# POSTER: Bridge the Gap Between Neural Networks and Neuromorphic Hardware

Yu Ji* YouHui Zhang*†* WenGuang Chen*† Yuan Xie‡
* *Department of Computer Science and Technology, Tsinghua University, PR.China*
† *Center for Brain-Inspired Computing Research, Tsinghua University, PR.China*
‡ *Department of Electrical and Computer Engineering, University of California at Santa Barbara, USA*
*zyh02@tsinghua.edu.cn

## I. Introduction

Different from the common neural network (NN) training for general-purpose processors (including general-purpose graphic processing units, GPGPU), for neural network chips, there are quite a few hardware-specific constraints to make programming such chips difficult: (1) Considering the usage efficiency of hardware resource and/or the feature of analog computation (e.g. memristor-crossbar-based designs [1]), the precision of input and output signals is always limited, as well as (2) the precision of weight values. (3) The present fabrication technology restricts the maximum number of connections that one neuron can own(e.g. TrueNorth chips [2]). (4) Moreover, the diversity of supported activation function (or neuron model) is limited.

This paper proposes a "trained-and-then-constrained" solution for this problem to bridge the gap between existing (software) ANN models and hardware. It transforms trained NNs (for software substrate) onto neuromorphic chips transparently, with very limited accuracy loss. To achieve the above targets, the following design principles are employed:

First, we abstract the target hardware as a set of connected cores (referred as *virtual core*), each of which is a vector-matrix multiplication engine (the crossbar structure is such a case) equipped with activation function or spiking neuron model, constrained by the hardware conditions. Then, it would transform an *existing trained, unrestricted NN* (referred to as the *golden model*) into an equivalent network composed of the aforementioned cores. After transformation, the obtained network of *virtual cores* will be mapped onto real cores (referred to as *physical cores*) of the target hardware for higher resource usage.

Second, to transform the golden model as a whole under strict hardware constraints would be difficult to converge. Thus, a "divide-and-conquer" method is proposed to divide the golden model into a set of basic network units (referred as an *NN subgraph*, or just *subgraph*), and then to confine the subsequent processing of various hardware constraints to such one or more relatively simple units. Afterwards, each unit is transformed into an equivalent network composed of *virtual cores*: several adaption phases are presented to fine-tune parameters of each unit to reduce the network error.

Third, the essential of this solution is to construct a network of constrained cores to approach the golden model. As hardware neurons and synapses are not comparable to their software counterparts, we propose a set of methods to enlarge the hardware network and/or complicate the topology properly to improve capability, especially under very strict conditions[1].

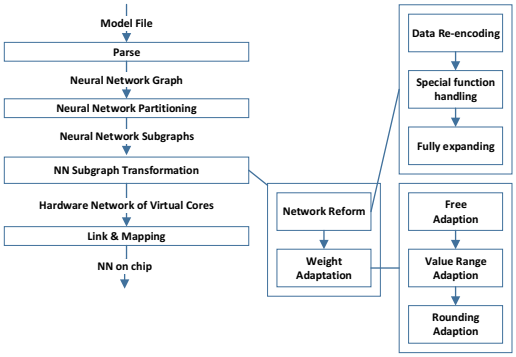


Figure 1. Workflow of our proposal.

Finally, it is necessary to emphasize that, although in the above content we take the crossbar-based architecture as the example, our solution is not constrained by the underlying architecture. As long as the target hardware is composed of networked processing units and each unit is constrained by one or more items of the following four tuple, $N$ (the maximum neuron number/fan-out), $M$ (the maximum axon number/fan-in), $U$ (the data width of a synaptic weight) and $V$ (the data width of input/output signals), it can be supported because our proposal uses independent technologies to deal with different types of constraints respectively.

Currently, we have implemented such a preliminary transformation tool, which can convert trained NNs (including some ImageNet-level networks) into equivalent counterparts that meet hardware limitations; the extra error caused by this process is very limited and time overhead is much less (compared to the whole training process of the original NN).

## II. Transformation workflow

The workflow is analogous to the traditional compiling procedure (Figure 1), which involves 4 steps:

### A. Parse model files to build the NN graph

The input of our workflow is the description of the golden model, including the trained parameters, network structure,

[1]Usually, there is significant redundancy for DNN models; compression is just used to remove the redundancy. Accordingly, we also employ network compression techniques to reduce the NN size, which is not contradictory to this principle.

