Neuro: overview of an Adaptive, Modular, Fault Tolerant Neural Network project

June 9, 2021

Abstract

Description of an ongoing personal project to develop a flexible system integrating different approaches to solve a generic problem, having the nervous system as an inspiration. The system is iterative, dynamically partitioning the input space-time, and assigning each partition to a specific module (Mixture of Experts), picked on the basis of its performance. This allows to combine multiple strategies to solve the same problem, providing improved performance, fault tolerance, efficiency, and reducing its overall cost. The modules are completely generic, e.g. from hard coded sets of rules to time-dependent neural networks (e.g. Spiking Neuron Model). Genetic algorithms are used for configuration, optimization, network structure design, formula evolution; focus on efficiency; extraction of symbolic information from raw data.

A more detailed presentation is available upon request.

Author's email: pietrom16@gmail.com

Key words:

Modular, Adaptive System, Neural Network, Deep-Learning, Spiking Neuron Model, Genetic Algorithms, Mixture of Experts, fault tolerant, time-dependent, real-time, symbolic knowledge, GPGPU, OpenCL, SYCL, TensorFlow.

***Note**: this project is entirely personal, self-founded, developed during my free time, with no external support. If interested, I can let you know about my motivation behind it.*

1 Introduction

Complex problems usually require complex approaches to be solved. There are situations where these are too computationally expensive, or missing altogether because either unknown or non-existent. One technique to face this issue, which dates back to the 1990s and is still progressing today, is the Mixture of Experts model. Basically it is a "divide and conquer" method, where a complex problem is dynamically split into simpler sub-problems.

The purpose of this personal project is to develop a system based on the Mixture of Experts approach, extending it with the addition of modules based, among others, on deep learning architectures and on the Spiking Neuron Model, and setting its configuration at runtime as part of the learning process and offline with genetic algorithms. Each module will specialize on an elementary piece of knowledge, and multiple modules will be combined statically or dynamically to represent higher level concepts. This approach is not affected by the catastrophic interference/forgetting issue, so the whole learning process can be simplified. The capability to automatically extract symbolic knowledge from raw data can be added as a last step.

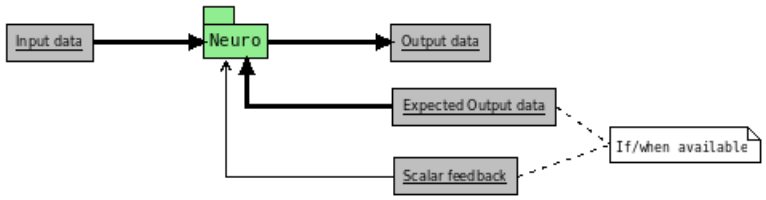
The long term objective is to build a system which imitates the brain at the functional level, i.e. replicating its inner mechanisms as long as they have an impact on its capabilities, an ignoring all the details that depend on its physical and biochemical properties and constraints. Another goal is to combine flexible/dynamic software modules with high performance hardware modules.

This project is still in an embryonic stage (the use of the present tense does not mean a feature is already available). The target hardware is generic, thanks to the use of Standard C++20 and SYCL, allowing it to run on CPUs, GPUs, DSPs and custom hardware. It is being developed in a Linux environment, but it is operating system agnostic at the source code level.

2 Project objectives

Objectives this projects intends to reach:

- Design and implementation of a general purpose system able to integrate any number of different techniques to perform a generic task.
- The task is a mapping between an input and an output (n- and m-dimensional).



- The mapping can be time-dependent.
- The system has an internal memory, which can be fed back as part of the input.
- The system is recurrent; it can work without input, on the basis of what stored in its internal memory.
- The system aims to be memory efficient, high performance, (soft) real-time.
- The system behaviour can be both hard coded and acquired through learning from an optional training set.
  - Provide a seamless integration between symbolic/theoretical knowledge and acquired/experimental data.
- The software is developed with standard tools: Standard C++20, Boost, GPGPU with OpenCL/SYCL.
- Modular architecture:
  - The output is provided by multiple modules.
  - The modules compete and/or cooperate against/with each other.
  - Allows to train on new data only, without forgetting what learnt before.
  - Modules can be implemented as software and/or hardware components.
  - Develop modules with different internal structures, among which: dynamical cell assemblies, neural microcircuits, cortical columns.

- Information coding: spike rates, spike timings (*Spiking Neuron Model*), high level information (text, bitmaps, video, ...).
- Adaptive system:
  - Explicit knowledge of the problem can be utilised, if available.
  - The system updates itself through interaction with the environment:
    * On-line/real-time: awake status.
    * Batch mode: sleep status/dreaming.
    * Evolution: Genetic Algorithms across generations.
- Fault tolerant:
  - Provide multiple paths for data to follow.
  - Allow to distribute the execution on multiple nodes, either local or remote.
  - Allow to set constraints each module's output must satisfy, and if they are not select another module.
- High performance / real-time:
  - Multiple modules can have the same purpose, with different output quality and computation time requirements.
  - When the output is required, provide the best one obtained so far.
- Small training sets / generalization:
  - Trainings sets cannot be too vast; use a size comparable to the one needed by a living organism/human in the same context.
  - Vectorization (or other transformations) of the training data to get more efficient learning.
- Extract explicit symbolic knowledge from the trained system for the I/O pairs that allow to do so.
- Energy efficient:
  - A Gating Network preselects which modules have to process the current input.
  - The other modules are idle, with no energy consumption. For most tasks, most of the network should be off.
    * E.g.: when we see an apple: 1) we do not involve our auditory cortex to predict what sound an apple may produce; 2) we do not involve parts of the visual cortex dedicated to the recognition of completely different objects.
  - Human brain:

        * $8.6 \cdot 10^{10}$ neurons, $2.5 \cdot 10^{-7} J$ of energy needed for a single action potential.
            - Assuming a single action potential per second per neuron, the total power consumption of the brain would be more than $20KW$ (without considering synaptic and dendrite transmission contribution).
            - The effective total power consumption is $12W$. This means that only roughly 0.06% of the brain should be active simultaneously in normal conditions.
            - These data would suggest to use event-based (instead of time-based) algorithms when simulating the nervous system.

- Extensible:
    - Plug-in architecture: modules can be added/removed at compile time and run time.
    - Each module can have its own commercial/closed or open source license. The infrastructure can be either closed or open source.

- Scalable:
    - The target hardware can range from mobile phones to workstations, and from peer-to-peer networks to clusters in the future.

3 Planned implementation

- Combined use of CPU and GPU.
- Minimize data transfer to/from GPU memory (performance).
- Split network elements into low- and high-priority groups, according to specific criteria. Give preferential GPU access to (in this order):
  1. High priority elements, even if used rarely.
  2. Most frequently used elements.
  3. Elements connected to others already processed by the GPU.
  4. Parametrise elements' memory swapping frequency for optimisation.
- Serial processing/CPU: use event lists to schedule operations (e.g. *spike event lists*).
- Use a common language to code for both CPU and GPU: C++ algorithms (with execution policy: `std::parallel::opencl`, `std::execution::unseq`, `std::execution::par_unseq`), SYCL, SyclParallelSTL.
- Consider using existing tools (or parts of) for the low level optimisation (e.g. TensorFlow).

4 Integration with other tools

- Other AI/NN/DL tools: e.g. *TensorFlow*, *IBM Watson*, ... where Neuro can become a module in these projects, or they can themselves be modules in Neuro.
- Databases: to store data instead of the filesystem; it would offer a higher level interface, the possibility to load a subset of the data through queries and to add/remove data dynamically as an independent process, automatic backups/redundancy/reliability, concurrent access by multiple instances of the Neuro tool (to perform multiple parallel training sessions from multiple executables with a shared memory).
- Graphics tools: *ParaView*, for analysis and visualization of n-dimensional data sets; *Tulip Software*, for graphs visualization and navigation.

5 Conclusions

This overview summarises my personal project plan and objectives. The main one is to develop a system that imitates the human brain, keeping in consideration its physiological constraints that are to a certain extent similar to the ones hardware and software also have; this implies using different information encoding strategies, first of all the *Spiking Neuron Model*. Another objective

is the separation of what the nervous system does at different abstraction levels, from high level thinking down to neurotransmitters release, and to only consider what has a direct impact on its capabilities, swapping the biological implementation details with software/hardware ones. Then the modularization of the system, allowing to develop progressively more complex units which can be used as basic building blocks for other modules, in an iterative development that imitates our knowledge acquisition processes, without the need to retrain on what already acquired.

6 References

- "Adaptive mixtures of local experts". R. A. Jacobs, M. I. Jordan, S. Nowlan, G. E. Hinton. Neural Computation, 3, 1-12, 1991.
- "Hierarchical mixtures of experts and the EM algorithm". M. I. Jordan, R. A. Jacobs. Neural Computation, 6, 181-214, 1994.
- "Spiking Neuron Models: Single Neurons, Populations, Plasticity". W. M. Kistler, W. Gerstner, Cambridge University Press 2002.
- "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer". G. E. Hinton et al. 2017.
- "A Theory of How Columns in the Neocortex Enable Learning the Structure of the World". Jeff Hawkins et al. 2017.
- "A cortical sparse distributed coding model linking mini- and macrocolumn-scale functionality". G. J. Rinkus, Frontiers in Neuroanatomy, 2010.

Document written with LyX / LaTeX.