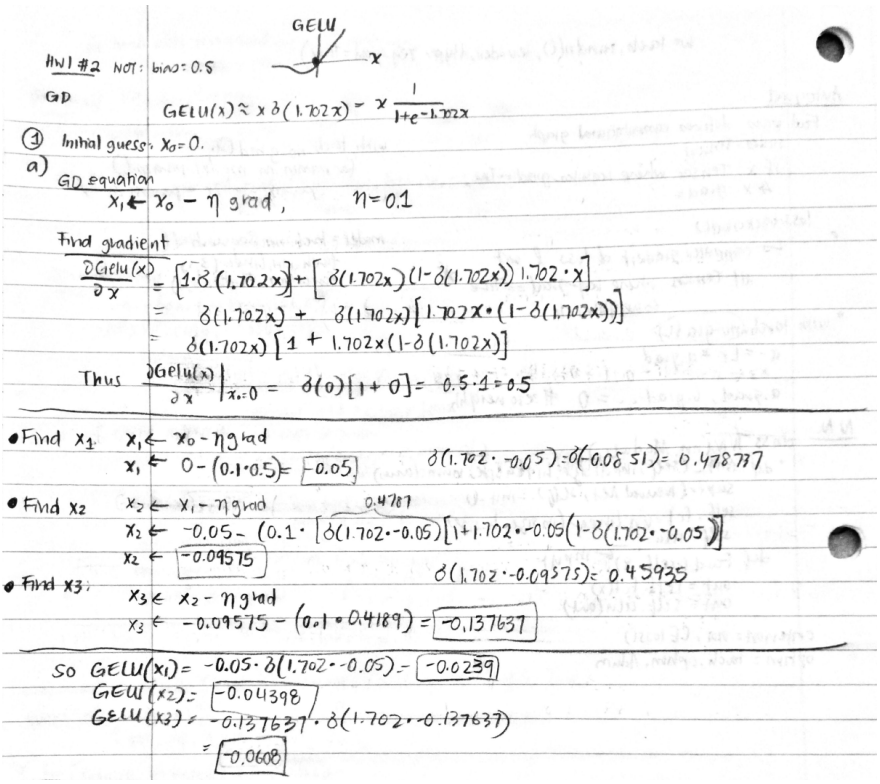
q1

Sunday, February 28, 2021 10:57 AM

Question 1:

*Note: For these questions I wrote a Python script for gradient descent with GeLU/momentum. to check my results. They are located in the files "gelu.py", "momentum.py".

1a) Calculation by Hand

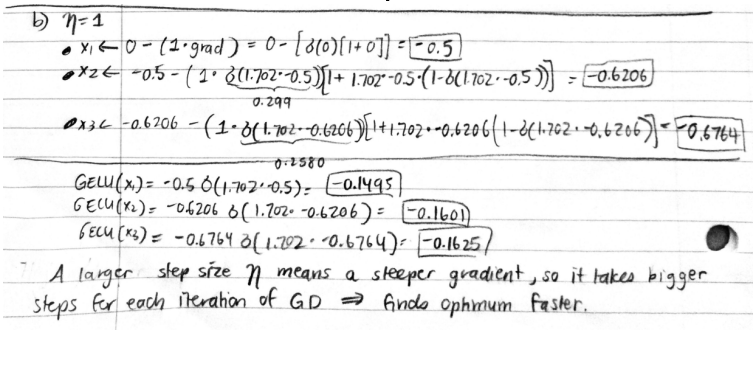


Output of Code

```
Using lr: 0.1, x_0:0, num_iters: 3
x_1:         -0.05
GeLU(x_1):   -0.02393689150943369
x_2:         -0.09575013021851926
GeLU(x_2):   -0.04398265469655064
x_3:         -0.13763771842581057
GeLU(x_3):   -0.06079478852684933
```

1b)

Calculation by Hand



Output of Code

```
Using lr: 1, x_0:0, num_iters: 3
x_1:         -0.5
GeLU(x_1):   -0.1496115633936199
x_2:         -0.6207780880345858
GeLU(x_2):   -0.1601399925974373
x_3:         -0.676497308917168
GeLU(x_3):   -0.16251748448720163
```

A larger step size means that a bigger step is taken for each iteration of gradient descent. This means it will move towards the minimum faster, which can be seen by comparing GeLU(x_3). From part a), GeLU(x_3) is –0.0608, which is less optimal than GeLU(x_3) = -0.1625 when the learning rate is 1. However, a higher learning rate poses the risk of overshooting near the minimum.

