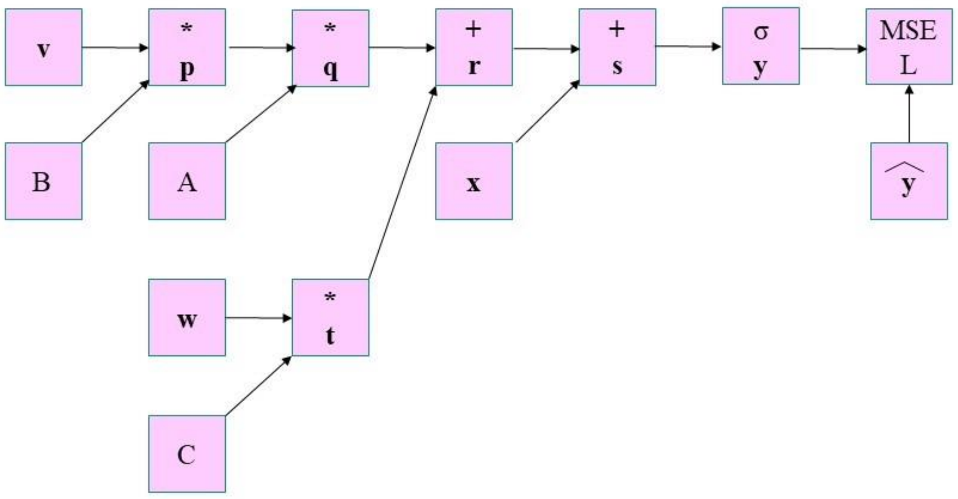
Solution of Gradiance Homework – 5

– By Anirban Haldar

Neural Networks

1. **Here is a compute graph:**



There are many relationships among the variables represented by these nodes. Identify, in the list below, the relationship that is FALSE.

a) **t = Cw**
b) **y = r + x**
c) **r = Cw + q**
d) **r = q + t**

⇨ Here, if we go by the options, then we will find

Option a : t = Cw

t = C * w == Cw ✓

Option b : y = r + x

y = σ(s)

= σ(r + x) ≠ (r + x) ✗

Option c : r = Cw + q

r = q + t

= q + (C * w) == Cw + q ✓

Option d : r = q + t

r = q + t ✓

So, the correct answer is Option **b) y = r + x**

2. **Two ways to turn the step function into a continuous function that approximates a step function are the (logistic) sigmoid s and the hyperbolic tangent t. Suppose a node has a weighted sum of inputs x. There is a relatively simple relationship between s(x) and t(x). Your task is to determine what t is, as a function of s (not x), and then identify the true statement below.**
   - **a) If s = 1/5, then t = -21/23.**
   - **b) If s = 3/5, then t = 5/13.**
   - **c) If s = 3/5, then t = 15/33.**
   - **d) If s = 3/10, then t = -31/41.**

⇨ We know, the value of sigmoid and hyper-tangent as

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} \quad \rightarrow \quad e^x = \frac{S(x)}{(1-S(x))}$$

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{\frac{S(x)}{(1-S(x))} - \frac{(1-S(x))}{S(x)}}{\frac{S(x)}{(1-S(x))} + \frac{(1-S(x))}{S(x)}}$$

$$\boxed{tanhx = \frac{2S(x)-1}{2S^2(x)-2S(x)+1}}$$

Now, if we write a small program to calculate the values, we will find,

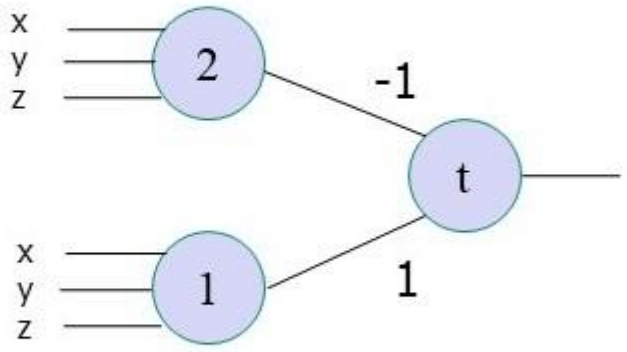
```
def tan_h(s):
    val = (2*s - 1) / (2*(s**2) - 2*s + 1)
    return round(val, 6)

print('Option 1 :', tan_h(1/ 5) == round(-21/23, 6))
print('Option 2 :', tan_h(3/ 5) == round(  5/13, 6))
print('Option 3 :', tan_h(3/ 5) == round( 15/33, 6))
print('Option 4 :', tan_h(3/10) == round(-31/41, 6))
```

```
Option 1 : False
Option 2 : True
Option 3 : False
Option 4 : False
```

So the correct option is **If s = 3/5, then t = 5/13.**

3. **Here is a small neural net with inputs x, y, and z.**



The two nodes in the input layer have thresholds 2 and 1, respectively, and the weights on all their inputs is 1. The output node has weight -1 for the first input node, and weight +1 for the second input node. The intent is that the net will have output 1 if exactly one of the inputs x, y, and z is 1, and 0 for any other combination of inputs. For what values of t, the threshold of the output node, will the net function as desired?

a) t = -0.5
b) t = 10
c) t = 0.8
d) t = -2

⇨ Here, by the node structure, we can say that

- The 1st node (t=2) will only be activated, when we have at-least two 1s.
- The 2nd node (t=1) will only be activated, when we have at-least one 1s.

