Networks of Spiking Neurons Learn to Learn

(work in progress)

Wolfgang Maass

Institute for Theoretical Computer Science
Graz University of Technology, Austria

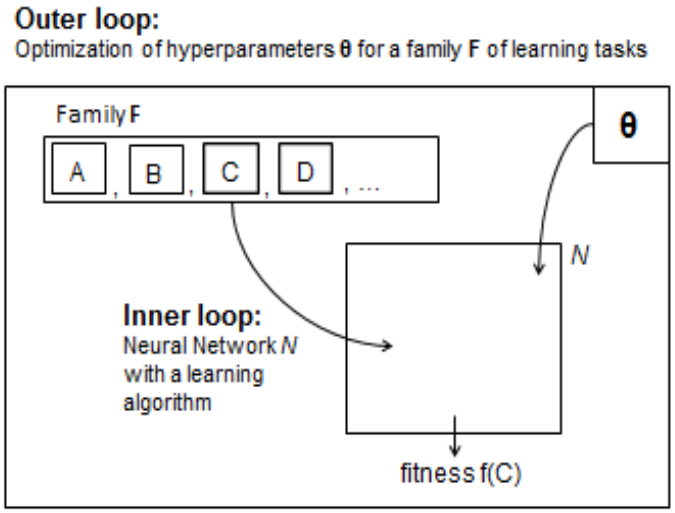
Learning-to-learn (L2L) is inspired by biology, including evolution

- L2L (also referred to as Metalearning) had already been discussed for several decades in neuroscience, cognitive science, and machine learning.

- But only recently, with sufficient computational power being available (+ very nice new ideas), it has become an important tool in machine learning (in particular for Deep Learning).

- But so far it has apparently not been applied to networks of spiking neurons, or to neuromorphic devices.

Standard L2L framework



Instead of single learning tasks, we consider a **family F of learning tasks** (F is in general infinitely large).

The first art is to define F in such a way, that L2L produces a desired result.

Define for any learning task C from the family F the **fitness** f(C) of a NN *N* with hyperparameters (hp's) **Θ** by evaluating how fast and/or how well it can learn C. These hp's Θ remain fixed while the NN *N* learns the task C (in the „inner loop").

The second art is to define the fitness function.

The third art is to choose the right NN and the right set of hp's.

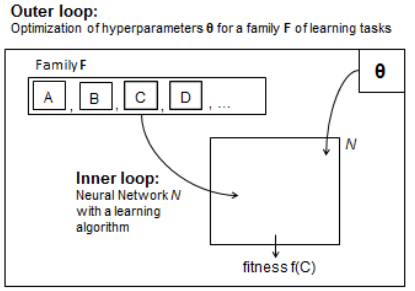
One optimizes Θ through some „outer loop" optimization (e.g. ES) or learning algorithm (e.g. BPTT) to produce high fitness f(C) for a (new) randomly drawn task from F.

Typically this outer loop optimization involves large numbers of learning episodes for many different tasks from F, hence it tends to be computation-intense.

An essential difference to the standard praxis of Machine Learning:

Testing is **not** carried out for new examples from the **same learning task**, but for new examples from a **new learning task** from the same family F.

Example setup for application of L2L to neuromorphic devices *N*



Try to capture in the definition of F the set of learning tasks that a neuromorphic device *N* is likely to encounter during its lifetime.

*Choose hp's **θ** so that they define all aspects of learning in the neuromorphic device N about which one is not sure, e.g.*

- architecture of *N*
- time constants and other physical parameters
- **some** or **all** of the synaptic weights of *N*
- plasticity rules and their parameters
- learning curriculum and learning rates.

First applications of L2L to SNNs

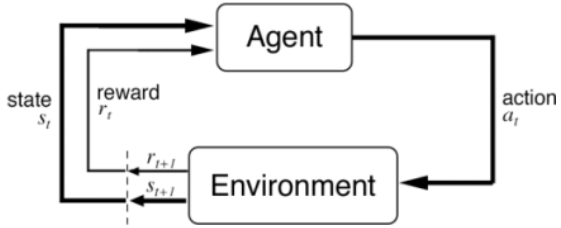
The L2L approach can be applied to any type of learning tasks C for the network *N* in the inner loop: supervised, unsupervised, reinforcement learning (RL).

I will present first demos for RL and supervised learning.