1c)

GeLU, x_0 = -3 (No momentum)

```
Using lr: 0.1, x_0:-3, num_iters: 3
x_1:         -2.997545167609435
GeLU(x_1):   -0.01813166529499688
x_2:         -2.995082708753491
GeLU(x_2):   -0.018192396733478253
x_3:         -2.9926125791343785
GeLU(x_3):   -0.018253507384506158
```

GeLU, x_0 = -3 (With Momentum)

GD with Momentum reaches a more optimal point/extremum (-0.0184 < -0.01825) in the same number of iterations, so it is generally faster.

```
GD with Momentum
lr: 1, x_0: -3, num_iters: 3, B: 0.9
x_1:         -2.997545167609435
GeLU(x_1):   -0.01813166529499688
x_2:         -2.9928733596019823
GeLU(x_2):   -0.01824704671202677
x_3:         -2.986191702877126
GeLU(x_3):   -0.01841325299731993
```

q1_code

Sunday, February 28, 2021 11:04 PM

Gradient Descent code for Q1 (gelu.py)

```python
import numpy as np
import math

# Gradient Descent, GeLU Activation
def sigmoid(x):
  return 1 / (1 + math.exp(-x))

def gelu(x):
  return x * sigmoid(1.702*x)

# gradient
def grad(x):
    return sigmoid(1.702*x) * (1+1.702*x*(1-sigmoid(1.702*x)))

# Set up variables and run GD
def main():
    lr = 0.1                                    # learning rate
    x = {0:-3,1:0,2:0,3:0}                      # store x[i]
    gelu_dict = {0:0, 1:0, 2:0, 3:0}            # store gelu(x[i])
    num_iters = 3

    for i in range(num_iters):
        x[i+1] = x[i] - lr * (grad(x[i]))       # update x[i+1]
        gelu_dict[i+1] = gelu(x[i+1])           # get gelu of new x[i+1]


if 'name == __main__':
    main()
```

GD with Momentum code (momentum.py)

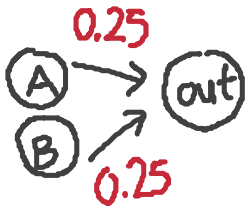
```python
# Set up variables and run perceptron
def main():
    B = 0.9                                     # Set up variables
    lr = 1
    x = {0:-3,1:0,2:0,3:0}                      # Store x0, x1, x2, x3
    v = {0:grad(x[0]), 0:0, 0:0, 0:0}           # Store v0, v1, v2, v3
    gelu_dict = {0:0, 1:0, 2:0, 3:0}            # Store geLU of x0, x1, x2, x3
    num_iters = 3

    print(f"GD with Momentum")
    print(f"lr: {lr}, x_0: {x[0]}, num_iters: {num_iters}, B: {B}")

    for i in range(3):
        v[i+1] = (B*v[i]) + ((1-B)*grad(x[i]))  # update v[i+1]
        x[i+1] = x[i] - (lr * v[i+1])           # update x[i+1]
                                                # get gelu of new x[i+1]
        gelu_dict[i+1] = gelu(x[i+1])
        print(f"GeLU(x_{i+1}): ", gelu_dict[i+1])


if 'name == __main__':
    main()
```

q2

Sunday, February 28, 2021 11:28 AM

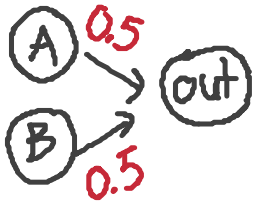
Note: all hidden nodes/output nodes have sigmoid activations, where output is 1 if the input >= 0.5.

