CPSC/ECE 3520
Spring 2020
SDE #2:
More Hopfield Network Experience Using Prolog
(incl. Code Reading)

Assigned 3/5/2020 (or earlier)
Due 11:59 PM 4/9/2020

Contents

1 The Assignment

SDE2 extends our look at the Hopfield network with a Prolog implementation. After SDE1, you should gain some perspective on the difference between a functional and declarative programming paradigm. I will not reiterate the earlier Hopfield background material used in SDE1, but that material is still relevant.

You will design, implement and test 6 Prolog predicates related to the previous Hopfield simulation. They correspond to functions specified in SDE1. In addition, like SDE1, you will be given a set of 4 relevant predicates for 'code reading'. The given predicates are contained in file:

```
sde2_given-predicates.pro
```

Your implementation of the other 6 predicates is to be in file:

```
sde2_sp2020.pro
```

Note that you must solve this SDE using only Prolog. We have no interest in other solution implementation approaches.

Disclaimer. A script will be used to evaluate your Prolog software solution. If you don't follow the assignment specifications herein, the script will probably fail.

2 Prolog Data Structures for Network

Like SDE1, all computations involving multiple unit outputs or weights will use a list representation. For example, the state of a 2-unit network would be represented as the Prolog list:

```
[-1.0,+1.0]
```

where unit 1 has output $o_1 = -1.0$ and unit 2 has output $o_2 = +1.0$.
For ease in testing (**only**), a way to 'hold' data is shown by designing a 'data' predicate. For example:

```
data(oi,[-1.0,-1.0]).
```

You will see numerous uses of this in the examples.
An example of a Prolog list of lists of floating point numbers (the 'weight matrix') could be the Prolog list (of lists):

```
[[0.0, 1.0, -1.0, -1.0],
[1.0, 0.0, -3.0, 1.0],
[-1.0, -3.0, 0.0, -1.0],
[-1.0, 1.0, -1.0, 0.0]]
```

3 Necessary Prolog Predicates and Prototypes

You will design, implement and test the 6 of the 10 predicates with prototypes given below. The first four predicates are given. Of course, you may also develop auxiliary predicates, which will not be graded directly. The Prolog predicates mirror the functionality of their `ocaml` namesakes.

3.1 Predicates Corresponding to Functions in SDE1 (for reference; predicate arity is indicated with slash notation)

```
/* given */
hop11Activation/3
hop11ActAll/3
netUnit/3
netAll/3
/* for state propagation */
nextState/3
updateN /4
findsEquilibrium/3
energy/3
/* for training */
hopTrainAstate/2
hopTrain/2
```

3.2 Specifying Prototypes

Recall predicate arity is indicated with slash notation, as used in Section 3.1.

```
/* Recall the argument designation from SWI-Prolog, where:

+Argument means the argument is bound before invoking the predicate

-Argument means the argument is bound to a value, if possible,
in the process of finding a solution

?Argument means either (works both ways)
*/
```

Note the arity and argument specification is only used to convey the prototype, **this notation is NOT used in invoking the predicates**. This is shown in the examples.

3.3 Reminder: Hopfield (1,-1) Unit Characteristic

$$o_i = \begin{cases} 1.0 & if \sum_{j;\ j \neq i} w_{ij} o_j > 0.0 \\ o_i \quad (previous\ value) & if \sum_{j;\ j \neq i} w_{ij} o_j = 0.0 \\ -1.0 & otherwise \end{cases} \tag{1}$$

3.4 Reminder: Network Energy

It is paramount to note that network equilibrium states are related to minima of the following energy function:

$$E = -\frac{1}{2} \sum \sum_{i \neq j} w_{ij} o_i o_j = -\frac{1}{2} \underline{o}^T W \underline{o} \tag{2}$$

4 Prototypes of the Given Predicates

```
/* 1. Hopfield activation function for (-1,1) units; this is for a single unit. */

hop11Activation(+Net,+Oldo,-Output)

/* 2. Entire Hopfield network state, given network net activation */

hop11ActAll(+AllNet,+OldState,-NewState)

/* 3. net activation computation for single set of weights (one unit) */

netUnit(+Inputs,+Weights,-Net)

/* 4. net activation computation for entire network */

netAll(+State,+Weights,-NetEntire)
```

