Autoencoder - Learning to represent signals spike by spike

Yu-Guan Hsieh, Martin Ruffel

June 4, 2017

Recap of our work

We worked on the spiking autoencoder. We started from a standard spiking network trying to represent an input signal. Then, we added recurrent weights to simulate the substraction of the estimate, and to force neurons' potentials to encode the signal reconstruction error.

We rederived the main equations and learning rules of the network, starting from the potential equations, and the loss function $L = ||x-\hat{x}||^2+\nu\sum r_i+\mu\sum r_i^2$ which penalizes high firing rates. We followed the same path as in the article's appendix.

- To simulate the reconstruction error, we want to have $\Omega = FD$
- Then, the potentials V represent approximately the projection $F(x-\hat{x})$ of the error.
- We want neuron i to spike if and only if its spike reduces the loss function L. Putting this in equations, we obtain an inequation that can be rewritten as a threshold condition. It corresponds directly to a LIF neuron's spiking condition, if we identify terms as membrane threshold and potential. This identifiaction allows us to obtain the expression of the optimal decoder $D = F^\top$.
- Since V is proportionnal to reconstruction error, we want to minimize V. Therefore, we derive a simple learning rule for Ω :

  $$\Delta\Omega_{i,j} = -V_i - \Omega_{i,j} \text{ when neuron j spikes}$$

  Indeed, this corresponds to reset potential of all the neurons at each spike.
- Adding the L2 penalization term changes this learning rule because it adds term $-\mu r$ in potential V:

  $$\Delta\Omega_{i,j} = -V_i - \mu r_i - \Omega_{i,j} - \mu\delta_{i,j}$$
- If we suppose that F is updated a lot slower than Ω, then some equations give us a similar learning rule for F :

$\Delta F_{i,j} = (x - \hat{x}) - F$, what is approximated by $\Delta F_{i,j} = \alpha x - F$ when neuron i spikes

With these learning rules, we have implemented in Python the spiking autoencoder, using the article's Appendix pseudo-code. But when we launched some tests to see if these equations work, we obtained a lot of problems.

- There were some little errors in pseudo-code, or unexplained modifications.
- The obtained network was very unstable with the given parameters. Indeed, after some training steps, only a few neurons were spiking almost at each time step, and the others did nothing. We spent a lot of time trying to solve this, and modifying constants like λ, μ, definition of c to obtain coherent results. After two weeks, we finally found some nice parameters.
- With these new parameters, the weights F don't change a lot, and therefore tuning curves don't move through training to cover the whole input space.
- Reconstruction error doesn't seem to decrease through training.
- It is very hard to adjust parameters of input and network to obtain such nice reconstructions as in the main paper, especially to choose the input parameters. Indeed, we didn't understand well what was the role of c instead of x, and which parameters we should choose to obtain coherent x and c. Because the terms in learning rules imply to fix c scale, it is hard to deal with modifications of x without readjusting other network parameters.

The only satisfying results we obtained were that the firing rates decrease during the training : there are divided by at least 4.

Since we didn't get nice results, we couldn't compare the outputs of the network to independent Poisson processes, and we didn't try to feed the network with correlated inputs.