978-1-5090-6764-0/17 $31.00 © 2017 IEEE
DOI 10.1109/PACT.2017.59

training dataset and training intermediate data of the golden model (the generated responses of all neurons during the training process). As the fine-tuning is easy to converge, the amount of response data needed is limited, which can be created on demand.

Then, we can construct the original NN as a directed graph $G(V, E)$. Vertex $v \in V$ is a group of homogeneous neurons and Edge $(u, v) \in E$ is the bundle of connections between neurons of $u$ and $v$.

### B. Partition NN graph into subgraphs

As a direct embodiment of the "divide-and-conquer" method, the partitioning strategy needs to meet three conditions: (1) All original computations have to be maintained. (2) Each subgraph should be simple enough to be approximated by a network of *virtual cores* (referred as *hardware network*). (3) When we link all *hardware networks* together to construct the counterpart of the golden model, constraints on fan-in and fan-out should still be met, as we treat every subgraph as the object to handle all the constraint problems. According to these requirements, a partitioning algorithm is proposed: any resulted subgraph contains two fully-connected sets of vertices (denoted as the pre-vertex set and the post-vertex set respectively); any vertex in $G$ must belong to and only belong to one pre-vertex set and one post-vertex set (the input/output vertices are the exception). Moreover, fan-outs of all vertices of the pre-vertex set (and fan-ins of the post-vertex set) of every subgraph should be confined to this subgraph.

### C. Transform every NN subgraph into hardware network

It is composed of two sub-steps. The first aims to improve the approximating capability of *hardware network*, which contains three phases: (1) Re-encoding neuron activations with autoencoders to improve the precision problem of I/O signals. (2) Implementation of special functions with MLPs (multilayer perceptrons). (3) Expanding the network generated by previous phases to meet the hardware limitation on vector-matrix multiplication scale.

The second sub-step modifies synaptic weights of *hardware network* properly to minimize network errors (compared to the golden model), including several adaption phases to deal with the weight-precision limitation.

The transformation of each subgraph may introduce errors. To prevent error accumulation, all *hardware networks* will be fine-tuned sequentially in topological ordering of the NN graph: for each one, the input is just the output of previously processed one(s), and then it is fine-tuned to approximate the corresponding output from the golden model's intermediate training-results.

### D. Link all hardware networks together and map them to target hardware

It means that the golden model has been finally transformed to a large *hardware network* composed of *virtual cores* that meets hardware constraints. Then we can map each *virtual core* to the target hardware for higher resource utilization.

## III. Evaluation and Conclusion

We have evaluated the tool under the constraints of a neuromorphic chip (Tianji chip [3]). The numerical accuracy of weight values is 8-bit fixed-point and the scale of vector-matrix-multiplication is $256 \times 256$. The I/O precision is 8-bit that is cut from the 24-bit internal computation output; the dynamic-fixed-point strategy is used for weight representation. The original NNs include an MLP for MNIST dataset (784-100-10 structure, $98.09\%$ accuracy), LeNet-5 for MNIST dataset ($99.05\%$ accuracy), AlexNet and VGG16 for ImageNet. The first two networks are trained using Theano [4]. The parameters of the next two CNNs for ImageNet are extracted from trained models of the Caffe Model Zoo directly.

We take the inference result of the golden model as the ground truth to show the error introduced by our method. Results show that under these constraints, the errors are almost negligible: all error rates are less than $5\%$.

As a conclusion, this paper presents an NN development method for neuromorphic system. Compared to existing methods, it is focused on the feature of decoupling NN applications from the underlying execution substrate. The evaluation shows that the extra error caused by this process is negligible.


## Acknowledgment

The work is supported by the National Key Research and Development Program of China under Grant No.2016YFB0200505 and the Science and Technology Plan of Beijing under Grant No. Z161100000216126.


## References


[1] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, *et al.*, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 24, no. 521, pp. 61–64, 2015.

[2] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[3] L. Shi, J. Pei, N. Deng, D. Wang, L. Deng, Y. Wang, Y. Zhang, F. Chen, M. Zhao, S. Song, F. Zeng, G. Li, H. Li, and C. Ma, "Development of a neuromorphic computing system," in *2015 IEEE International Electron Devices Meeting (IEDM)*, pp. 4.3.1–4.3.4, Dec 2015.

[4] R. Al-Rfou, G. Alain, A. Almahairi, and etc., "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.