Prof. Dr. A.K. Hartmann WiSe 2023/24, 5.04.4521
Blatt 10, 1.03.2024

Exercises Computational Physics

10 Feed-Forward Neuronal Networks

1. Download the programm `feed_forward_fragment.c` from StudIP.
2. Compile, also when you changed the program, with

   ```
   cc -o feed_forward feed_forward_fragment.c -lm -Wall
   ```

3. Investigate the already existing functionen (also the main program) in `feed_forward_fragment.c`.
4. Complete the functions
   - `output_ff_network(int L, double *x, int M, double *y, double **w, double *wt)!`

     It calculates the output of the neural network

     $$y_j = \sigma\left(\sum_{k=0}^{L} w_{jk} x_k\right) \quad (j = 1 \ldots M) \tag{1}$$

     $$z = \sigma\left(\sum_{j=0}^{M} \tilde{w}_j y_j\right) \tag{2}$$

     Keep in mind that $x_0 = 1$ and $y_0 = 1$ have to be assigned.
     Hint: utilize the existing function `output_neuron2()`. (1 P)
   - `void ff_learning(int L, int M, double **w, double *wt, double epsilon, int (*f)(int, double *), int K )`

     This function is supposed to adjust the weights (w_{ij} and $\tilde{w}_j$), by iterating a loop `K` times. Within the loop, each time
     - random vectors $(x_1, \ldots, x_L) \in \{0, 1\}^L$ are generated by calling `random_vector_d()`
     - the net output is obtained by `output_ff_network()`
     - the target output is obtained by calling `f()`
     - the weights are adjusted as explained in the lecture (where first the weights $\tilde{w}_j$ and next the weights w_{jk} are updated) Hint: it *may* help to recalculate the network output after $\tilde{w}_j$ was updated (and obtaining new values for z).

Hint: You have to allocate corresponding vectors locally and free them before the function returns. (4 P)

Test the functions seperately, by initializing the weights within `main()` in a suitable way.

Test your program also with `valgrind`.

5. Perform simulations with the finished program `feed_forward` within the shell by calling `feed_forward <L> <M> <num iterations>`. (There is an optional final parameter for the seed).

   - Consider different number of input and hidden neurons $L = 2, 4, 10$, $M \in \{1, \dots, 2L\}$
   - with target functions `test_majority_d()` and the parity function `test_parity_d()`. (Attention: the function is hard-coded in the programm by the variable `function` and has to be changed accordingly. You can alternatively use more comand line parameters, to avoid multiple compiling.)

     Vary the number of learning iterations (or make a very long run), more that 10^7 is not needed.

     Investigate the error rate as function of the number of (so-far) iterations. This is printed by the main program.

     For which cases do you observe (somehow) convergence to low/zero error? (2 P)
   - Plot (e.g. with `gnuplot`) for $L = 6$ and two values of M ($\approx L$ and $\approx 2L$) for both to be learned functions the error rate $e(t)$ as a function of the number t of iterations. (1 P)

6. Implement the simple "by hand" solution for the parity function (with L input bits and suitable M) and suitable value for the weights.

   Test your function with these fixed couplings. The error rate should be 0%. (2 P)