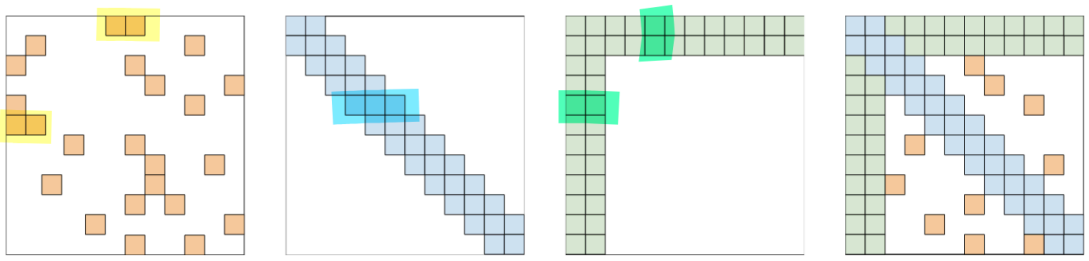
Previous work



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Figure 1: Building blocks of the attention mechanism used in BIGBIRD. White color indicates absence of attention. (a) random attention with $r = 2$, (b) sliding window attention with $w = 3$ (c) global attention with $g = 2$. (d) the combined BIGBIRD model.

Introduction

My goal was to generate python functions, using only math, numpy and pure python, with a algorithm approche, simplicity and easy to port to C / C ++. For each attention mask (a, b, c and d), i have written the following features :

- add a mask to a matrix wich zero out some value, based on the concerned parameter,
- same, but based on a given sparsity
- generate artificial matrix of only ones and zero based on the mask
- a test function which shows how to use it and the output

Code

Imports

In [311…
```
import numpy as np
import matplotlib.pyplot as plt
import math
```

UTILS

In [312…
```
def get_nb_non_zero(matrix):
    return np.count_nonzero(matrix)
```

In [313…
```
def get_density(matrix, length):
    return float(get_nb_non_zero(matrix)) / float(length * length)
```

In [314…
```python
def get_sparsity(matrix, length):
    return 1.0 - get_density(matrix, length)
```

In [315…
```python
def show_matrix_infos(matrix, length):
    # conditions : shape(matrix) = (length, length)
    sparsity = get_sparsity(matrix, length)
    text = f"Given length : ({length}, {length}) and calculated length : {np
    plt.title(label=text)
    plt.imshow(matrix, cmap='gray', interpolation='nearest')
    plt.colorbar()
    plt.show()
```

