Day 4 programs

1. Find the performance of mojo programming with optimization and vectorization for matrix transpose problem statement. [Note: size of matrix: 128 x 128]

MOJO > DAY-4 > 1.mojo

```
from sys.intrinsics import sizeof
from memory import memset_zero
from time import now

struct Matrix:
    var data: DTypePointer[DType.float32]
    var rows: Int
    var cols: Int

    fn __init__(inout self, rows: Int, cols: Int):
        self.rows = rows
        self.cols = cols
        self.data = DTypePointer[DType.float32].alloc(rows * cols)
        memset_zero(self.data, rows * cols)

    fn __del__(owned self):
        self.data.free()

    fn __getitem__(self, row: Int, col: Int) -> Float32:
        return self.data.load(row * self.cols + col)

    fn __setitem__(self, row: Int, col: Int, val: Float32):
        self.data.store(row * self.cols + col, val)

fn min(a: Int, b: Int) -> Int:
    if a < b:
        return a
    return b
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
chocku@imperio:/mnt/c/Users/chock/OneDrive/Desktop/MOJO$ cd DAY-4
chocku@imperio:/mnt/c/Users/chock/OneDrive/Desktop/MOJO/DAY-4$ mojo 1.mojo
Matrix size: 128 x 128
Unoptimized transpose time: 2.9999999999999997e-08 seconds
Optimized transpose time: 2e-08 seconds
Speedup: 1.4999999999999998 x
chocku@imperio:/mnt/c/Users/chock/OneDrive/Desktop/MOJO/DAY-4$
```

2. Find the performance of mojo programming with parallelization for matrix addition and subtraction in a single program. [Note: size of matrix: 128 x 128]

In [2]:

```
from benchmark import Unit
from sys.intrinsics import strided_load
from math import CeilDivableRaising
from memory import memset_zero
from memory.unsafe import DTypePointer
from random import rand, random_float64
from sys.info import simdwidthof
from runtime.llcl import Runtime
from algorithm import parallelize
import time
```

In [3]:

```
alias type = DType.float32

struct Matrix[rows: Int, cols: Int]:
    var data: DTypePointer[type]

    # Initialize zeroeing all values
    fn __init__(inout self):
        self.data = DTypePointer[type].alloc(rows * cols)
        memset_zero(self.data, rows * cols)

    # Initialize taking a pointer, don't set any elements
    fn __init__(inout self, data: DTypePointer[type]):
        self.data = data

    # Initialize with random values
    @staticmethod
    fn rand() -> Self:
        var data = DTypePointer[type].alloc(rows * cols)
        rand(data, rows * cols)
        return Self(data)

    fn __getitem__(self, y: Int, x: Int) -> Scalar[type]:
        return self.load[1](y, x)

    fn __setitem__(self, y: Int, x: Int, val: Scalar[type]):
        self.store[1](y, x, val)

    fn load[nelts: Int](self, y: Int, x: Int) -> SIMD[type, nelts]:
        return self.data.load[width=nelts](y * self.cols + x)

    fn store[nelts: Int](self, y: Int, x: Int, val: SIMD[type, nelts]):
        return self.data.store[width=nelts](y * self.cols + x, val)
```

In [4]:

```
alias nelts = simdwidthof[DType.float32]() * 2
```

In [5]:

```
from algorithm import vectorize
from algorithm import parallelize

#write similar code for fn matadd_parallelized
fn matadd_parallelized(C: Matrix, A: Matrix, B: Matrix):
    @parameter
    fn calc_row(m: Int):
        for n in range(A.cols):
            @parameter
            fn add[nelts : Int](n : Int):
                C.store[nelts](m,n, A[m,n] + B[m,n])
            vectorize[add, nelts, size = C.cols]()
    parallelize[calc_row](C.rows, C.rows)
fn matsub_parallelized(C: Matrix, A: Matrix, B: Matrix):
    @parameter
    fn calc_row(m: Int):
        for n in range(A.cols):
            @parameter
            fn sub[nelts : Int](n : Int):
                C.store[nelts](m,n, A[m,n] - B[m,n])
            vectorize[sub, nelts, size = C.cols]()
    parallelize[calc_row](C.rows, C.rows)
```

In [6]:

```
alias M = 128
```

3. WAP using multi-layer perceptron model to implement the functionality of XOR gate by using 3 different activation functions and show the change in error.

MOJO > DAY-4 > 3.mojo

```
from math import exp, tanh
from random import seed, random_float64
from time import now


struct Array:
    var data: Pointer[Float64]
    var size: Int

    fn __init__(inout self, size: Int):
        self.size = size
        self.data = Pointer[Float64].alloc(self.size)

    fn __init__(inout self, size: Int, default_value: Float64):
        self.size = size
        self.data = Pointer[Float64].alloc(self.size)
        for i in range(self.size):
            self.data.store(i, default_value)

    fn __copyinit__(inout self, copy: Array):
        self.size = copy.size
        self.data = Pointer[Float64].alloc(self.size)
        for i in range(self.size):
            self.data.store(i, copy[i])

    fn __getitem__(self, i: Int) -> Float64:
     return self.data.load(i)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

```
Epoch  9800  loss =  0.36784116604498074
Epoch  9900  loss =  0.36784116604498074

Predictions:
0 0 1
0 1 0
1 0 1
1 1 0
chocku@imperio:/mnt/c/Users/chock/OneDrive/Desktop/MOJO/DAY-4$
```