# In-network Neural Networks

Giuseppe Siracusano, Roberto Bifulco
NEC Laboratories Europe

## 1 INTRODUCTION

Network devices, such as switches and routers, process data at rates of terabits per second, forwarding billions of network packets per second. Recently, such devices' switching chips have been enhanced to support new levels of programmability [3]. Leveraging these new capabilities, a switching chip's packets classification and modification tasks can now be adapted to implement custom functions. For example, researchers have proposed approaches that rethink load balancers [11], key-value stores [7], and consensus protocols [5] operations. In general, there is a trend to offload to the switching chips (parts of) functions typically implemented in commodity servers, thereby achieving new levels of performance and scalability.

These solutions often offload some data classification tasks, encoding relevant information, e.g., the *key* of a key-value store entry [10], in network packets' headers. Unlike packets' payload, the header values can be parsed and processed by the switching chips, which perform classification using lookup tables. While providing very high throughput, lookup tables need to be filled with entries that enumerate the set of values used to classify packets, and therefore the table's size directly correlates to the ability to classify a large number of patterns. Unfortunately, the amount of memory used for the tables is hard to increase, since it is the main cost factor in a network device's switching chip [3], accounting for more than half of the chip's silicon resources.

In this paper, we explore the feasibility of using an artificial neural network (NN) model as classifier in a switching chip, as a complement to existing lookup tables. A NN can better fit the data at hand, potentially reducing the memory requirements at the cost of extra computation [9]. Here, our work builds on the observation that, while adding memory is expensive, adding circuitry to perform computation is much cheaper. For reference, in a programmable switching chip the entire set of computations is implemented using less then a tenth of the overall chip's area.

To this end, we implement N2Net, a system to run NNs on a switching chip. We provide the following contributions: first, we show that a modern switching chip is already provided with the primitives required to implement the forward pass of quantized models such as *binary neural networks*, and that performing such computation is feasible at packets processing speeds; second, we provide an approach to efficiently leverage the device parallelism to implement such models; third, we provide a compiler that, given a NN model, automatically generates the switching chip's configuration that implements it. Our experience shows that current switching chips can run simple NN models, and that with little additions a chip design could easily support more complex models, thereby addressing a potentially larger set of applications.

**Use cases** At the time of writing we are still in the process of implementing full-fledged applications, thus, we just mention our two initial use cases, postponing to a later publication a throughout technical description. First, similar to [9], we envision the use of a neural network classifier to implement packet classification inside the chip, e.g., to create large white/blacklist indexes for Denial of Service protection. Second, the outcome of the NN classification can be encoded in the packet header and used in an end-to-end system, to provide "hints" to a more complex processor located in a server, e.g., on how to handle the packet's payload to optimize data locality/cache coherency or to support load balancing [15].

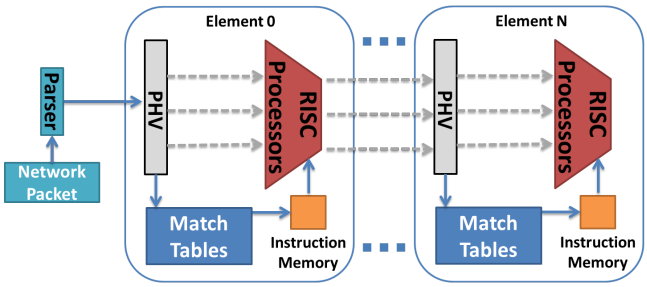


**Figure 1: A schematic view of a switching chip's pipeline.**

*To encourage the community in exploring more use cases, we are in the process of making N2Net code publicly available.*

## 2 N2NET

**Switching chip primer** Switching chips process network packets to take a forwarding decision, e.g., forward to a given port or drop. They also perform transformations to the packet header values, e.g., to decrement a time-to-live field. We use RMT [3] as representative model of state-of-the-art switching chips (Cf. Fig. 1). When a packet is received, an RMT chip parses several 100s bytes of its header to extract protocol fields' values, e.g., IP addresses. These values are written to a packet header vector (PHV) that is then processed by a pipeline of *elements* that implement match-action tables. Each element has a limited amount of memory to implement lookup tables (the match part), and hundreds of RISC processors that can read and modify the PHV in parallel (the action part). The values in the PHV are used to perform table lookups and retrieve the instruction the processors should apply. To provide very high performance, these processors implement only simple operations, such as bitwise logic, shifts and simple arithmetic (e.g., increment, sum). Using a language such as P4 [2], the chip can be programmed to define the parser logic and the actions performed on the PHV. In particular, the actions are defined as a combination of the simpler primitives mentioned earlier.