RANDOM ATTENTION

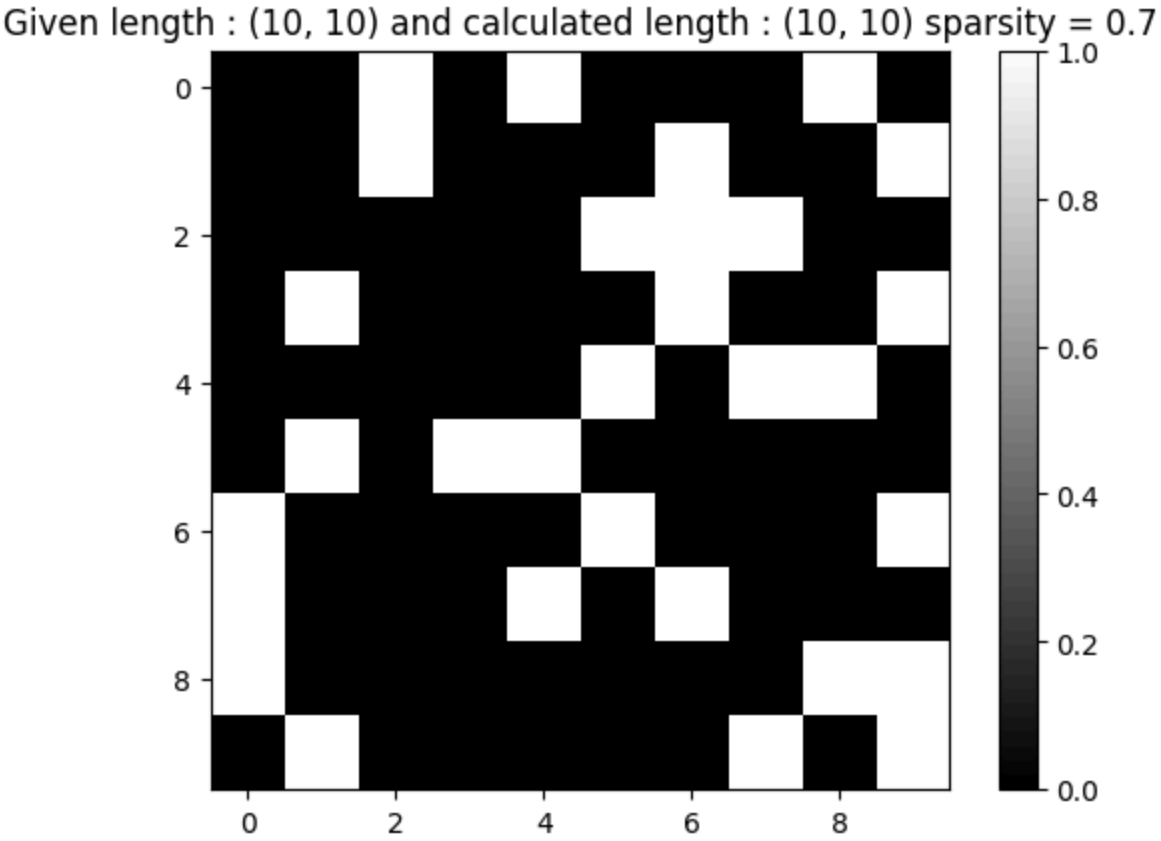
By number of non-zeros per row

In [316…
```python
def add_random_attention_mask(matrix, length, nz_per_row):
    # conditions : shape(matrix) = (length, length), nz_per_row <= length
    res = matrix.copy()
    z_per_row = length - nz_per_row
    masked_indices = []
    for row in range(length):
        column_indices = np.random.choice(a=length, size=z_per_row ,replace=
        for col in column_indices:
            masked_indices.append((row,int(col)))
    for mi in masked_indices:
        res[mi] = 0
    return res
```

In [317…
```python
def generate_matrix_with_random_attention_mask(length, nz_per_row):
    # conditions : nz_per_row <= length
    matrix = np.ones((length, length))
    matrix = add_random_attention_mask(matrix=matrix, length=length, nz_per_
    return matrix
```

In [318…
```python
def test_generate_matrix_with_random_attention_mask():
    length = 10
    nz_per_row = 3
    matrix = generate_matrix_with_random_attention_mask(length=length, nz_pe
    show_matrix_infos(matrix=matrix, length=length)

test_generate_matrix_with_random_attention_mask()
```



