# Fully-connected neural networks

## 1 Problem description

We are interested in the parallelization of the inference phase of a neural network composed of multiple fully-connected layers of constant size. With each layer $l = 0 ... L-1$ of the network is associated a weight matrix $W^l \in \mathbb{R}^{n \times n}$, a bias vector $b^l \in \mathbb{R}^n$, an input matrix $X^l \in \mathbb{R}^{n \times m}$ and an output $X^{l+1} \in \mathbb{R}^{n \times m}$ in such a way that

$$X^{l+1} = \sigma\left(W^l X^l + B^l\right)$$

where $B = b^l e_m^T$, $e_m$ is a vector of ones of size $m$ and $\sigma$ an activation function assumed to be the same for all layers. We assume $m \leq n$ and all the $W$, $X$ and $B$ are partitioned into blocks of size $m \times m$

$$\begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1N} \\ W_{21} & W_{22} & \cdots & W_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ W_{N1} & W_{N2} & \cdots & W_{NN} \end{bmatrix} \cdot \begin{bmatrix} X_{11} \\ X_{21} \\ \vdots \\ X_{N1} \end{bmatrix} + \begin{bmatrix} B_{11} \\ B_{21} \\ \vdots \\ B_{N1} \end{bmatrix}$$

where $N = n/m$. In the provided code, all the $W$ and $b$ are stored in the `Layers` array of size $L$ such that `Layers[l].W` and `Layers[l].b` store $W^l$ and $b^l$, respectively. The $X$ matrices instead are stored in the `Datas` array of size $L+1$ such that `Datas[l].X` stores $X^l$; `Datas[L].X` is the output of the network.

The whole inference phase can be performed with the following code:

```
for(l=0; l<L; l++){
  for(i=0; i<N; i++)
    for(j=0; j<N; j++){
      block_mult(Layers[l].W[i][j], Datas[l].X[j],
                 Datas[l+1].X[i], m);
    }
  for(i=0; i<N; i++)
    block_bias_act(Layers[l].b[i], Datas[l+1].X[i], m);
}
```

The first loop on `l` goes over all the layers; the loops on `i` and `j` go over all the blocks of $W$, $X$ and $B$. The `block_mult(Layers[l].W[i][j], Datas[l].X[j],` `Datas[l+1].X[i], m)` routine computes $X_{i1}^{l+1} += W_{ij}^{l} \times X_{j1}^{l}$. The `block_bias_act( Layers[l].b[i], Datas[l+1].X[i], m)` routine computes $\sigma\left(X_{i1}^{l+1} + B_{i1}^{l}\right)$.

This exercise is about parallelizing the above code.

## 2 Package content

In the `neural_network` directory you will find the following files:

- `main.c`: this file contains the main program which first calls the `init_data` routine which initializes with random values the `W` , `X` and `b` matrices for all the layers. These matrices are coded as 2D arrays of type `block`; each entry `W[i][j]` is simply a 2D array of size `m * m`. The main program then calls the `sequential_nn` routine executes the above code, the `parallel_nn_loops` and `parallel_nn_tasks` routines which contain the parallelized code to be implemented. After these calls the main program checks that the output of the parallel routines is the same as that of the sequential one. **Only this file has to be modified for this exercise**.
- `aux.c, aux.h`: these two files contain auxiliary routines and **must not be modified**.

The code can be compiled with the `make` command: just type `make` inside the `neural_network` directory; this will generate a `main` program that can be run like this:

```
$ ./main n m L
```

where `n` is the size (number of rows and columns) of the $W$ matrices, `m` the number of columns of the $X$ matrices and `L` the number of layers in the network.

## 3 Assignment

- ⌨ At the beginning, the `parallel_nn_loops` and `parallel_nn_tasks` are a copy of `sequential_nn`. Modify these routines in order to parallelize them. The first **must** be parallelized using the `omp for` directive and the second **must** be parallelized using the `omp task` directive. In the second, the `depend` clause can be used to define dependencies between tasks, where needed. Make sure that the difference between sequential and parallel outputs (i.e., the value printed on screen by the `compare_matrices` routine) is smaller than $10^{-14}$.
- ✎ Report the execution times for the implemented parallel versions with 1, 2 and 4 threads and compare them with the sequential routine. What speedup could you achieve? analyze and comment on your results. Report your answer in the form of comments at the bottom of the `main.c` file.

## Advice

- When using the OpenMP `task` construct, always think about data scoping to make sure input data has the correct value upon execution of the task and returned results do not go out of scope when the task is finished.
- Note that the `depend` clause allows you to specify dependencies due to access to single coefficients of an array. For example:

```
#pragma omp task depend(inout:x[i])
   x[i] += 1;
```

- Choose small values for $n$, $m$ and $L$ when developing (for example, $n = 20,\ m = 5,\ L = 1$) and big ones when evaluating performance (for example, $n = 2000,\ m = 100,\ L = 10$).