**Design** The limited set of arithmetic functions supported by a switching chip does not enable the implementation of the multiplications and activation functions usually required by a NN. However, simplified models designed for application in resource-limited embedded devices, such as *binary neural networks* (BNNs), do not require such complex arithmetic, especially during the forward pass [4]. In our case, we select models that only use bitwise logic functions, such as XNOR, the Hamming weight computation (POPCNT), and the SIGN function as activation function. While research in these models is at its early stages, it shows already promising results [6, 8, 13].

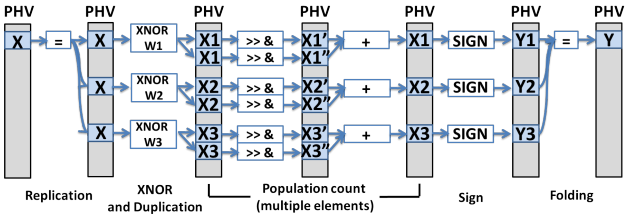


**Figure 2: Implementation of a 3 neurons BNN processing.**

| Activations (bits) | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 |
|---|---|---|---|---|---|---|---|---|
| Parallel neur. (max) | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| Elements number | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 25 |

**Table 1: Maximum number of parallel neurons and required number of elements for different activations vector sizes.**

N2Net implements the forward pass of a BNN, and assumes that the BNN activations are encoded in a portion of the packet header. The header is parsed as soon as a packet is received, and the parsed activations vector is placed in a PHV's field. Fig. 2 summarizes the operations performed by N2Net to implement a 3 neurons BNN. The entire process comprises five steps:

- **Replication**: in the first step the activations are replicated in the destination PHV as many times as the number of neurons that should be processed in parallel;
- **XNOR and Duplication**: in the second step such fields are processed by the element's RISC processors. The applied actions perform XNOR operations on the activations taking as parameters the neurons' weights. The results are stored twice in the destination PHV. This duplication is required by the implementation of the POPCNT as explained next;
- **POPCNT**: RMT does not support a POPCNT primitive operation. A naive implementation using an unrolled *for* cycle that counts over the vector bits may require a potentially big number of elements. Instead, we adapted a well-known algorithm that counts the number of 1 bits performing additions of partial counts in a tree pattern [1]. The advantage of this algorithm is that it can perform some computations in parallel, while using only shifts, bitwise logic and arithmetic sums. N2Net implements such algorithm combining two pipeline's elements. The first element performs shift/bitwise AND in parallel on the two copies of the input vector. Each copy contains the mutually independent leaves of the algorithm's tree structure. The second element performs the SUM on the outcome of the previous operations. Depending on the length of the activation vector, there may be one or more groups of these two elements, in which case the sum's result is again duplicated in two destination PHV's fields.
- **SIGN**: the fourth step implements the sign operation verifying that the POPCNT result is bigger or equal to half the length of the activations vector. The result is a single bit stored in the least significant bit of the destination PHV's field.
- **Folding**: the last step folds together the bits resulting from the SIGN operations to build the final Y vector, which can be used as input for a next sequence of 5 steps.

N2Net currently implements fully-connected BNNs taking a model description (number of layers, neurons per layer) and creating a P4 description that modifies/replicates the above five steps as needed. BNN are relatively small models whose weights fit in the pipeline element's SRAMs, however, we are required to pre-configure the weights. This is similar to the BrainWave's approach [12].