a) AND gate



Both A, B must be 1 for the output to be 1
0.25*1 + 0.25*1 >= 0.5 --> output = 1

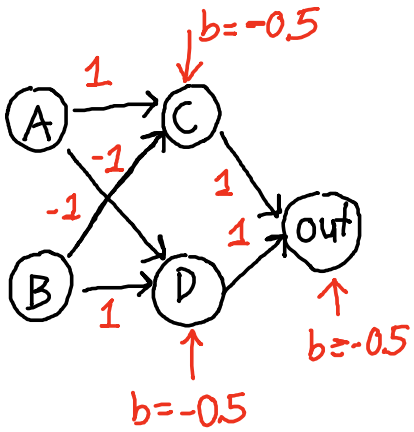
b) OR gate



Only one of A, B needs to be 1 for the output to be 1
0.25*1 + 0.25*0 >= 0.5 --> output = 1

c) XOR gate

Exactly one of A,B must be 1 for the output to be 1



A=0, B=0
C = σ(0+0-0.5) = 0
D = σ(0+0-0.5)=0
Out = σ(0+0-0.5)=**0**

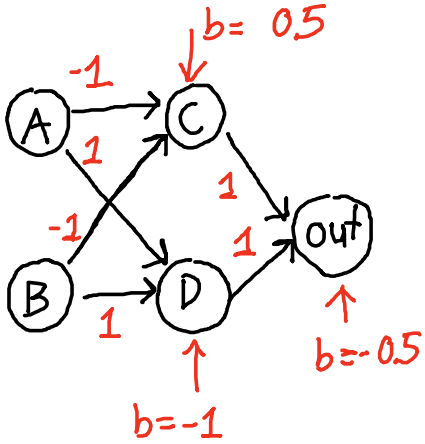
A=0, B=1
C = σ(0-1-0.5) = 0
D = σ(0+1-0.5)=1
Out = σ(0+1-0.5)=**1**

A=1, B=0
C = σ(1+0-0.5) = 1
D = σ(-1+0-0.5)=0
Out = σ(1+0-0.5)=**1**

A=1, B=1
C = σ(1-1-0.5) = 0
D = σ(-1+1-0.5)=0
Out = σ(0+0-0.5)=**0**

d) XNOR gate

A,B must both be 1 or both be 0 for the output to be 1



A=0, B=0
C = σ(0+0+0.5) = 1
D = σ(0+0-1)=0
Out = σ(1+0-0.5)=**1**

A=0, B=1
C = σ(0-1+0.5) = 0
D = σ(0+1-1)=0
Out = σ(0+0-0.5)=**0**

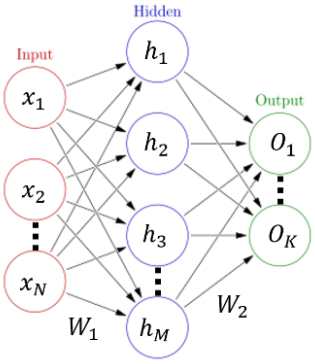
A=1, B=0
C = σ(-1+0+0.5) = 0
D = σ(1+0-1)=0
Out = σ(0+0-0.5)=**0**

A=1, B=1
C = σ(-1-1+0.5) = 0
D = σ(1+1-1)=1
Out = σ(0+1-0.5)=**1**

q3

Sunday, February 28, 2021 11:54 AM

$$f_1 = xW_1 + b_1$$
$$a = \sigma(f_1)$$
$$f_2 = aW_2 + b2$$
$$o = S(f_2)$$



$x = [1 \times N]$ $f_2 = [1 \times K]$
$h = [1 \times M]$
$W_1 = [N \times M]$, $W_2 = [M \times K]$
$o = [1 \times K]$
$f_1 = [1 \times M]$
$a = [1 \times M]$

$E(o) = -\sum y_i \log o_i$
$= -(y_1 \log o_1 + \ldots + y_k \log$

$\frac{\partial E(o)}{\partial o} = \begin{bmatrix} -\frac{y_1}{o_1} & -\frac{y_2}{o_2} & \ldots & -\frac{y_k}{o_k} \end{bmatrix}$

$\frac{\partial o}{\partial f_2} = \begin{bmatrix} o_1(1-o_1) & -o_1 o_2 & \cdots & -o \\ -o_2 o_1 & o_2(1-o_2) & \cdots & \\ -o_K o_1 & \vdots & & o_k(\end{bmatrix}$

