Integrating programmable plasticity in experiment descriptions for analog neuromorphic hardware

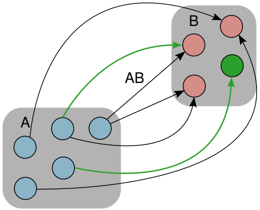
Philipp Spilger[1,2,*], Eric Müller[1], Johannes Schemmel[2]

2025-03-25

[*] pspilger@kip.uni-heidelberg.de
[1] Kirchhoff-Institute for Physics, Heidelberg University.
[2] Institute of Computer Engineering, Heidelberg University.

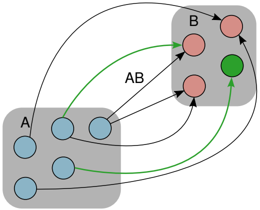
Co-funded by the European Union
EBRAINS

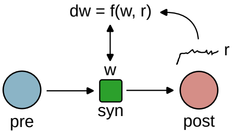
Programmable plasticity



- **plasticity**: altering topology or parameterization during experiment runtime according to decisions made on real time observations

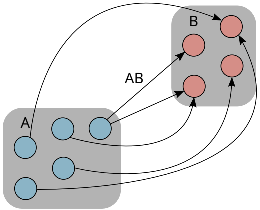
Programmable plasticity





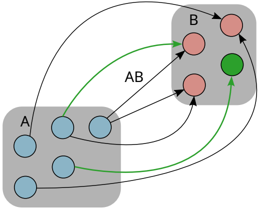
- **plasticity**: altering topology or parameterization during experiment runtime according to decisions made on real time observations
- **programmable**: "freely"-configurable algorithm and execution schedule

Using mixed-signal neuromorphic hardware



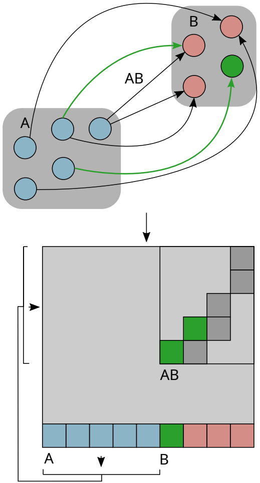
- topology description
    - populations of neurons, projections of synapses
- pre-defined experiment protocol

Using mixed-signal neuromorphic hardware



- topology description
  - populations of neurons, projections of synapses
  - **+ plasticity**
- pre-defined experiment protocol

Using mixed-signal neuromorphic hardware



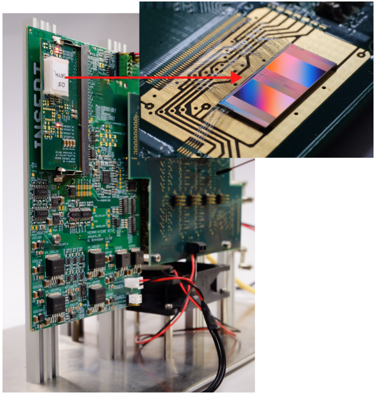
- topology description
  - populations of neurons, projections of synapses
  - **+ plasticity**
  - unplaced!
- pre-defined experiment protocol
- automated translation/mapping to/from hardware
  - placement
  - calibration
  - routing
  - data transform

BrainScaleS-2 neuromorphic system

- developed in Heidelberg
- analog AdEx neurons and COBA/CUBA synapses
- 1000 x network dynamic speed-up

HW: Pehle et al.

SW: Müller et al.

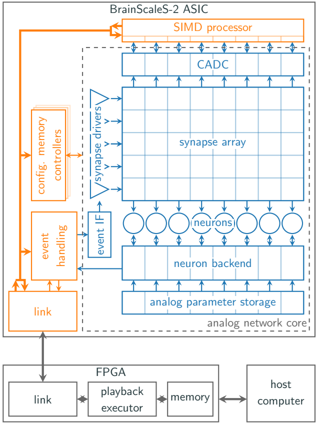


BrainScaleS-2 neuromorphic system

- 512 neurons with 256 synapses each
- two embedded SIMD plasticity processors
- layered software, multiple front ends

HW: Pehle et al.

SW: Müller et al.