By sparsity

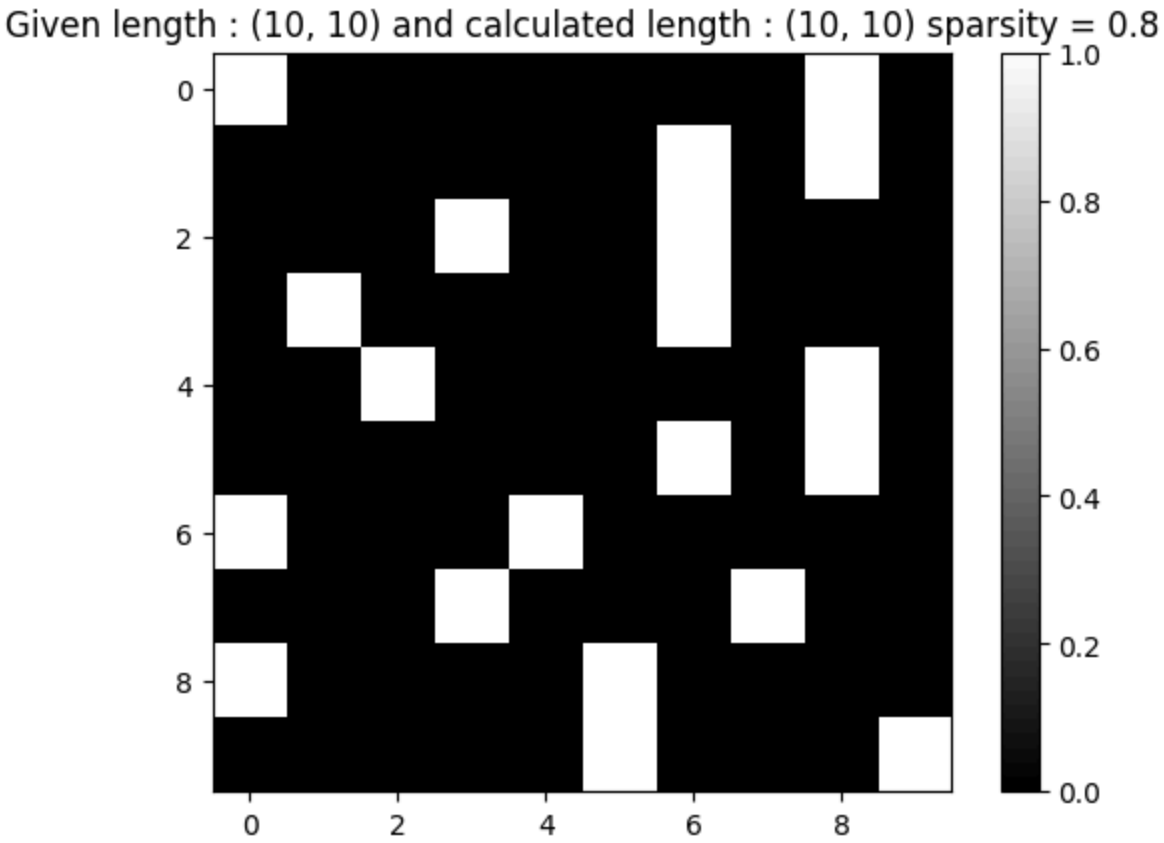
In [319…
```
def sparsity_to_nz_per_row(length, sparsity):
    # conditions : 0 <= sparsity <= 1
    return round(length * (1 - sparsity))
```

In [320…
```
def add_random_attention_mask_with_sparsity(matrix, length, sparsity):
    # conditions : 0 <= sparsity <= 1
    nz_per_row=sparsity_to_nz_per_row(length=length, sparsity=sparsity)
    return add_random_attention_mask(matrix=matrix, length=length, nz_per_ro
```

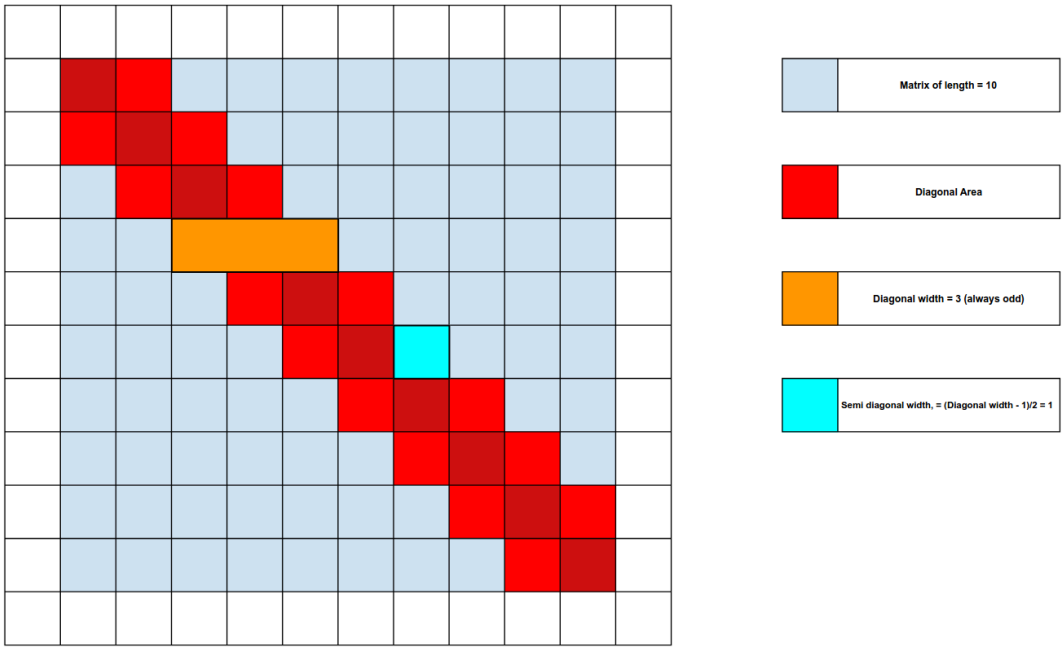
In [321…
```
def generate_matrix_with_random_attention_mask_with_sparsity(length, sparsit
    # conditions : 0 <= sparsity <= 1
    matrix = np.ones((length, length))
    matrix = add_random_attention_mask_with_sparsity(matrix=matrix, length=l
    return matrix
```