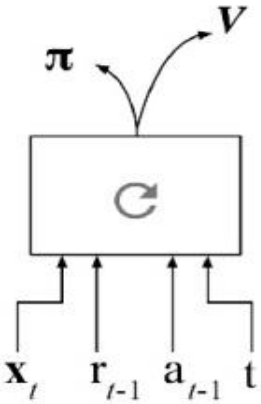
We use **benchmark challenges** that were proposed for L2L applied to **non-spiking** recurrent ANNs (LSTM networks):

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., ... & Botvinick, M. (2016). Learning to reinforcement learn. arXiv preprint arXiv:1611.05763.

Hochreiter, S., Younger, A. S., & Conwell, P. R. Learning to learn using gradient descent. ICANN 2001.



Demos for letting SNNs learn to learn from rewards



- The fitness f(C) reflects here the **the sum of rewards** that *N* receives

- *N* gets in addition, like in (Wang et al, 2016), at each step t the current state, as well as the action and reward for the **preceding** time step.

- Hence it **can** compare the expected and actual reward for the preceding time step, e..g, in order to update its value function.

- The weights W of the NN in the inner loop are hp's, that are optimized by the **outer** loop. Information gathered while **learning** the current task C has to be **stored in the dynamic state of *N***.

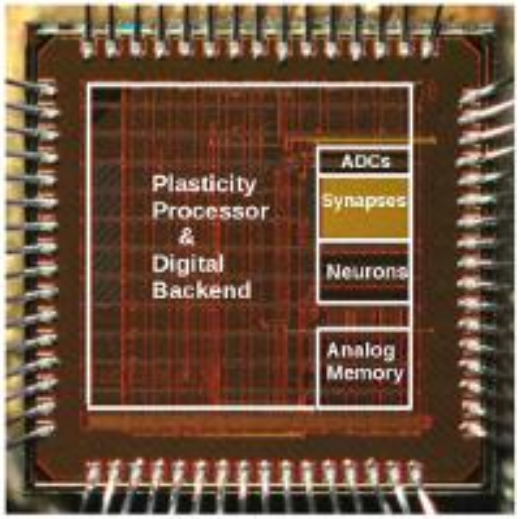
- One can achieve that by employing new „**spiking LSTM modules"** in the SNN, or by using digital registers on a neuromorphic chip.

- The resulting **weight vector W** of the SNN encodes **the RL-algorithm** that it learnt to use for tasks from the family F,

Botvinick et al present in forthcoming work **experimental data from neuroscience which suggests that brains use a similar method for fast learning (**without changing synaptic weights).

Application to a neuromorphic chip

We implemented learning-to-learn from rewards on the very fast and energy-efficient spiking **HICANN-DLS** chip , using mixed analog-digital technology from the Lab of Karlheinz Meier in Heidelberg.



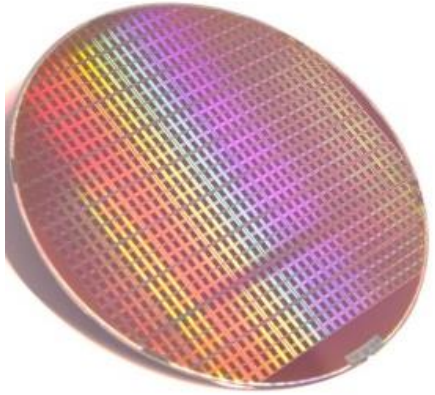
This chip is a small prototype (32 neurons) for larger chips that will form the basis of larger systems with wafer-scale integration.

The outer loop of L2L was implemented with the digital plasticity processsor on the chip.

It optimized hp's $\boldsymbol{\theta}$ that controlled physical parameters as well as details of the learning process.

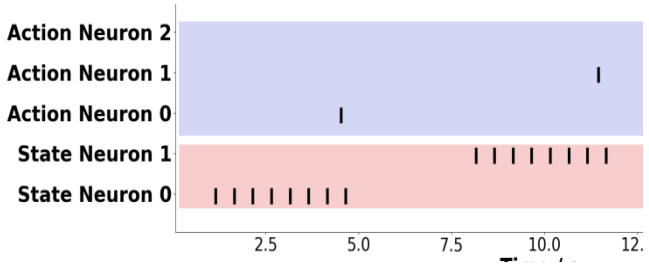
The hardware SNN applied TD-lambda to learn the Q-function.