5 Prototypes of Predicates You Will Design, Implement and Test

5.1 State Propagation-Related Predicates

```
/* 5. Next state computation */

nextState(+CurrentState,+WeightMatrix,-Next)

/*  6. Update network state N iterations (N=0 is current state) */

updateN(+CurrentState,+WeightMatrix,+N,-ResultState)

/* 7. Succeeds if finds equilibrium, fails otherwise.

findsEquilibrium(+InitialState,+WeightMatrix,+Range)
 */

8. /* determines state energy, i.e., implements Equation (2) */

energy(+State,+WeightMatrix,-E)
```

5.2 Network Storage Prescription Predicates ((-1,1) Units)

```
/* 9. Computes weight matrix for only one (i.e., a single) stored state */

hopTrainAstate(+Astate,-WforState)

/* 10. Computes Hopfield weight matrix for a list of desired stored states */

hopTrain(+ListofStates,-WeightMatrix)
```

6 Examples

You should probably spend some time looking over these examples and check numerical results with your `ocaml` effort. Some of these examples correspond to the given predicates.

6.1 Sample Data for Illustration and Testing

```
/* data for illustration and testing */

allAct([-10.0,5.0,-1.0,2.3,0.0,7.7,-3.5]).

data(oi,[-1.0,-1.0]).

data(we,[[0.0,-1.0],[-1.0,0.0]]). /* for cycle */

/* some 4-D data for simulation/debugging */

data(os1,[1.0, -1.0, 1.0, -1.0]).

data(os2,[-1.0, -1.0, 1.0, -1.0]).

data(os3,[-1.0, -1.0, 1.0, 1.0]).

/* more data for state crowding and other effects */

data(os4,[1.0, 1.0, 1.0, 1.0]).

data(os5,[-1.0, -1.0, -1.0, -1.0]).

data(os6,[1.0, 1.0, -1.0, -1.0]).
data(o1o2o3,[[1.0, -1.0, 1.0, -1.0],[-1.0, -1.0, 1.0, -1.0],[-1.0, -1.0, 1.0, 1.0]]).

data(o1o1o1o2o3,[[1.0, -1.0, 1.0, -1.0],[1.0, -1.0, 1.0, -1.0],[1.0, -1.0, 1.0,
 -1.0],[-1.0, -1.0, 1.0, -1.0],[-1.0, -1.0, 1.0, 1.0]]).

data(w,[[0.0, -1.0, 1.0, -1.0],
        [-1.0, 0.0, -1.0, 1.0],
        [1.0, -1.0, 0.0, -1.0],
        [-1.0, 1.0, -1.0, 0.0]]).
```