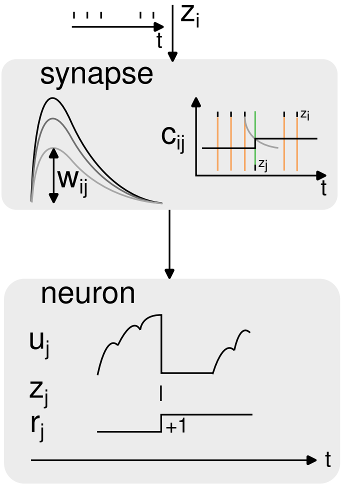
Plasticity on BrainScaleS-2: Embedded processor observables

controllables:

- synaptic weight (6 bit)
- neuron parameterization
- ...

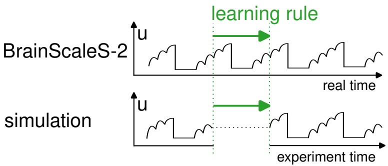
observables:

- parallel ADC recording (8 bit)
  - synaptic correlation (causal, acausal)
  - neuron membrane/adaptation/synaptic inputs
- firing rate per neuron (spike counter, 1+8 bit)



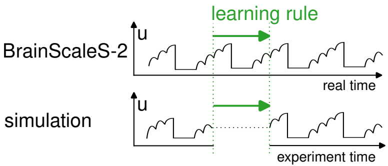
Plasticity on BrainScaleS-2: Restrictions/Limitations

- learning rule execution duration limited: concurrent time-continuous evolution of network dynamics

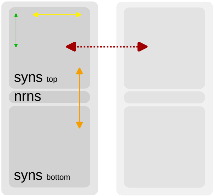


Plasticity on BrainScaleS-2: Restrictions/Limitations

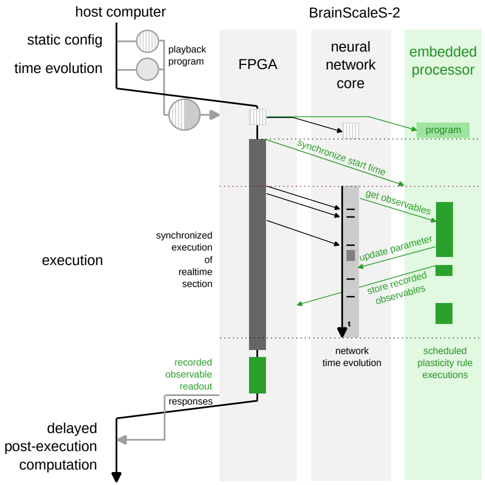
- learning rule execution duration limited: concurrent time-continuous evolution of network dynamics



- physical locality
- available memory
- calculation accuracy (no hardware float)

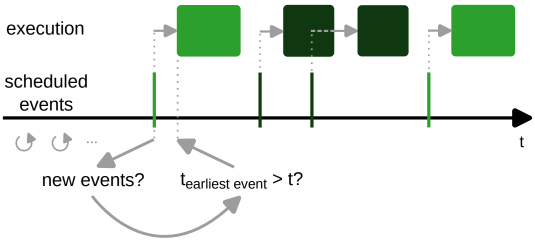


Plasticity abstraction: execution model



- JIT-compiled program for embedded processors
- synchronized execution of neural network evolution and timed plasticity on processors
- scheduling of potentially multiple (sequential or alternating) plasticity rules

Plasticity abstraction: execution model



earliest-deadline-first scheduler

- fixed latency from passed deadline to rule execution
- low computational overhead
- supports dynamic rule execution schedule

Plasticity abstraction: user interface in PyNN

```
import pynn_brainscales.brainscales2 as pynn

class MyPlasticity(pynn.PlasticityRule):
  def __init__(self, timing, recording):
    ...

  def generate_kernel(self) -> str:
    return "...  embedded processor code ..."

my_plasticity = MyPlasticity(...)

pop = pynn.Population(
  ...,
  Neuron(plasticity_rule=my_plasticity))

proj = pynn.Projection(
  ...,
  Synapse(plasticity_rule=my_plasticity))

# ... experiment protocol

proj.get_data("my_syn_obsv")
pop.get_data("my_nrn_obsv")
```

- plasticity rule
  - timing
  - recording
  - code for the embedded processor
- acting on network elements
- post-execution access to recorded data

