Neural Modeling and Computational Neuroscience

Claudio Gallicchio, Ph.D.

Computational Intelligence & Machine Learning
http://www.di.unipi.it/groups/ciml

Dipartimento di Informatica
Università di Pisa - Italy

Additional Info

1) Using state-space models for next-step time-series prediction

2) Hold out cross validation

LSMs for solving a learning problem

- load the dataset
  - `load laser_dataset.mat`
  - `data = cell2mat(laserTargets)`
- split input & target output
  - `input = data(1:end-1)`
  - `target = data(2:end)`
- divide the data in training set and test set
  - `tr_input = input(1:1500)`
  - `ts_input = input(1500+1:2000)`
  - `tr_target = target(1:1500)`
  - `ts_target = target(1500+1:2000)`

LSMs for solving a learning problem

- Run the liquid using the input of the training set
  - you will need to change this line:
    ```
    I=[5*randn(Ne,1);2*randn(Ni,1)]; % thalamic input
    ```
    into something like this:
    ```
    I=[scaling_p1 * tr_input(t) * ones(Ne,1); scaling_p2 * tr_input(t)
    *ones(Ni,1)];
    ```
  - where scaling_p1 and scaling_p2 are two hyper-parameters

LSMs for solving a learning problem

- Collect the activations of the liquid for the training samples, i.e.:
  - put the value of the membrane potential of every neuron in the liquid into a column vector, e.g. `state(t)`
    - You can try other choices for the neuronal encoding (e.g. firing rates, or moving averages over the spiking activations)
  - concatenate these columns into a matrix, e.g. `trainStates`
  - analogously, put the target for the training samples into a matrix (a row vector, in this case), e.g. `trainTargets = tr_target`

LSMs for solving a learning problem

- Solve the LMS problem in closed form using, e.g., pseudo-inversion
  - `Wout = trainTargets * pinv(trainStates)`
  - this makes tr_output = Wout * trainStates approximate the trainTargets vector in the LMS sense
- You can now compute the MAE on the training set
  - compute the output: `tr_output = Wout * trainStates`
  - compare with the target: `tr_error = mean(abs(tr_output - trainTargets))`

LSMs for solving a learning problem

- Compute the output for the test set
  - run the liquid on the test set time-steps
    now you will have to use `ts_input` instead of `tr_input`
  - as done for the training set, collect the states of the test samples into a matrix, e.g. `testStates`
  - compute the output on the test set:
    `ts_output = Wout * testStates`
- Compute the MAE on the test set
  - `ts_error = mean(abs(ts_output - ts_target))`

Tuning the values of the hyper-parameters

- The performance of your model depends on the values of some crucial hyper-parameters, e.g., the number of neurons, the scaling of the input, etc.
- The values of these hyper-parameters should be determined by *model selection*, **without looking at the test set**

Hold out cross validation

- Put the test set aside for now
- Split the training samples into a training set and a validation set (e.g., 70%-30%)
- Train on the training set and evaluate the performance on the validation set for your network, using multiple values of the hyper-parameters
- Choose the set of hyp.-par. values that minimize the MAE on the validation set

Refit

With the chosen values of the hyper-parameters:

- train on the original (complete) training set
- evaluate on the test set