Derivative of softmax

a)

$$\frac{\partial E(o)}{\partial f_2} = \left(\frac{\partial E(o)}{\partial o}\right)\left(\frac{\partial o}{\partial f_2}\right)$$

$$= \begin{bmatrix} -y_1/o_1 & -y_2/o_2 & \ldots \end{bmatrix} \begin{bmatrix} o_1(1-o_1) & \cdots & -o_1 o_k \\ -o_2 o_1 & o_2(1-o_2) & \cdots \\ \vdots & & \\ -o_K o_1 & \cdots & o_k(1-o_k) \end{bmatrix}$$

$$= \begin{bmatrix} -y_1 + o_1\left(\sum y_k\right) & -y_2 + o_2\left(\sum y_k\right) & \ldots & -y_k + o_k\left(\sum y_k\right) \end{bmatrix}$$

$$= \begin{bmatrix} o_1 - y_1 & o_2 - y_2 & o_3 - y_3 & \cdots & o_k - y_k \end{bmatrix}$$

b) $1\times N \quad 1\times K \quad K\times M \quad M\times M \quad M\times N$

$$\frac{\partial E(o)}{\partial x} = \frac{\partial E(o)}{\partial f_2}\frac{\partial f_2}{\partial a}\frac{\partial a}{\partial f_1}\frac{\partial f_1}{\partial x}$$

Since $f_2 = aW_2 + b_2$, $\frac{\partial f_2}{\partial a} = W_2^T$

Since $f_1 = xW_1 + b_1$, $\frac{\partial f_1}{\partial x} = W_1^T$

$$\frac{\partial a}{\partial f_1} = \frac{\partial(\sigma(f_1))}{\partial f_1} = \begin{matrix} \\ M \end{matrix} \overset{M}{\begin{bmatrix} a_1(1-a_1) & \cdots & 0 \\ 0 & a_2(1-a_2) & \vdots \\ \vdots & & \ddots \\ 0 & \cdots & a_M(1-a_M) \end{bmatrix}}$$

So

$$\frac{\partial E(o)}{\partial x} = \begin{bmatrix} o_1 - y_1 & o_2 - y_2 & \ldots & o_k - y_k \end{bmatrix} W_2^T \begin{bmatrix} a_1(1-a_1) & 0 & \cdots & 0 \\ 0 & a_2(1-a_2) & & \vdots \\ \vdots & & \ddots & \\ 0 & \cdots & & a_M(1-a_M) \end{bmatrix} W_1^T$$

q4

Sunday, February 28, 2021 12:36 PM

④ CNN

4×4

3	5	2	3
9	1	8	4
6	4	3	7
7	0	2	4

$*$

f_1 = 3×3

-1	0.5	-2
2	0	1
0	1	1.5

f_2 = 2×2

-1	0.5
2	0

a) convolve F with f to get F′ $n=4$, $f=3$

want output size: $\left(\frac{n+2p-f}{s}+1\right)$

$\left(\frac{4+2p-3}{1}+1\right)=4 \Rightarrow \boxed{p=1}$

zero pad image

0	0	0	0	0	0
0	3	5	2	3	0
0	9	1	8	4	0
0	6	4	3	7	0
0	7	0	2	4	0
0	0	0	0	0	0

$*$

-1	0.5	-2
2	0	1
0	1	1.5

$=$

15.5	21	27	8
4.5	30	9.5	22.5
13.5	-6.5	18	4
-5	6	12.5	4.5

$= F'$

b) filter 2, $p=1$. $n=4$, $f=2$

$\left(\frac{n+2p-f}{s}+1\right)=4$

$\left(\frac{4+2-2}{s}+1\right)=4$ $\quad \frac{4+2-2}{s}=3$ $\quad 3s=4$

↳ $s=4/3$

No, since the stride can't be fractional.

c) Average pool

$\frac{15.5+21+4.5+30}{4}$	$\frac{27+8+9.5+22.5}{4}$
$\frac{13.5-6.5-5+6}{4}$	$\frac{18+4-12.5+4.5}{4}$

$=$

17.75	16.75
2	3.5

d)

Use filter with uniform weights 0.25 to "average" across 4 cells

$f=$

1/4	1/4
1/4	1/4

2×2 no padding, stride: 2

output size $=\left(\frac{n+2p-f}{s}+1\right)=\left(\frac{4+0-2}{2}+1\right)=2$ ✓