Principles of Brain Computation KU

708.086 18S

Homework Sheet 5

Problems marked with * are optional.

Liquid State Machines [6+2*P]

In this task, we use a liquid state machine (LSM) to solve simple tasks. The LSM is implemented as a recurrent network of spiking neurons using NEST.

Model details

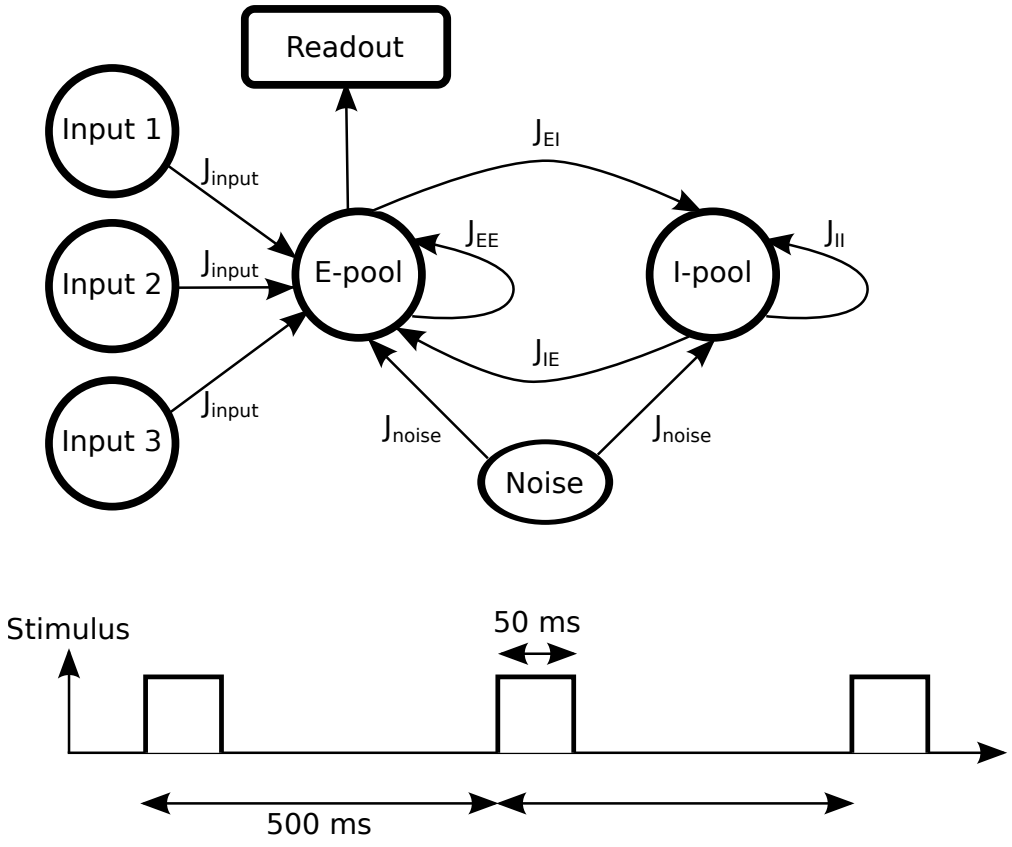

Figure 1: LSM task illustration. Top: network model. Bottom: schematic illustration of stimulus presentation times.

We will use a recurrent neural network as a liquid filter (Figure 1). The network consists of a pool of excitatory ($N_E = 1000$) and a pool of inhibitory neurons ($N_I = 250$). All neurons are modeled as LIF neurons with exponential-shaped PSCs with $C_\mathrm{m} = 30\,\mathrm{pF}$, $\tau_\mathrm{m} = 30\,\mathrm{ms}$, $V_\mathrm{rest} = 0\,\mathrm{mV}$, $V_\mathrm{threshold} = 15\,\mathrm{mV}$, and $V_\mathrm{reset} = 13.8\,\mathrm{mV}$. The decay time constants for excitatory and inhibitory currents should be 3 ms and 2 ms, respectively. Each neuron should furthermore receive a bias current of $I_e = 14.5\,\mathrm{pA}$.

All neurons should receive random Poisson spikes as input with a rate of 25 Hz (weight: $J_\mathrm{noise} = 5\,\mathrm{pA}$).