In [322…
```
def test_generate_matrix_with_random_attention_mask_with_sparsity():
    length = 10
    sparsity = 0.8
    matrix = generate_matrix_with_random_attention_mask_with_sparsity(length
    show_matrix_infos(matrix=matrix, length=length)

test_generate_matrix_with_random_attention_mask_with_sparsity()
```



WINDOW ATTENTION



Utils

In [323…
```python
def diagonal_area(length,diagonal_width ):
    # conditions : length
    if(diagonal_width == 0):
        return 0
    else:
        n = length
        #semi diagonal widht
        sdw = diagonal_width // 2 # (diagonal_width / 2 - 1 because is odd)
        da = n * ( 1 + 2 * sdw ) - sdw * (sdw + 1)
        return da
```

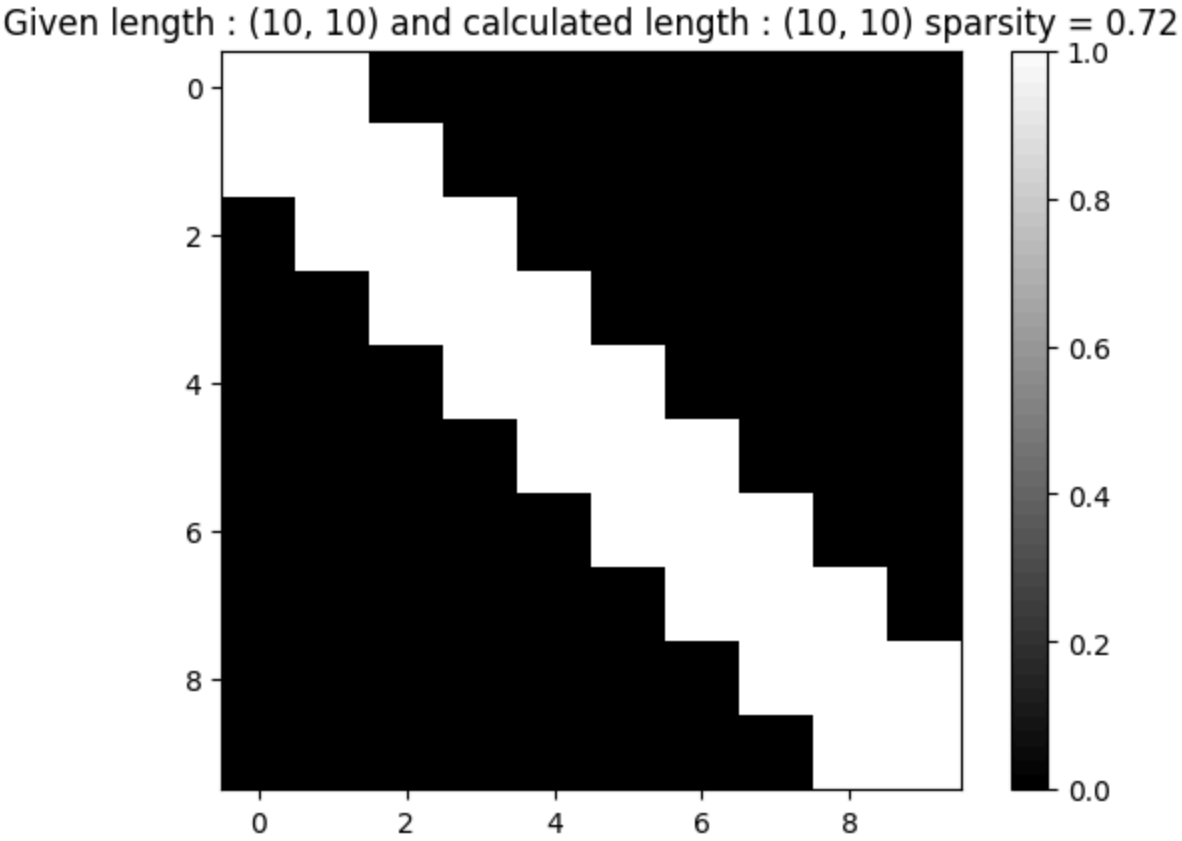
By diagonal width

In [324…
```python
def add_window_attention_mask(matrix, length, diagonal_width):
    # conditions : shape(matrix) = (length, length), 0 <= diagonal_width <=
    res = matrix.copy()
    if (diagonal_width > 0):
        mask = np.zeros(shape=(length, length), dtype=bool)
        if(diagonal_width == 1):
            mask[np.diag_indices(length)] = True
        else:
            #semi diagonal widht
            sdw = diagonal_width // 2 # (diagonal_width / 2 - 1 because is d
            for i in range(length):
                for j in range(max(0, i-sdw), min(length, i+sdw+1)):
                    mask[i,j] = True
        res[~mask] = 0
    return res
```

In [325…
```python
def generate_matrix_with_window_attention_mask(length, diagonal_width):
