# Milestone Report: Extending neuromorphic classifier to a large-scale platform

Alan Diamond, Michael Schmuker, Thomas Nowotny

University of Sussex, January 2016

## Introduction

This reports summarises the work undertaken to extend the deployment of the neuromorphic classifier model for MNIST digit recognition developed at University of Sussex on small neuromorphic hardware to run at larger size on a scaled up neuromorphic hardware platform.

## Previous work

In previous HBP outputs (Diamond et al., 2015b, 2016) we have described the adaptation of our spiking classifier model to the MNIST dataset and its implementations on GPU (via “GeNN”: Yavuz et al., 2016), Spikey neuromorphic and, particularly, the SpiNNaker platform (4-chip , “SPiNN3” board). The classifier design takes inspiration from the insect olfactory system (see Diamond et al., 2015a, 2016; Schmuker and Schneider, 2007; Schmuker et al., 2014) by placing a set of “virtual receptors” (VRs) in input space. In the model these provide scaled input through the rate coded net activation per VR of a cluster comprising some 30 “receptor” (Poisson spiking) neurons. In the work reported, the SPiNN3 board was able to accommodate some 200 of these VR clusters, resulting in a 10 digit (classes) classifying model comprising some 12K neurons, 18M synapses, and running in real time.

## Scaling up

Previous work (e.g. Diamond et al., 2016) has shown that this classifier’s performance can be improved by extending the set of receptors in input space. On conventional hardware, scaling up the number of neurons in a high connectivity model like our classifier leads to quadratically-increased simulation times. On a highly parallel platform, such as the neuromorphic platforms employed here, it is expected that this penalty can be largely sidestepped through the use of higher capacity hardware. It is therefore of interest to assess the overall performance and behaviour when the model design is retained but the number of VRs is scaled significantly.

## Large Scale platform

We employed a scaled up SpiNNaker system, namely the 48-chip, SPiNN5 board. This allowed the existing SpiNNaker development to be employed with suitable modifications developed to accommodate issues arising from the expanded scale. The initial proposal to employ the Heidelberg “wafer-scale” platform (ref) proved unfeasible due to the public unavailability of this platform at the time.

## Results

Following the implementation of the set of modifications described below, it was possible to scale the model to successfully produce an enhanced classification performance employing some 500VRs (30K neurons, 112M synapses). Beyond this point the current Python-based model configuration routines within the SpyNNaker software (“Arbitrary” release) slowed to an impractical level with model start-up times passing the one hour mark.

## Obstacles and solutions

Three main obstacles were encountered in the attempted scaling. Firstly, the input bandwidth to accommodate the very large static spike source data files that comprised the input to the model, incorporating the presentation of several thousand MNIST digits. Secondly, the output bandwidth to retrieve all the spikes occurring in the output layer across an entire training or testing run. These spikes are required to determine the “winning” output and hence the classifier’s decision on the class of a given MNIST digit. Finally, the time required for the Python-based PyNN/SpyNNaker routines to configure and upload the model on the SpiNNaker hardware increased in line with the number of synapses specified, this rises approximately quadratically with number of VRs specified. These three main issues will now be dealt with in turn.

## Input bandwidth

On the large scale deployment this is addressed with two mechanisms. Firstly, eliminating the use of spike source files and developing an interface to pass rate-coded input as real-time “live” spike injections using the SpiNNaker UDP interface. Due to the scale of simultaneous spike injection channels required (500+), this was developed in C as a threaded application. The resultant C library is made available as an open source output (see section below on published software).

Secondly, a model was developed whereby the Poisson-based spiking of a whole population (VR cluster in this case) can be made to depend linearly on the spike rate of a single driving neuron. This means that the external input generator need only generate and transmit spikes for a single neuron per VR, rather than an entire cluster (30 times larger). This mechanism is detailed in the supplementary material of (Diamond et al., 2016), see

http://journal.frontiersin.org/file/downloadfile/37235/octet-stream/Supplementary Material.PDF/31/2/164125.