The excitatory pool and the inhibitory pool should be interconnected (mean weights $J_\mathrm{EI} = 250\,\mathrm{pA}$, $J_\mathrm{IE} = -200\,\mathrm{pA}$). Furthermore, each pool should have recurrent connections (mean weights $J_\mathrm{EE} = 50\,\mathrm{pA}$, $J_\mathrm{II} = -200\,\mathrm{pA}$). These connections have short-term plasticity (the parameters are given in the code template). The number incoming excitatory synapses per neuron should be $C_E = 2$, the number of incoming inhibitory synapses should be $C_I = 1$.

Furthermore, the excitatory neurons receive spiking input from a number of input spike generators ($J_\mathrm{input} = 50\,\mathrm{pA}$). Each spike generator should project to $C_\mathrm{input} = 100$ excitatory neurons.

All connections should have delays drawn from a Gaussian with $\mu = 10\,\mathrm{ms}$ and $\sigma = 20\,\mathrm{ms}$ which is clipped to the range $[3\,\mathrm{ms},\ 200\,\mathrm{ms}]$. The weights connecting noise input to the LIF neurons should have a constant value (J_noise). The input weights should be uniformly distributed in $[0.5 \cdot J_\mathrm{input},\ 1.5 \cdot J_\mathrm{input}]$. All other weights should be drawn from Gaussian distributions with a mean of J and a standard deviation of $0.7 \cdot J$ for a given mean weight of J.

When set up correctly, the mean excitatory and inhibitory firing rates should lie around 10 and 25 Hz, respectively.

Stimulus presentation

The LSM receives input from three inputs (Figure 1). One stimulus is presented every 500 ms, each stimulus lasts 50 ms. For each individual stimulus, each input neuron fires either at a high firing rate (200 Hz), or it remains silent. Thus, 3 bits x_0, x_1, x_2 with $x_j \in \{0, 1\}\ \forall j$ are given to the network per stimulus. Using the response of the recurrent network, we perform simple computations on these inputs by computing some $y = f(x_0, x_1, x_2)$.

Learning tasks

Using the spikes from 500 randomly chosen excitatory neurons, the liquid states can be computed at any time by filtering the spikes with an exponential. After defining targets (depending on the task), we can train linear readouts on these states to perform computations.

We will consider the following tasks, which are passed to the `experiment()` function as a parameter:

- `sum`: We extract the liquid states 20 ms after stimulus presentation has ended. The goal is to learn the sum of the three input bits ($y = x_0 + x_1 + x_2$).