6.2 Sample Predicate Use

These are to introduce you to the required Prolog behavior. You should not need as many examples, since you have completed SDE1 and can generate all you want.

```
/* Consult source files
   sde2_given_predicates.pro and (your) sde2_sp2020.pro */

?- ['sde2_given_predicates.pro', 'sde2_sp2020.pro'].
true.

?- hop11Activation(-1,-1,What).
What = -1.

?- hop11Activation(-0.1,-1,What).
What = -1.

?- hop11Activation(0.1,-1,What).
What = 1.

?- hop11Activation(0.1,1,What).
What = 1.

?- hop11Activation(0.0,1,What).
false.

?- hop11Activation(0.0,1,What).
What = 1.

?- hop11Activation(0.0,-1,What).
What = -1.
```

```
?- netUnit([-1.0, -1.0],[0.0, -1.0],NU).
NU = 1.0.

?- hopTrainAstate([-1.0,1.0,-1.0,1.0],WM).
WM = [[0.0, -1.0, 1.0, -1.0], [-1.0, 0.0, -1.0, 1.0], [1.0, -1.0, 0.0, -1.0],
[-1.0, 1.0, -1.0, 0.0]].

?- hopTrain([[-1.0,-1.0]],W1).
W1 = [[0.0, 1.0], [1.0, 0.0]].

?- hopTrain([[-1.0,1.0]],W1).
W1 = [[0.0, -1.0], [-1.0, 0.0]].

?- data(o1o2o3,Show).
Show = [[1.0, -1.0, 1.0, -1.0], [-1.0, -1.0, 1.0, -1.0], [-1.0, -1.0, 1.0, 1.0]].

?- data(o1o2o3,H),hopTrain(H,W3).
H = [[1.0, -1.0, 1.0, -1.0], [-1.0, -1.0, 1.0, -1.0], [-1.0, -1.0, 1.0, 1.0]],
W3 = [[0.0, 1.0, -1.0, -1.0], [1.0, 0.0, -3.0, 1.0], [-1.0, -3.0, 0.0, -1.0],
[-1.0, 1.0, -1.0, 0.0]].

?- data(o1o1o1o2o3,H2),hopTrain(H2,W4),data(os2,O2),nextState(O2,W4,NS).
H2 = [[1.0, -1.0, 1.0, -1.0], [1.0, -1.0, 1.0, -1.0], [1.0, -1.0, 1.0, -1.0],
[-1.0, -1.0, 1.0, -1.0], [-1.0, -1.0, 1.0, 1.0]],
W4 = [[0.0, -1.0, 1.0, -3.0], [-1.0, 0.0, -5.0, 3.0], [1.0, -5.0, 0.0, -3.0],
[-3.0, 3.0, -3.0, 0.0]],
O2 = [-1.0, -1.0, 1.0, -1.0],
NS = [1, -1, 1, -1].

/* test updateN */

?- data(o1o2o3,H2),hopTrain(H2,W2),data(os1,O1),updateN(O1,W2,0,Res).
H2 = [[1.0, -1.0, 1.0, -1.0], [-1.0, -1.0, 1.0, -1.0], [-1.0, -1.0, 1.0, 1.0]],
W2 = [[0.0, 1.0, -1.0, -1.0], [1.0, 0.0, -3.0, 1.0], [-1.0, -3.0, 0.0, -1.0],
[-1.0, 1.0, -1.0, 0.0]],
O1 = Res, Res = [1.0, -1.0, 1.0, -1.0].

?- data(o1o2o3,H2),hopTrain(H2,W2),data(os1,O1),updateN(O1,W2,1,Res).
H2 = [[1.0, -1.0, 1.0, -1.0], [-1.0, -1.0, 1.0, -1.0], [-1.0, -1.0, 1.0, 1.0]],
W2 = [[0.0, 1.0, -1.0, -1.0], [1.0, 0.0, -3.0, 1.0], [-1.0, -3.0, 0.0, -1.0],
[-1.0, 1.0, -1.0, 0.0]],
O1 = [1.0, -1.0, 1.0, -1.0],
Res = [-1, -1, 1, -1].

?- data(o1o2o3,H2),hopTrain(H2,W2),data(os1,O1),updateN(O1,W2,2,Res).
```

```
H2 = [[1.0, -1.0, 1.0, -1.0], [-1.0, -1.0, 1.0, -1.0], [-1.0, -1.0, 1.0, 1.0]],
W2 = [[0.0, 1.0, -1.0, -1.0], [1.0, 0.0, -3.0, 1.0], [-1.0, -3.0, 0.0, -1.0],
[-1.0, 1.0, -1.0, 0.0]],
O1 = [1.0, -1.0, 1.0, -1.0],
Res = [-1, -1, 1, -1].

?- data(o1o2o3,H2),hopTrain(H2,W2),data(os3,O3),updateN(O3,W2,1,Res).
H2 = [[1.0, -1.0, 1.0, -1.0], [-1.0, -1.0, 1.0, -1.0], [-1.0, -1.0, 1.0, 1.0]],
W2 = [[0.0, 1.0, -1.0, -1.0], [1.0, 0.0, -3.0, 1.0], [-1.0, -3.0, 0.0, -1.0],
[-1.0, 1.0, -1.0, 0.0]],
O3 = [-1.0, -1.0, 1.0, 1.0],
Res = [-1, -1, 1, -1].

?- data(oi,Oi),data(we,W),updateN(Oi,W,1,Next).
Oi = [-1.0, -1.0],
W = [[0.0, -1.0], [-1.0, 0.0]],
Next = [1, 1].

?- data(oi,Oi),data(we,W),updateN(Oi,W,2,Next).
Oi = [-1.0, -1.0],
W = [[0.0, -1.0], [-1.0, 0.0]],
Next = [-1, -1].

?- data(oi,Oi),data(we,W),updateN(Oi,W,3,Next).
Oi = [-1.0, -1.0],
W = [[0.0, -1.0], [-1.0, 0.0]],
Next = [1, 1].

?- data(oi,Oi),data(we,W),updateN(Oi,W,4,Next).
Oi = [-1.0, -1.0],
W = [[0.0, -1.0], [-1.0, 0.0]],
Next = [-1, -1].

\* test findsEquilibrium */

?- findsEquilibrium([-1.0,-1.0],[[0.0, -1.0], [-1.0, 0.0]],5).
false.

?- data(w,W).
W = [[0.0, -1.0, 1.0, -1.0], [-1.0, 0.0, -1.0, 1.0], [1.0, -1.0, 0.0, -1.0],
[-1.0, 1.0, -1.0, 0.0]].

?- data(os1,A).
A = [1.0, -1.0, 1.0, -1.0].
```

```
?- data(os1,A),data(w,W),findsEquilibrium(A,W,3).
A = [1.0, -1.0, 1.0, -1.0],
W = [[0.0, -1.0, 1.0, -1.0], [-1.0, 0.0, -1.0, 1.0], [1.0, -1.0, 0.0, -1.0],
[-1.0, 1.0, -1.0, 0.0]].

/* test energy */

?- data(os1,A),data(w,W),energy(A,W,E).
A = [1.0, -1.0, 1.0, -1.0],
W = [[0.0, -1.0, 1.0, -1.0], [-1.0, 0.0, -1.0, 1.0], [1.0, -1.0, 0.0, -1.0],
[-1.0, 1.0, -1.0, 0.0]],
E = -6.0.
```