**Evaluation** Our implementation is subject to two main constraints. First, the PHV is 512B long. Since we use the PHV to store the BNN input, the maximum activation vector length is 2048 (i.e., half the PHV's size, 256B, since we perform the duplication step). Smaller activation vectors enable the parallel execution of multiple neurons, using the replication step of Fig. 2. For example, with a 32b activation vector, up to 64 neurons can be processed in parallel. Second, the RMT pipeline has 32 elements, and each element can only perform one operation on each of the PHV's fields (for a maximum of 224 parallel operations on independent fields in each element). While we implement the POPCNT leveraging parallelism in order to minimize the number of required elements, we still need $3 + 2log_2(N)$ elements to implement a single neuron, where N is the size of the activations vector (cf. Table1). Using the previous examples, the execution of a neuron with 2048 activations would require 25 elements, while with a 32b activations vector we would take just 13 elements. Furthermore, in this last case the addition of the replication step (i.e., an additional element) would correspond to the parallel execution of up to 64 neurons using only 14 out of the 32 pipeline's elements.

In terms of throughput, an RMT pipeline can process 960 million packets per second. Since we encode in one packet our activations, N2Net enables the processing of 960 million neurons per second, when using 2048b activations. Processing smaller activations enables higher throughput because of parallel processing.

To put this in perspective, considering the above constraints, we could run about a billion small BNNs per second, i.e., one per each received packet. For instance, we could run 960 million two-layers-BNNs per second, using 32b activations (e.g., the destination IP address of the packet), and two layers of 64 and 32 neurons.

## 3 CHALLENGES AND OUTLOOK

N2Net shows that BNN models can be implemented with already available switching chip technology. However, models complexity is limited by relevant constraints. In turn, this limits the possible applications. We argue these constraints are the outcome of an architecture that was not designed to run NN models, and that supporting them would require relatively cheap design changes.

For example, implementing a simple POPCNT primitive on 32b operands requires few additional logic gates [16] but could cut the number of required pipeline's elements. I.e., this would change the 12-25 elements range of Table1 to a 5-10 range. Also, this removes the need for the duplication step, immediately doubling the available space in the PHV, hence doubling the neurons executed in parallel.

Furthermore, the circuitry dedicated to computation (including parsers) accounts for less than 10% of the switching chip's area. Using 5-10 pipeline's elements to implement BNN computations takes less than a third of that circuitry. Thus, adding a dedicated circuitry for the execution of BNN computations is likely to account for less than a 3-5% increase in the overall chip area costs.

Overall, we believe N2Net has the potential to open a new interesting field of applications, contributing a novel building block for future networked systems [14].

# REFERENCES


[1] M. Beeler, R. W. Gosper, and R. Schroeppel. Hakmem. Technical report, Cambridge, MA, USA, 1972.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM CCR*, 44(3):87–95, 2014.

[3] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 99–110, New York, NY, USA, 2013. ACM.

[4] M. Courbariaux and Y. Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

[5] H. T. Dang, M. Canini, F. Pedone, and R. Soulé. Paxos made switch-y. *SIGCOMM Comput. Commun. Rev.*, 46(2):18–24, May 2016.

[6] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4107–4115. Curran Associates, Inc., 2016.

[7] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 121–136, New York, NY, USA, 2017. ACM.

[8] M. Kim and P. Smaragdis. Bitwise neural networks. *CoRR*, abs/1601.06071, 2016.

[9] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. 2017.

[10] X. Li, R. Sethi, M. Kaminsky, D. G. Andersen, and M. J. Freedman. Be fast, cheap and in control with switchkv. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 31–44, Santa Clara, CA, 2016. USENIX Association.

[11] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 15–28, New York, NY, USA, 2017. ACM.

[12] Microsoft. Microsoft unveils project brainwave for real-time ai. https://www.microsoft.com/en-us/research/blog/microsoft-unveils-project-brainwave/.

[13] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279, 2016.

[14] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis. In-network computation is a dumb idea whose time has come. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks, Palo Alto, CA, USA, HotNets 2017, November 30 - December 01, 2017*, pages 150–156, 2017.

[15] N. K. Sharma, A. Kaufmann, T. Anderson, A. Krishnamurthy, J. Nelson, and S. Peter. Evaluating the power of flexible packet processing for network resource allocation. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 67–82, Boston, MA, 2017. USENIX Association.

[16] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking. Packet transactions: High-level programming for line-rate switches. In *ACM SIGCOMM '16*, ACM SIGCOMM '16, pages 15–28. ACM, 2016.