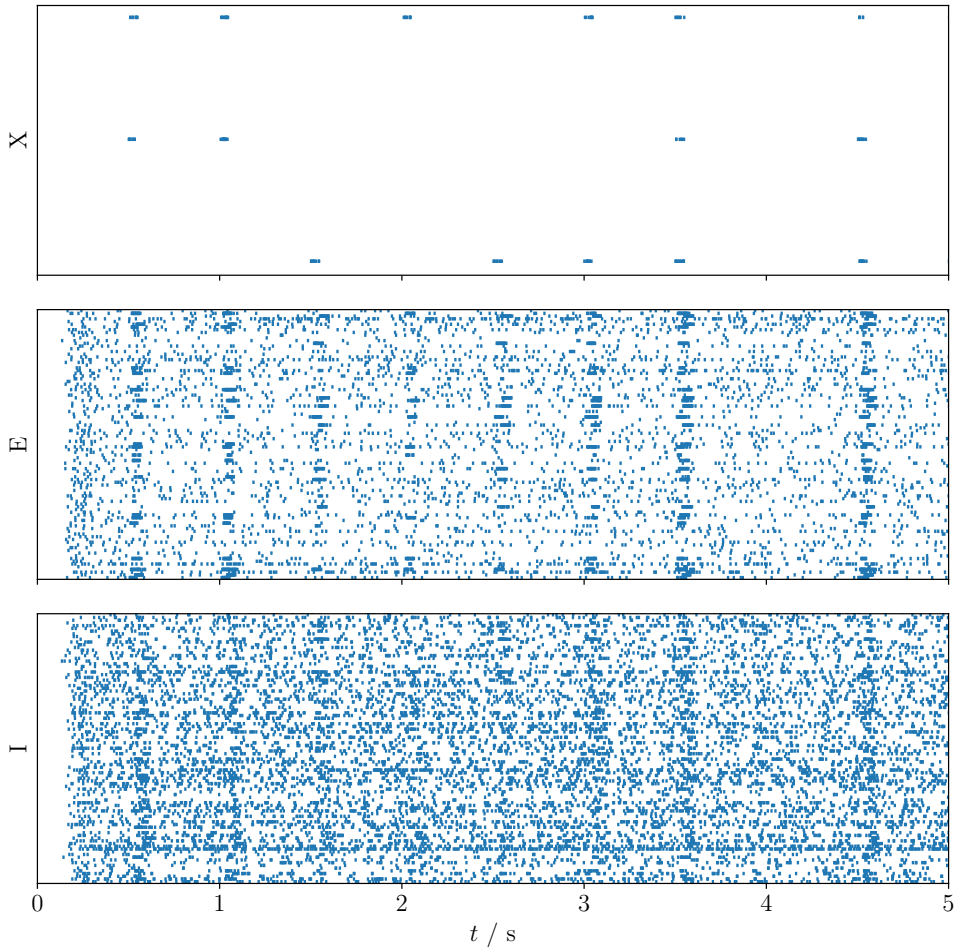

Figure 2: example network activity of the LSM. Only 100 neurons from the excitatory and the inhibitory pool are shown.

- `xor`: Again, liquid states are extracted 20 ms after the end of each stimulus. The targets in this case should be the XOR of the first two input bits ($y = \mathrm{XOR}(x_0, x_1)$). The third input bit is not releveant for this computation and can be viewed as a distraction.
- `mem1`: We assess it the network has the fading memory property and determine how long the information about the stimulus is present in the spiking activity. To do so, we extract the liquid states with varying delay after each stimulus. For `mem1`, we see how long the first input bit x_0 can be decoded from the network ($y = x_0$).
- `memall`: In a similar fashion, we assess how long we can extract all three input bits simultaneously by mapping each possible input bit combination to an integer.

The code provided for training readouts assume that the given targets are integers $\{0, 1, ..., N_{\mathrm{targets}}\}$.

Code

The provided code template implements most of the functionality required to run the experiments. The inputs are generated by the function `generate_stimulus` which returns both the actual input bits for each stimulus presentation as well as spike times for spike generators which encode this information.

You need to fill in parts at all locations indicated by `TODO` comments. In particular, you need to

- create all neurons, input spike generators, spike detectors, and the noise generator,
- set the spike times of the spike generators as given by the `generate_stimulus` function,
- compute the mean excitatory and inhibitory firing rate after the simulation,
- plot the network activity within the first 5 s to check the network dynamics,
- define the target vectors for the different tasks (described above), and
- implement the training procedure for the memory tasks `mem1` and `memall`.

The provided code trains linear readouts given some target vector. The targets should be integers $\{0, 1, ...\}$. During training, a regularization factor can be used (`reg_fact` parameter of the `experiment` function).

One useful function implemented in the code template is the saving of all recurrent spikes after the simulation (output file `data.pkl`). Since we can perform different tasks using the same liquid states, you only need to run the simulation (which may take quite long) once and reload the input data and the network spikes by running `experiment()` with `simulate=False`.

Note: the code uses the Python libraries `joblib` and `sklearn` (which have to be installed with pip: run `pip3 install --user --upgrade joblib sklearn` in the console).

Task 5a [2P]

Run the simulation to generate input data and network spikes. Train a linear readout for the `sum` task. How well does the network perform?

Task 5b [2P]

Next, train a linear readout to perform the `xor` task as specified above. You do not need to re-run the simulation. You well does the network perform? Use a regularization factor > 0 to improve the performance.

Task 5c [2P]

Next, we check the fading memory property and assess how long we can extract information about the stimulus identity after the stimulus presentation has ended. Define a mapping of input bits to targets for the `memall` task and state it in your report. Train readouts for the `mem1` and `memall` tasks and plot the results. Does the network posses the fading memory property? Explain any occuring difference between the time course of the test error on the two tasks.

Task 5d [2*P]

The network parameters are not ideal. Tune the connection strengths, connectivity parameters, and/or synaptic delays so the identity of the previous stimulus can be extracted for a longer time span.

Submit the code until 8:00 AM of the day of submission to `mueller@igi.tugraz.at` and `lydia.lindner@student.tugraz.at`. Use `PoBC HW5,` ⟨`name team member 1`⟩ ⟨`name team member 2`⟩ as email subject. Only one email per team is necessary. Submit regular Python code files (`*.py`). You need to hand in a printed version of your report at the submission session. Each team member needs to write their own report. Use the cover sheet provided on the course website.