    # conditions :  0 <= diagonal_width <= 2*length - 1 (cover full matrix),
    matrix = np.ones((length, length))
    matrix = add_window_attention_mask(matrix=matrix, length= length, diagor
    return matrix
```

In [326…
```python
def test_generate_matrix_with_window_attention_mask():
    length = 10
    diagonal_width = 3
    matrix = generate_matrix_with_window_attention_mask(length=length, diagc
    show_matrix_infos(matrix,length)

test_generate_matrix_with_window_attention_mask()
```



By sparsity

In [327…

```python
def best_diagonal_width_from_sparsity(length, sparsity):
    n = length
    density = 1.0 - sparsity
     # ideal diagonal aera
    da = n * n * density
    # from this point, all is explained in the related document
    a = -1
    b = 2 * n - 1
    c = n - da
    det = b * b - 4 * a * c
    x = (-b + math.sqrt(det))/(2 * a)

    sdw = round(x)

    dw = 2 * sdw + 1

    if(dw < 0) : dw = 0
    elif(dw > 2*n - 1): dw = 2*n - 1
    # print(f"For matrix of size: {n} and given sparsity: {sparsity}, ideal
    return dw
```

In [328…

```python
def test_diagonal_width_from_sparsity():
    for length in {10, 250, 900}:
        for sparsity in {0.1, 0.5, 0.88}:
            da = best_diagonal_width_from_sparsity(length, sparsity)
            real_sparsity = 1.0 - (float(diagonal_area(length,da))/float(ler
```

```
            print(f"for lenght={length}, given sparsity ={sparsity}, optimal

test_diagonal_width_from_sparsity()
```