Digital registers of the plasticity processor were used as working Memory.



Continued: L2L application to the HICANN-DLS

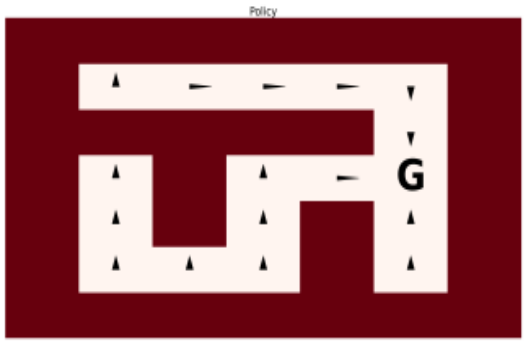
Action choices were encoded on the level of single spikes (**not rates!**)
Both parameters of the learning rule (TD-learning) and of the hardware were hp's that were optimized in the outer loop.

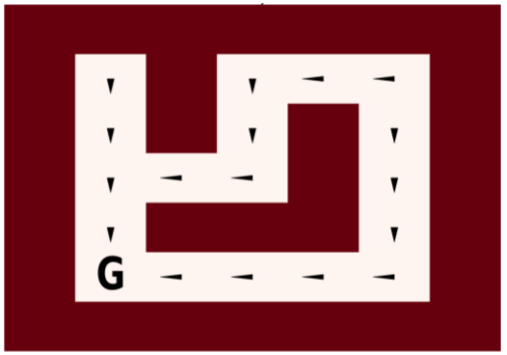


We tried this out for two **families F** of RL-tasks:
--**navigation** to a goal G in **random mazes** of a given size
--**random MDPs** of a given size.

For both families the HICANN-DLS acquired transfer learning capability:

Shown are the policies learnt after 1000 steps for navigating to a goal G in a new random maze, **before** and **after** L2L

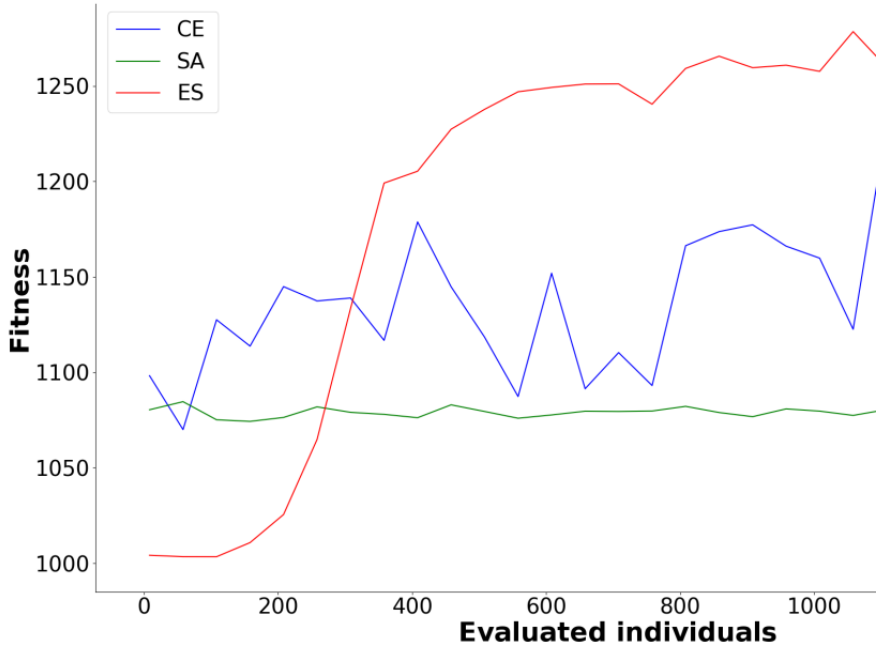




Choosing a good optimization algorithm for the outer loop is essential

We examine here the emergence of transfer learning capability for the **family F** of **random MDPs** with 6 states and 8 actions

Fitness was the sum of rewards collected during a learning episode of fixed length (2000 steps), in dependence of the number of previously learnt MDPs.