Plasticity abstraction: kernel code & data flow

```
void PLASTICITY_RULE_KERNEL(
  array<SynapseArrayViewHandle, N> const& synapses,
  array<NeuronViewHandle, M> const& neurons)
{ ... }
```

- embedded processor code
  - functional interface
  - access to network entities
- code generation for
  - rule scheduling
  - network entity handles

Plasticity abstraction: kernel code & data flow

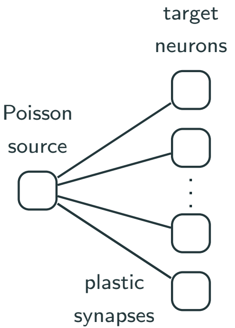
```
void PLASTICITY_RULE_KERNEL(
  array<SynapseArrayViewHandle, N> const& synapses,
  array<NeuronViewHandle, M> const& neurons,
  Recording& recording)
{ ... }
```

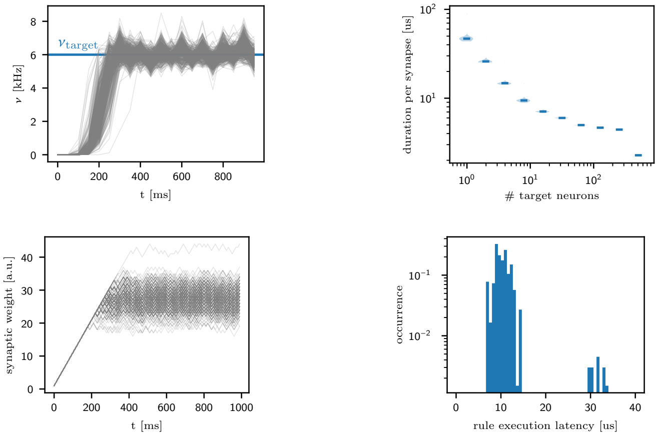
```
observables = {
  "my_obsv": PlasticityRule.ObservablePerSynapse(
    uint8, LayoutPerRow.packed),
  ...
}
```

```
struct Recording
{
  ObsvPerSynPacked<uint8_t> my_obsv;
  ...
};
```

- embedded processor code
  - functional interface
  - access to network entities
  - access to recording
- customizable observable specification
- code generation for
  - rule scheduling
  - network entity handles
  - recording structure

Evaluation via simple homeostatic rule





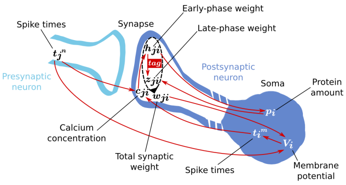
$$\Delta w = \mathrm{sign}\left(\nu_{\text{target}} - \nu\right)$$

Application Dietrich et al. *Sequence Learning with Analog Neuromorphic Multi-Compartment Neurons and On-Chip Structural STDP*, ACAIN 2024.

Application Atoui et al. *Multi-timescale synaptic plasticity on analog neuromorphic hardware*, NICE 2025.

Poster: Atoui et al. Multi-timescale synaptic plasticity on analog NMHW

Plasticity in biology

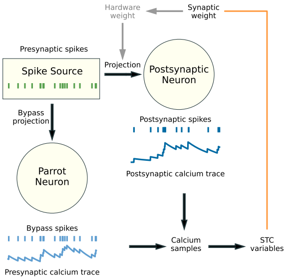


$$\frac{\mathrm{d}h(t)}{\mathrm{d}t} = \alpha\Big(h(t), c(t)\Big)$$

$$\frac{\mathrm{d}p(t)}{\mathrm{d}t} = \beta\Big(p(t), h(t)\Big)$$

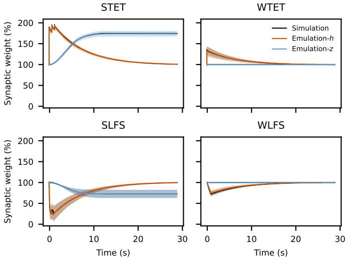
$$\frac{\mathrm{d}z(t)}{\mathrm{d}t} = \gamma\Big(z(t), h(t), p(t)\Big)$$

