UNIVERSITY OF COPENHAGEN

Volr: A declarative interface language for neural computation

Experimental neural systems modeling with Jupyter Notebooks

Jens Egholm Pedersen
Department of Computer Science
University of Copenhagen

Christian Pehle
Kirchoff-Institute for Physics
University of Heidelberg

Contact Information:
Universitetsparken 5
2100 København Ø

+45 25122752
xtp778@alumni.ku.dk

Abstract

For researchers with little to no programming experience, NEST can be intimidating. Despite the efforts done by the NEST community to adopt interface libraries like PyNN, pre-and post-processing remains time-consuming and complicated. By drawing on concise abstractions and tools such as connection-set algebra and Jupyter notebooks, a holistic execution environment has the potential to lower the barrier to entry and drastically increase the adoption of NEST. Developed in collaboration with cognitive neuroscientists, a domain-specific language and a Jupyter notebook plugin is presented. It establishes a single execution environment, to act as an intermediate layer between the researcher and the simulation. The environment builds on the efforts of PyNN and has been demonstrated to work with both NEST and BrainScaleS. Future plans are to integrate high level abstractions for learning in the domain-specific language, making use of the learning-to-learn paradigm.

Introduction

Despite the pioneering work of simulator APIs for neural experiments, there are still significant barriers to entry for inexperienced users: learning an interface language like PyNN (including comprehending the theoretical counterparts), digitising analog experiments and performing complicated pre- and post-processing of large amounts of data.

The showcased project presents a prototype of a development environment for experimenting with neural simulations. The domain-specific language (DSL) Volr offers a concise method for specifying experiments, while the execution environment around Volr integrates seamlessly with Jupyter Notebooks and several execution targets like NEST. The project makes working with neural simulations simpler and more efficient, thereby increasing the research output and widening the audience for neural simulators.

The prototype is designed to facilitate the entire life cycle of an experiment: generation/preprocessing of data, neural network topology and properties, experimental setup and finally data extraction and post-processing (see figure 1).

Documentation is available at: **`https://volr.readthedocs.io`**

Project objectives

1. Integrate with evaluation platforms for simulated, neuromorphic and artificial neural networks.
2. Support fast feedback loops between experimental ideas and results.
3. Develop a concise and formal declarative language to reproduce and reason about experiments.
4. Explore the domain space for cognitive experiments including learning and online plasticity.

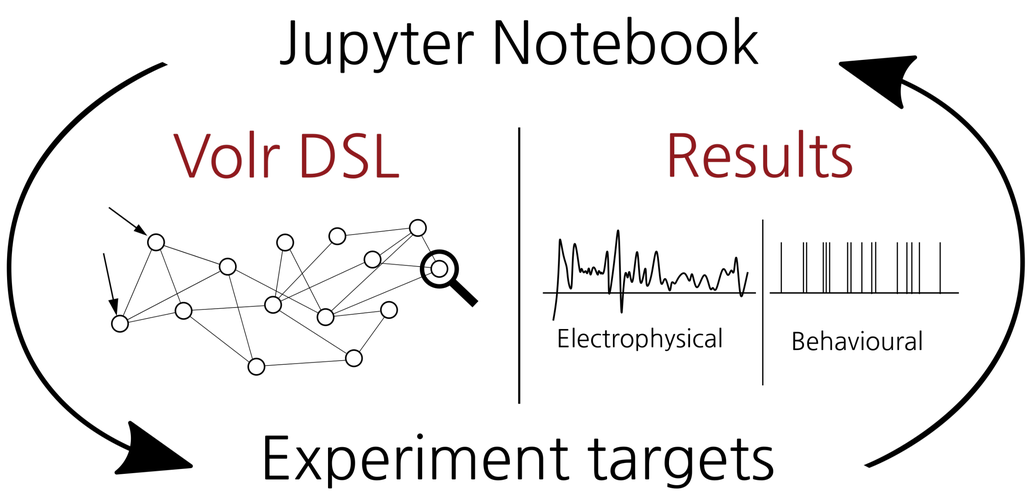

Figure 1: The workflow of an entire experiment, from the modeling in a Jupyter notebook, through the evaluation on one or more platforms, to the retrieving of the experiment results for subsequent analysis. From the perspective of the user everything happens directly in the notebook.

1. Execution targets

The current prototype integrates with three simulation targets: NEST (through the NEST API), BrainScaleS for spiking neural networks (SNN) (through PyNN) and artificial neural networks (ANN) with GPU acceleration (through Futhark [5]).

The DSL maintains an intermediate representation of the experiments, including all the necessary details to reproduce the setup. Generated specifications exist for all execution targets to correctly map the experiment to running code. This lets Volr expose constraints directly in the language to provide conveniences like compile-time model verification.

Finally, the intermediate representation simplifies the integration with execution targets, currently by through the efforts of Futhark, PyNN and the NEST API (see figure 2). However other targets would be interesting to examine, such as SpiNNaker, TensorFlow, Intel's Loihi chip and the next generation BrainScaleS 2 system from Heidelberg.