ES= Evolution Strategies (Salismans et al., 2017)

CE = Cross Entropy Method

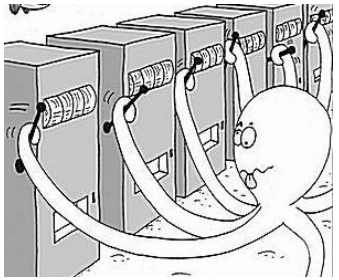
SA = Simulated Annealing

Result: L2L with ES in the outer loop, but not with SA, endows the HICANN-DLS with substantial transfer learning capability.

We can expect a qualitative jump in the computational capability of SNNs through L2L

Wang et al proposed a really mean family F of learning tasks for LSTM networks:

The reward of the first arm of a 11-armed bandit encodes the „address“ of that one of the other 10 arms that has the highest reward probability.

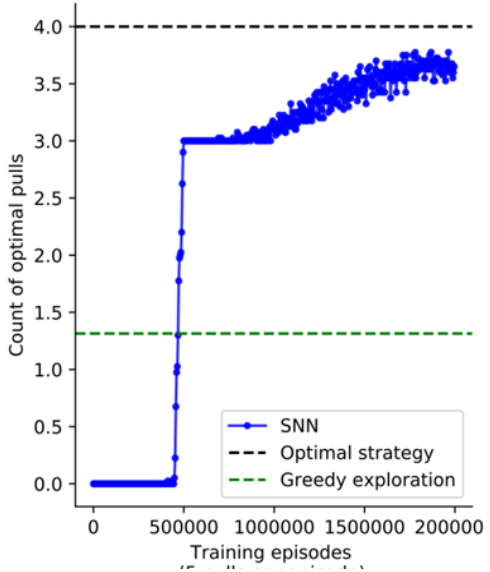


It turns out that SNNs (with „spiking LSTM modules“) can learn to cope with that.

The SNN had to decide in 20ms which arm to pulll next.

The recurrent SNN was optimized via BPTT to learn a policy and a value function according to the Advantage-Actor-Critic algorithm (Mnih et al, 2016).

After 500 000 learning episodes the recurrent SNN discovered a policy that used a simple sequential program to explore multi-armed bandits from this family F.



SNNs can also learn-to-learn from a *teacher*

- Here the SNN *N* learned to give output values that approximated target values that were given by a teacher („**supervised learning**“).

- Benchmark family F from (Hochreiter et al., 2001): Quadratic functions G of the form

$$G(x_1, x_2) = a\, x_1^2 + \mathrm{b}\, x_2^2 + \mathrm{c}\, x_1\, x_2 + d\, x_1 + e\, x_2 + f$$

with analog parameters a, b, c, d, e, f from [-1, 1], and arbitrary analog input values x_1, x_2 from [-10, 10], outputs scaled to [0.2, 0.8].

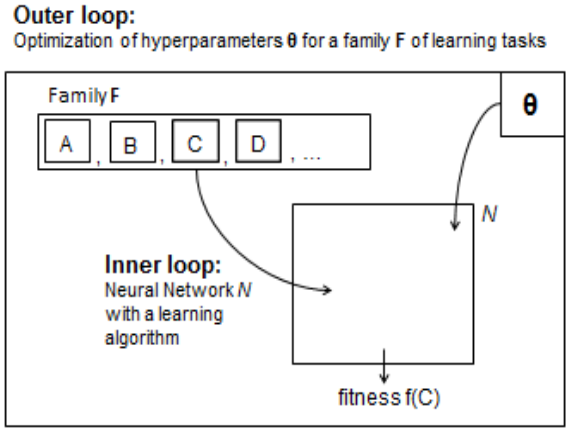
- Together with a new sample < x_1, x_2 > the network gets as additional input the target value $G(x_1', x_2')$ for the **preceding** input sample $<x_1', x_2'>$.

- This was the learning setup proposed in (Hochreiter et al, 2001), where each **teacher value** is given to the network in a **delayed** manner, so that it cannot cheat.

- If it wants, the SNN can store the preceding input $<x_1', x_2'>$ and compute the error that it made for it. ***But it does not have to do that.***

- Here **the SNN is not allowed to change its weights *W* for learning a particular function G**, since *W* is included in the hp‘s, and hence is determined by the outer loop. Thus the **weights W of the SNN also encode its learning algorithm.**
- But one can just as well tune none or just some of the weights in the outer loop.