Use your `ocaml` implementation to check these numerical results.

7 Pragmatics and Constraints

Use this document as a checklist to be sure you have responded to all parts of the SDE3 assignment, and have included the required files (readme.txt, sde3.pro and sde3.log).

- The simulation uses `only Prolog code` you wrote (excluding the given predicates).
- **No use of Prolog's 'if..then' construct** (written as `->/2`):

  ```
  :Condition -> :Action  construct

  (A -> B ; C). /* this is if-then else */
  ```

- No use of `assert` or `retract` predicates

8 Deliverables

The final, single zipped archive which must be submitted (to Blackboard) by the deadline must be named `<yourname>`**-sde2.zip**, where `<yourname>` is your (CU) assigned user name.

The contents of this archive are:

1. A `readme.txt` file listing the contents of the archive and a very brief description of each file. Include 'the pledge' here. Here's the pledge:

   > **Pledge:**
   > On my honor I have neither given nor received aid on this exam
   > SIGN ____

2. `sde2_sp2020.pro`: The single file containing the 6 predicates (and any supporting predicates, if developed and used). **Do not include the 4 given predicates; we will supply those.**

3. `sde2.log`: A file containing 2 sample uses of each of your predicates, **different from the examples shown herein.**

We will attempt to consult your file and test each of these Prolog predicates via a series of goals. As before, most of the grade will be based upon this evaluation.

9 Deadlines Matter

This remark was included in the course syllabus and SDE #1. Since multiple submissions to Canvas are allowed[1], if you have not completed all functions, you should submit a freestanding archive of your current success before the deadline. This will allow the possibility of partial credit. **Do not attach any late submissions to email and send them to either me or the graders.**

[1]But we will only download and grade the latest (or most recent) one, and it must be submitted by the deadline.