Plasticity on BrainScaleS-2



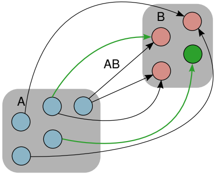
- Adaptation traces for emulating calcium
- Reduced precision (integer arithmetics)

Results

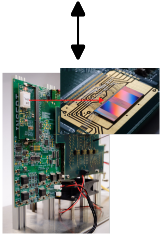


- Accurate emulation of the plasticity rule

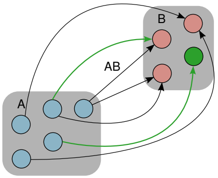
Summary



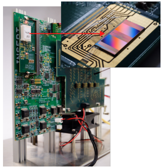

```
class MyRule(pynn.PlasticityRule):
    ...

pop = pynn.Population(...(my_rule))

pop.get_data("my_obsv")
```

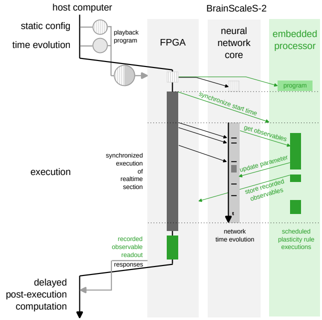


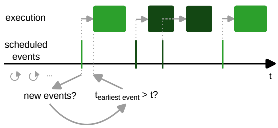
Summary



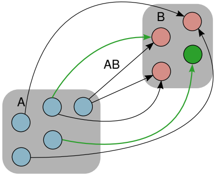

```
class MyRule(pynn.PlasticityRule):
    ...

pop = pynn.Population(...(my_rule))

pop.get_data("my_obsv")
```







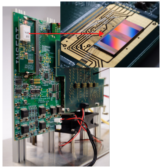
Summary

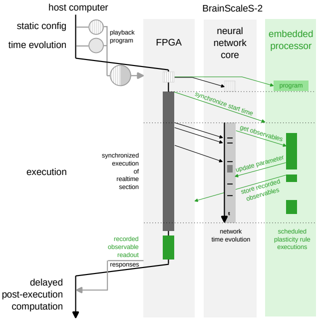


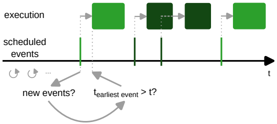
class MyRule(pynn.PlasticityRule):
    ...

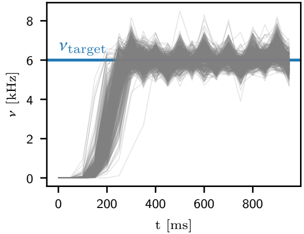
pop = pynn.Population(...(my_rule))

pop.get_data("my_obsv")



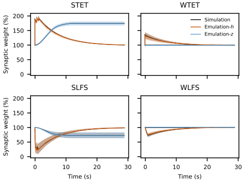






$$\Delta w = \operatorname{sign}\left(\nu_{\text{target}} - \nu\right)$$

Poster *Multi-timescale synaptic plasticity*



Outlook

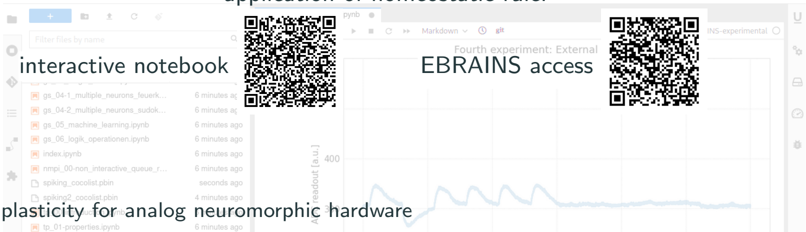
- embedded processor hardware optimizations
    - SIMD unit instruction set enhancements (floats?)
- domain-specific language for plasticity rule code
    - removes need for low-level knowledge about the embedded processors
- combining online plasticity and gradient-based learning
    - meta-learning on plasticity rule hyperparameters, local regularization

Outlook

- embedded processor hardware optimizations
  - SIMD unit instruction set enhancements (floats?)
- domain-specific language for plasticity rule code
  - removes need for low-level knowledge about the embedded processors
- combining online plasticity and gradient-based learning
  - meta-learning on plasticity rule hyperparameters, local regularization

---

application of homeostatic rule:



interactive notebook

EBRAINS access