Implementation details



- For each target function G one draws randomly input samples $< x_1, x_2 >$, transforms them via population coding into spike trains, and injects these spikes for 10ms into the recurrent SNN *N* (which consisted here of 200 neurons).

- The SNN gives output values through a linear readout from the spiking neurons.

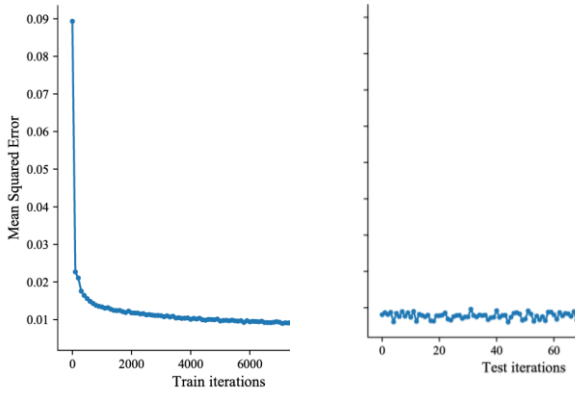
- **One modifies the weights W of the SNN and of the linear readout via BPTT** (i.e., Deep Learning applied to SNNs).

- Since the quadratic target functions G change during training, the weights W do not specialize for computing a particular function G.

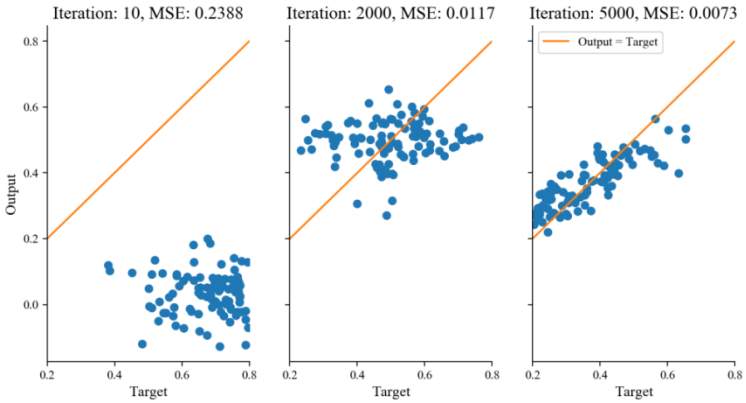
- Key difference to earlier uses of SNNs in **liquid computing**: There the weights of the SNN were randomly chosen, not optimized for a range of learning tasks.

After many iterations, the SNN learned to learn from a teacher

During the initial stages of training in the outer loop the SNN in the inner loop gets its basics right, such as giving output values in the same range [0.2, 0.8] as the target functions.



Sample performance for random inputs $< x_1, x_2 >$, at 3 stages of training in the outer loop:



After training the SNN in the outer loop, the **weights W of the SNN and the weights of the linear readout are frozen**. Nevertheless the SNN is enabled to learn from a teacher any concrete quadratic function G.

Summary I

- I have presented proof of concept that the addition of „spiking LSTM modules“ allows us to port computational and learning capabilities of recurrrent ANNs into recurrent SNNs.

- Learning to learn from rewards can also be applied to neuromorphic devices such as the HICANN-DLS, where digital registers can hold working memory.

- In this way neuromorphic devices are enabled to carry out transfer learning.

- For SNNs in software simulations one can use Deep Learning for producing computationally powerful recurrent SNN architectures, and for the design of learning algorithms for SNNs.

Summary II

- We are currently developing methods for understanding how the resulting SNNs and learning algorithms do their job.
-
- The resulting SNNs I have seen so far appear to compute with spike patterns, rather than rates.

- This reverse engineering of optimized SNNs will produce a better understanding of the computational role of specific SNN architectures and components, and create new links to neuroscience.

- I expect that L2L will revolutionize the design of recurrent SNNs for concrete computations, as well as the design of learning algorithms for recurrent SNNs.

- In particular, we will be able to produce SNNs that are able to follow rules, make use of episodic memory for decision making, and carry out symbolic computations and reasoning..

Credits

Researchers from our Lab:

Thomas Bohnstingl

Darjan Salaj

Anand Subramoney

Guillaume Bellec

Robert Legenstein

Collaborators at the University of Heidelberg:

Benjamin Cramer, Karlheinz Meier, Christian Pehle, David Stöckel

Funding: Human Brain Project of the EU