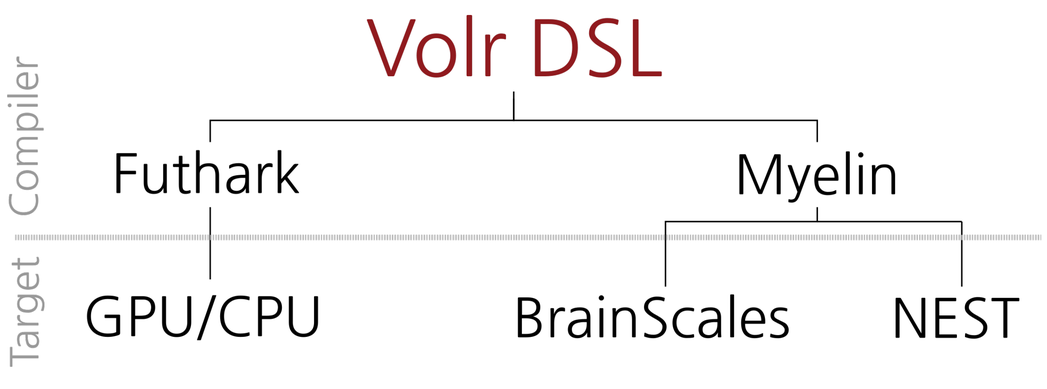

Figure 2: The translation from the DSL to the execution targets. ANN models are interpreted by Futhark which in turn is evaluated on GPU/CPUs. SNN models are interpreted by the internal Myelin compiler that checks for inconsistencies and generates target-specific code through PyNN and the NEST API.

2. Fast experiment feedback loop with Jupyter Notebooks

Jupyter Notebooks are increasingly popular in data processing and data analysis domains. The interactive and iterative approach allows researchers to quickly map hypotheses to experiments, with the support of the vast Python ecosystem.

After installing the environment it takes less than a minute to setup an experiment and extract the first results. Figure 3 illustrates how data can be injected and extracted. By building on open technologies like Python and Jupyter Notebooks, the experiments are straight-forward to share and expand upon.

a) Generating experiment input

```
in_stim = ... # Spikes or data

stimulus s
  input: $in_stim
```

b) Analysing experiment output

```
# Plot experiment data
result.spikes.plot()
# Plot target specific data
result.nest.voltages.plot()
```

Figure 3: Examples for a) injecting experimental data and b) extracting plots with Python.

3. DSL specification and design

At the core of project is the domain-specific language (DSL) Volr, that models neural network topology as well as simulator-specific properties. The DSL is built in Haskell and seeks to form the basis for a tailor-made modeling language for the cognitive and theoretical neuroscience [6].

The inner representational structure allows for abstract analyses, by understanding networks as first-class citizens subject to function composition and lambda calculus. These *algebraic* properties could pave the way for the application of techniques such as reverse differentiation, dependent types and semantic error checking.

The syntax of the DSL is showcased in figure 4. It supports detailed descriptions of population and neuron properties and employs connection-set algebra to model connectivity [3].

```
stimulus s
  input: "data.json"
population p
  from s: one to one
  neurons: 20
response
  from p: all to one
target NEST
  model: izhikevich
target BrainScaleS
  model: if_cond_exp
```

Figure 4: A simple experimental setup using the Izhikevich neuron model on NEST and an integrate-and-fire model on BrainScales.

4. Future work

In collaboration with the Unit for Cognitive Neuroscience at Copenhagen University the current prototype is employed to study computational models of cognitive rehabilitation theories. Volr gives cognitive neuroscientists a larger degree of independence, while letting them construct complex experiments without the need for intimate knowledge of the execution targets.

Learning-to-learn is a technique for training a SNN with a similar ANN using gradient analysis. Because of the easy translation between artificial and spiking models in Volr, the learning-to-learn framework would be ideal to explore further [2].

Lastly the algebraic properties of the internal model deserve to be explored. Techniques such as automatic differentiation could permit the fast generation of large and complex SNN [1].

Conclusions

1. Prototypical integration with Jupyter Notebooks has been shown to allow fast iterations of experiments, with immediate access to already familiar Python tools for large-scale data analysis.
2. The formalised domain-specific language, Volr, has been developed to describe reproducible and consistent neural network experiments for artificial and spiking substrates. It exploits algebraic techniques such as connection-set algebra [3] and network-function compositions.
3. Three targets are currently supported: NEST, BrainScales and artificial neural networks through Futhark. Further work is needed to expand existing targets and include platforms such as Intel's Loihi, SpiNNaker and the new DLS chip from Heidelberg, Germany.
4. Ongoing work is exploring the domain of more complex cognitive experiments, including the possibility for *learning-to-learn* integration between ANN and SNN, cognitive tasks modeling and reverse differentiation for ANN.

References

[1] A. Baydin, B. Pearlmutter, and A. Radul. Automatic differentiation in machine learning: a survey. *CoRR*, abs/1502.05767, 2015.

[2] G. Bellec, D. Salaj, A. Subramoney, R. A. Legenstein, and W. Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. *CoRR*, abs/1803.09574, 2018.

[3] M. Djurfeldt. The connection-set algebra—a novel formalism for the representation of connectivity structure in neuronal network models. *Neuroinformatics*, 10(3):287–304, Jul 2012.

[4] J. Eppler. A Python interface to NEST. *The Neuromorphic Engineer*, 2009.

[5] T. Henriksen, N. Serup, M. Elsman, F. Henglein, and C. Oancea. Futhark: purely functional GPU-programming with nested parallelism and in-place array updates. *Proceedings of the 38th ACM SIGPLAN*, pages 556–571, 2017.

[6] M. Innes, D. Barber, T. Besard, J. Bradbury, V. Vhuravy, S. Danisch, A. Edelman, S. Karpinski, J. Malmaud, J. Revels, V. Shash, P. Stenetorp, and D. Yuret. On machine learning and programming languages. `https://julialang.org/blog/2017/12/ml&pl`, 2017.

Acknowledgements

We are grateful for the valuable time and counsel of Sebastian Schmitt and Karlheinz Meier from the Kirchoff-Institute for Physics at the University of Heidelberg, Germany.

We would like to thank Jesper Mogensen and Nicolai Daugaard from the Unit for Cognitive Neuroscience and Martin Elsman from the Department of Computer Science at the University of Copenhagen for their interest and important contributions to the project.