## Spike collection and output bandwidth

All spikes occurring in the output layer across an entire training or testing run must be retrieved to determine the “winning” output and hence the classifier’s decision on the class of any given MNIST digit example. The original implementation used the PyNN population “getSpikes” functionality followed by partition by the spike timings to determine, in conjunction with the input spike source, which digit was being presented. The scaled-up version presents a larger spike count, along with synchronisation/timing issues introduced by the live spike injection mechanism. This was addressed by constructing a further threaded host-side spike receiver using the live spike collection mechanism provided by SpiNNaker, again over UDP protocol. This allowed output spikes occurring in response to a given presentation to be collected and counted simultaneously. Again, the C implementation is provided in the open source code release accompanying this report.

## Configuration time

The classifier model generates PyNN populations and PyNN projections (inter-connection matrices) in each layer of the design using an optimisation metric obtained empirically from the development of the first version. This model preferentially uses as many SpiNNaker cores as are available. However, this leads to an untenably high number of projection objects being created in memory (500 VRs in WTA formation implies some 250,000 separate projections), leading to gigabyte-level memory object storage requirements. This metric was therefore altered to combine population clusters onto a minimum number of cores. This allowed the large model sizes to be accommodated on the host in order to generate the SpyNNaker-deployed implementation.

Nevertheless, we found that the time required for the Python-based PyNN/SpyNNaker routines to configure and upload the model on the SpiNNaker hardware increased in line with the number of synapses specified, this rises approximately quadratically with the number of VRs specified. At around 600VRs (160M synapses) this passes a 1 hour delay before starting simulation, some 8 times longer than the 200VR original implementation on the SPiNN3 board.

## Conclusions and recommendations

Note that we were unable to exercise more than around 20% of the SpiNN5 board’s nominal neuron carrying capacity before the issues described became untenable for useable development or research work. The following recommendations are made to potentially alleviate these:-

* Provide functionality for pausing and altering Poisson spike rate for a population “on-the-fly’.
* Provide gamma distribution - based rate coding for populations, this has been shown to allow smaller clusters to act out WTA style competition effectively.
* Profile and optimise the configuration routines for large models with significant connectivity. Rewriting these routines in the fastest possible language (e.g. C plus CUDA/GPU) should also help considerably.

## Published open source software

As an output, the resultant SpiNNaker implementation for large-scale version of the classifier network, including the C-based library developed for live spike injection and collection has been made available alongside the first spike-source based classifier model for SpiNNaker (see previous milestone release). The repository is available at the GITHUB **spinnaker-neuromorphic-classifier** project (see

*https://github.com/alandiamond/spinnaker-neuromorphic-classifier/tree/master/Spinn5-LargeModel-LiveSpiking*).

## References

Diamond, A., Nowotny, T., and Schmuker, M. (2016). Comparing Neuromorphic Solutions in Action: Implementing a Bio-Inspired Solution to a Benchmark Classification Task on Three Parallel-Computing Platforms. *Front. Neurosci.* 9. doi:10.3389/fnins.2015.00491.

Diamond, A., Schmuker, M., Berna, A., Trowell, S., and Nowotny, T. (2015a). Classifying continuous, real-time e-nose sensor data using a bio-inspired spiking network modeled on the insect olfactory system. *Bioinspir. Biomim.*

Diamond, A., Schmuker, M., Yavuz, E., Turner, J., and Nowotny, T. (2015b). Implementing neuromorphic Computing with Large, High-Dimensional Data Sets Using GeNN - a Meta-Compiler for Neuronal Modelling on General Purpose GPU-Accelerators. in *2015 Conference on Brain Informatics and Health*, 21.

Schmuker, M., Pfeil, T., and Nawrot, M. P. (2014). A neuromorphic network for generic multivariate data classification. *Proc. Natl. Acad. Sci.* 111, 2081–2086. doi:10.1073/pnas.1303053111.

Schmuker, M., and Schneider, G. (2007). Processing and classification of chemical data inspired by insect olfaction. *Proc. Natl. Acad. Sci. U. S. A.* 104, 20285–9. doi:10.1073/pnas.0705683104.

Yavuz, E., Turner, J., and Nowotny, T. (2016). GeNN: a code generation framework for accelerated brain simulations. *Sci. Rep.* 6, 18854. doi:10.1038/srep18854.