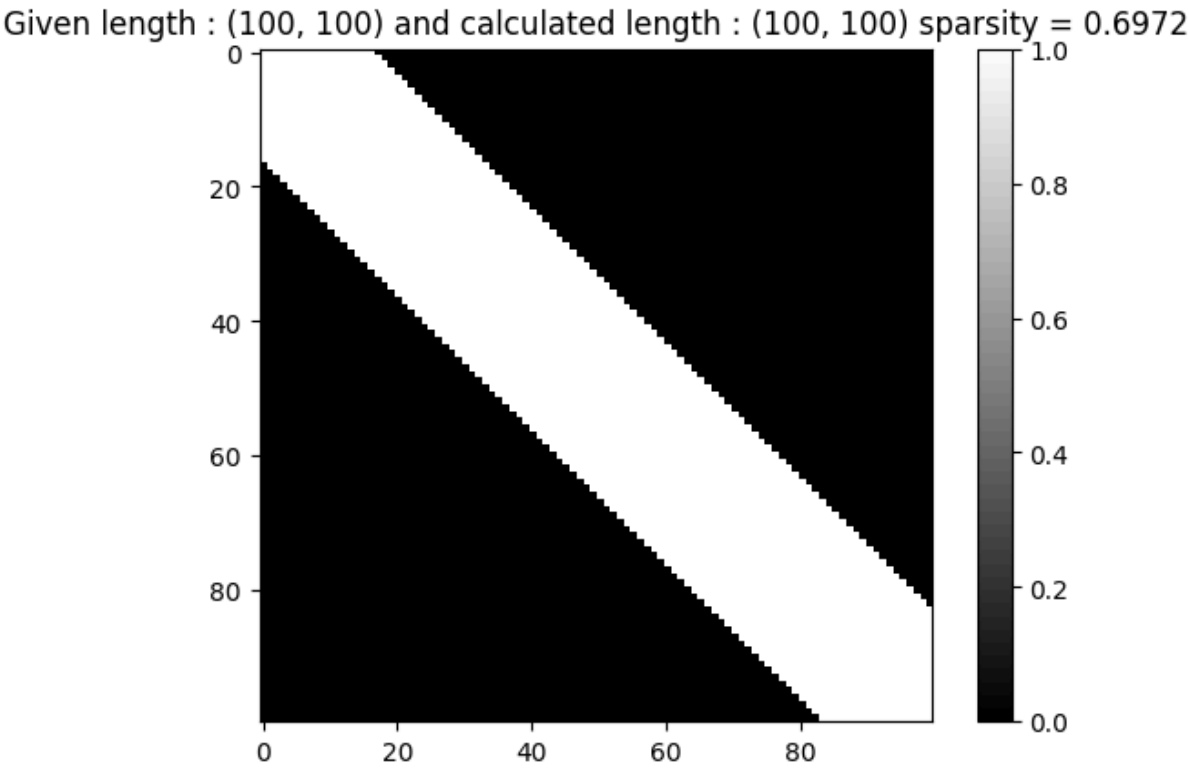
```
for lenght=10, given sparsity =0.1, optimal diagonal width=13, and real spar
sity = 0.12
for lenght=10, given sparsity =0.5, optimal diagonal width=5, and real spars
ity = 0.56
for lenght=10, given sparsity =0.88, optimal diagonal width=1, and real spar
sity = 0.90
for lenght=250, given sparsity =0.1, optimal diagonal width=341, and real sp
arsity = 0.10
for lenght=250, given sparsity =0.5, optimal diagonal width=147, and real sp
arsity = 0.50
for lenght=250, given sparsity =0.88, optimal diagonal width=31, and real sp
arsity = 0.88
for lenght=900, given sparsity =0.1, optimal diagonal width=1231, and real s
parsity = 0.10
for lenght=900, given sparsity =0.5, optimal diagonal width=527, and real sp
arsity = 0.50
for lenght=900, given sparsity =0.88, optimal diagonal width=111, and real s
parsity = 0.88
```

In [329…
```
def add_window_attention_mask_with_sparsity(matrix, length, sparsity):
    # conditions : 0 <= sparsity <= 1
    dw = best_diagonal_width_from_sparsity(length, sparsity)
    return add_window_attention_mask(matrix=matrix, length=length, diagonal_
```

In [330…
```
def generate_matrix_with_window_attention_mask_with_sparsity(length, sparsit
    # conditions :  0 <= diagonal_width <= 2*length - 1 (cover full matrix),
    matrix = np.ones((length, length))
    matrix = add_window_attention_mask_with_sparsity(matrix=matrix, length=
    return matrix
```

In [331…
```
def test_generate_matrix_with_window_attention_mask_with_sparsity():
    length = 100
    sparsity = 0.7
    matrix = generate_matrix_with_window_attention_mask_with_sparsity(length
    show_matrix_infos(matrix,length)

test_generate_matrix_with_window_attention_mask_with_sparsity()
```



GLOBAL ATTENTION

Utils

In [332…
```python
def global_attention_aera(length,global_attention_width):
    w = global_attention_width
    n = length
    return (2 * w * n) - (w * w)

print(global_attention_aera(10,2))
```

36

In [333…
```python
def add_global_attention_mask():
    return
```