Now to get exactly one 1, we need to capture the situation, when 1st node is off and 2nd is on. Now all the possible situations are

Situation	n1	n2	n1 * (-1) + n2 * 1
More than two 1s	1	1	0
Exactly one 1	0	1	1
No 1s	0	0	0

From the table, we can conclude that t must be greater than 0 but less than or equals to 1, i.e. 0 < t <= 1.
So, the correct answer is **t = 0.8**

4. Suppose we have a convolutional layer with a single filter:

1	0	-1
0	1	0
-1	0	1

We apply this filter to a small array with the following values:

1	3	6	10
1	4	10	20
1	5	15	35
1	6	21	56

Assuming a pad of 1, apply the filter to the array to get another 4-by-4 array of values. Then, identify in the list below the number that is NOT in the resulting array.

a) 23 **b) -4** **c) 24** **d) 3**

⇨ So here, initially, we will apply a pad of 1 pixel wide, so that we will get the output size same as input matrix size. As

```
arr = np.matrix([[ 1, 3,  6, 10],
                 [ 1, 4, 10, 20],
                 [ 1, 5, 15, 35],
                 [ 1, 6, 21, 56]])

pad_1 = np.zeros((6, 6), dtype='int16')
pad_1[1:5, 1:5] = arr
pad_1
```

```
array([[ 0,  0,  0,  0,  0,  0],
       [ 0,  1,  3,  6, 10,  0],
       [ 0,  1,  4, 10, 20,  0],
       [ 0,  1,  5, 15, 35,  0],
       [ 0,  1,  6, 21, 56,  0],
       [ 0,  0,  0,  0,  0,  0]], dtype=int16)
```

Now, for each submatrix of size 3 x 3 in the padded matrix, we will multiply respective elements of our filter matrix and sum the results to perform convolution operation. as

```
filt = np.matrix([[ 1, 0,-1],
                  [ 0, 1, 0],
                  [-1, 0, 1]], dtype='int16')

res = []
for i in range(4):
    for j in range(4):
        res.append(np.sum(np.multiply(pad_1[i:i+3, j:j+3], filt)))

np.array(res).reshape(4, 4)
```

```
array([[ 5, 12, 22,  0],
       [ 3, 13, 33, 11],
       [ 3, 16, 49, 24],
       [-4, -8, -9, 71]])
```

Now, from the options, all the elements -4, 24 & 3 are present in the resultant matrix except the element 23, so the correct answer is **23**

5. **In this question, you will be asked to do a softmax calculation. However, to make the computation easier, we shall redefine the softmax to use the number 2 in place of e. That is, given [x_1, x_2,..., x_n], our "softmax" function will map x_i to 2^{x_i} divided by the sum of 2^{x_j} for $1 \leq j \leq n$.**

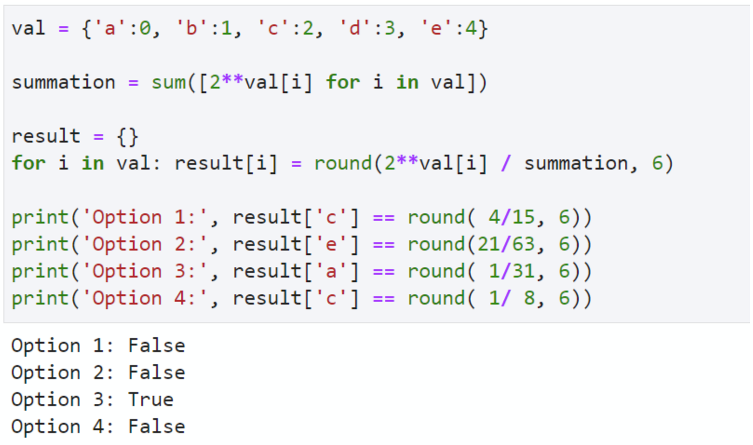
   Suppose that a net has five output nodes a, b, c, d, and e. The weighted sums of inputs to these five nodes are 0, 1, 2, 3, and 4, respectively. Compute the "softmax" values for each of these nodes and then identify the correct statement below.
   - **a) The output for node c is 4/15.**
   - **b) The output for node e is 32/63.**
   - **c) The output for node a is 1/31.**
   - **d) The output for node c is 1/8.**

⇨ We know, the actual softmax function is $S(y)_i = \dfrac{\exp(y_i)}{\sum\limits_{j=1}^{n} \exp(y_j)}$

But according to the question, we will replace the exp with 2, then the new function will be

$$S(y)_i = \frac{2^{y_i}}{\sum_{j=1}^{n} 2^{y_i}}$$

Now, we can write a small code as



```python
val = {'a':0, 'b':1, 'c':2, 'd':3, 'e':4}

summation = sum([2**val[i] for i in val])

result = {}
for i in val: result[i] = round(2**val[i] / summation, 6)

print('Option 1:', result['c'] == round( 4/15, 6))
print('Option 2:', result['e'] == round(21/63, 6))
print('Option 3:', result['a'] == round( 1/31, 6))
print('Option 4:', result['c'] == round( 1/ 8, 6))
```

```
Option 1: False
Option 2: False
Option 3: True
Option 4: False
```