12.]
[ 0. 10.  6.]]

               gb2: [4. 2.]

               gq: [14.  2. -8.]

               gp: [ 0.  2. -8.]

               gv1: [[  0.   0.]
[ -2.   4.]
[  8. -16.]]

               gb1: [ 0.  2. -8.]
```

In [120…

```
print_snapshot(1)
```

```
Network 1

        Forward Pass

                x: [-1.  2.  0.]

                p: [-2.  1.]

                q: [0. 1.]

                z: 1.0

                lambda: 2.0



        Local Jacobians

                jv2: [[[0. 1.]]]

                jb2: [[1.]]

                jq: [ 2. -3.]

                jp: [[0. 0.]
 [0. 1.]]

                jv1: [[[-1.  2.  0.]
  [-0.  0.  0.]]

 [[-0.  0.  0.]
  [-1.  2.  0.]]]

                jb1: [[1. 0.]
 [0. 1.]]



        Gradients

                gz: -2.0

                gv2: [[[-0. -2.]]]

                gb2: [[-2.]]

                gq: [-4.  6.]

                gp: [0. 6.]

                gv1: [[ 0.  0.  0.]
 [-6. 12.  0.]]

                gb1: [0. 6.]
```

In [109…

```
print_snapshot(2)
```

```
Network 2

        Forward Pass

                x: [-1.  2.]

                p: [-2.  1.]

                q: [0. 1.]

                z: [-2.  3. -3.]

                lambda: 6.0091744845917345



        Local Jacobians

                jv2: [[[0. 1.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 1.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 1.]]]

                jb2: [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

                jq: [[ 2. -3.]
 [ 1.  0.]
 [-1. -1.]]

                jp: [[0. 0.]
 [0. 1.]]

                jv1: [[[-1.  2.]
  [-0.  0.]]

 [[-0.  0.]
  [-1.  2.]]]

                jb1: [[1. 0.]
 [0. 1.]]



        Gradients

                gz: [ 0.00667641  0.99086747 -0.99754389]
```

```
                gv2: [[ 0.          0.00667641]
 [ 0.          0.99086747]
 [ 0.         -0.99754389]]

                gb2: [ 0.00667641  0.99086747 -0.99754389]

                gq: [2.00176418 0.97751465]

                gp: [0.         0.97751465]

                gv1: [[ 0.          0.        ]
 [-0.97751465  1.9550293 ]]

                gb1: [0.         0.97751465]
```

In [ ]:

In [ ]: