FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

DEPARTMENT OF COMPUTER SCIENCE

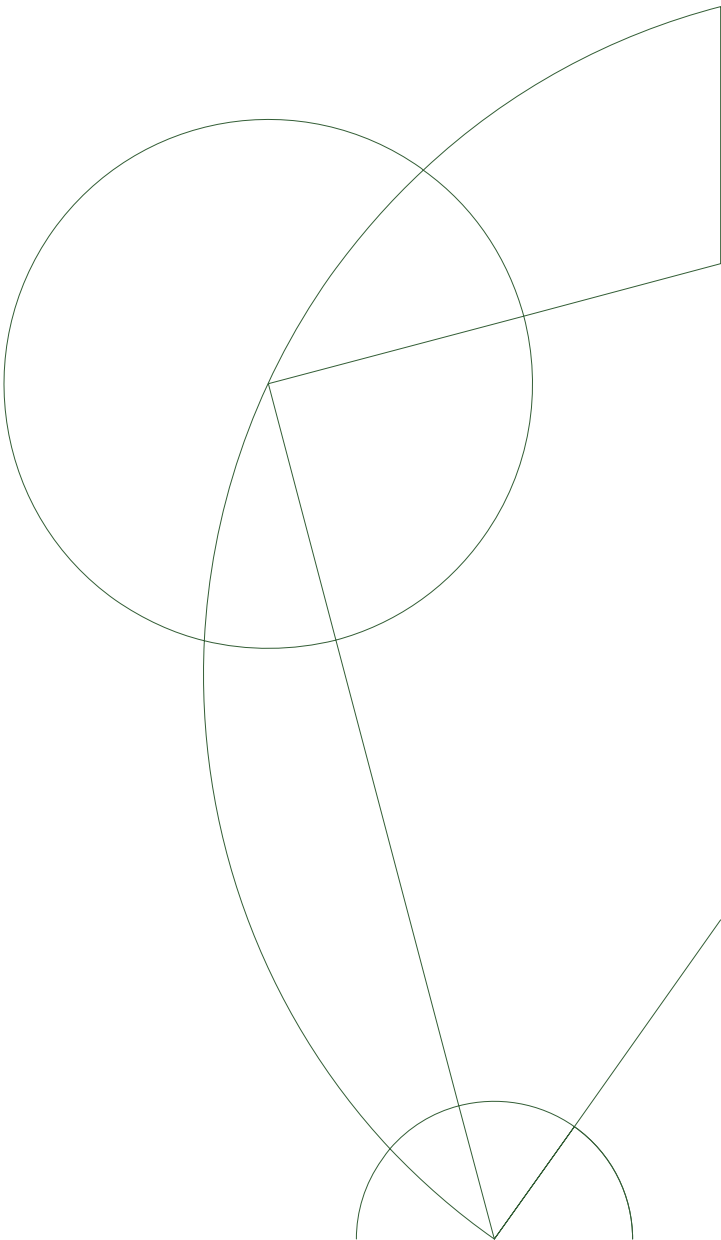

# Modelling learning systems

## A DSL for cognitive neuroscientists

Jens Egholm Pedersen <xtp778@alumni.ku.dk>

**Supervisor**

Martin Elsman <mael@di.ku.dk>



**Acknowledgements**

I would like to thank the Unit for Cognitive Neuroscience in Copenhagen for the many talks and assistance in the making of this paper. The patience of Nicolaj Daugaard and Jesper Mogensen in particular deserves high praise.

# Contents

# Chapter 1

# Introduction

In the past years machine learning has surpassed humans in some recognition tasks, and the development shows no signs of slowing down. These developments are however based on relatively old research on neural networks (Nilsson 2009; Russell and Norvig 2002). Newer investigation into rehabilitation and learning indicates that such networks alone cannot account for the same amount of learning that happens in the brain (Mogensen 2011; Block 2007; Russell and Norvig 2002; Moravec 1998; Dennett 2017). For that reason the breakthroughs in machine learning are hard to transfer to the domain of cognitive neuroscience.

As an attempt to remedy this, this project sets out to define a domain-specific language (DSL) that is capable of representing learning systems that resemble those from the domain of neuroscience.

The second part of the paper validates this DSL through the modelling of a small learning task. The benchmark will be written in Futhark and compiled to the OpenCL standard. The language abstraction however, allows it to be executed on any machine architecture.

The goal is for the DSL to explore a more accurate scientific representation of learning and learning concepts, serving as a more approachable simulation tool for the cognitive neurosciences.

## 1.1 Problem statement

Building on theories and concepts of the domain of cognitive neuroscience this paper examines the hypothesis that *the DSL presented in this paper can model meaningful machine learning tasks for the cognitive neurosciences, agnostic of the learning system implementation*. The paper will approach this in two steps:

1. Defining and implementing a DSL abstraction for the expression of learning tasks, based on the REF model from (Mogensen 2011).

2. Testing the DSL by expressing a learning task in a Krechevsky T-maze (Krechevsky 1932), backed by a traditional machine learning implementation in Futhark.

# Chapter 2

# Theoretical Foundations

This section accounts for the theoretical foundation of the paper and is divided into three parts. The first part concerns the broad topic of computation and learning in neural systems as seen from the perspective of computational neuroscience. By focusing on cognition, plasticity, learning and rehabilitation, it derives the necessary and sufficient language abstractions to capture the complexity of the domain. The second part introduces traditional machine learning from the perspective of computer science. These concepts will be applied in the validation phase of learning model abstractions in section 3.2. In the final part the theoretical and methodological background for the development of domain specific languages will be treated.

## 2.1 Learning in neural systems

> Activity-dependent synaptic plasticity is widely believed to be the basic phenomenon underlying learning and memory (Dayan and Abbot 2001).

Commonly referred to as *what fires together, wires together*, Hebbian learning suggests that synaptic connections from neuron $A$ to neuron $B$ are strengthened or weakened when neuron $A$ excites or inhibits the chance of firing neuron $B$ respectively (Dayan and Abbot 2001). Hebbian learning is believed to play a large part in the plastic nature of the brain, especially within learning and memory formation (Dayan and Abbot 2001; Johnston 2009; Robertson and Murre 1999).

### 2.1.1 Reorganisation of elementary functions

(Robertson and Murre 1999) studied patients during rehabilitation of brain damage and conjectured that learning — either when acquiring new information or recovering from lost information — occurs based on the structural changes induced by the Hebbian principle (Robertson and Murre 1999). They further concluded that the damaged brain areas regenerate themselves based on this principle (Robertson and Murre 1999). Mogensen refutes this point by claiming that, while there may be some synaptogenesis, the rehabilitation mainly occurs when other parts of the brain learns to take over the lost functions (Mogensen 2011).

Mogensen arrives at a theoretical framework which divides the brain into localized and highly specialised, basic information processing elementary functions (EF). These modular functions are contained in a *substructure* or *local circuit* within the brain (Mogensen 2011).

Multiple EFs interact to form algorithmic strategies (AS), established as a consequence of learning and experiencing (Mogensen 2011). An algorithmic strategy combines the capacity of multiple EFs into a single - and for the AS appropriate - response (Mogensen 2011; Mogensen 2012).

The EF and AS interplay to create what Mogensen dubbed *'surface phenomena'*, which manifests the behaviour of the system (Mogensen 2011). Surface phenomena are the product of applying an AS to a particular problem, and Mogensen hypothesised that a given AS is evaluated for every success or failure of the surface behaviour predicted by that AS (Mogensen 2011). Such evaluation either strengthens the AS's association with the given behaviour scenario, or weakens it, potentially starting a search for another AS to perform the task instead (Mogensen 2011). According to Mogensen these changes are controlled by a specialised AS dubbed the *Supervisory Attentional System* (SAS) from (Norman and Shallice 1986), manifested through neuroplastic changes in the synaptic connections within the SAS (Mogensen 2011).

*Behaviour* in the REF model is thus defined as the response of a single AS to a given task, that in turn consists of a number of EFs (Mogensen 2011; Mogensen 2012).

An elaboration to the REF model arrived in the form of the REFGEN model

(general reorganisation of elementary functions) (Mogensen and Overgaard 2017). The REFGEN model further explains the feedback mechanisms of the SAS to account for the 'learning' or adaptive feature of neural systems, by introducing two new concepts: the goal algorithmic strategy (GAS) and the comparator (Mogensen and Overgaard 2017; Mogensen 2012).

In this new framework the SAS is tasked with maintaining the current state of the system, while the GAS reflects the goal towards which it is desired to to move, for instance the exit condition in a maze (Mogensen and Overgaard 2017). For this to be useful a comparator is needed to constantly compare the SAS and GAS (the current state versus the goal), such as to select the optimal AS for the task at hand (Mogensen and Overgaard 2017). The feedback (back-propagation) from the actuation on the surrounding world is received by the comparator, who will assert influence on the SAS and GAS to better account for the new reality (Mogensen and Overgaard 2017).

## 2.2 Learning in machines

Systems that display the capability of learning (or progressively improving on a certain task) are historically associated with the field of machine learning. Machine learning goes back to the 1950s, and is tightly coupled with artificial intelligence — the idea of creating intelligence in machines (Nilsson 2009; Russell and Norvig 2002). It is a large and active research field, and this is why this section will strictly focus on a brief introduction and motivation of the topic in the context of learning tasks. This will be followed by a more in-depth description of multilayered perceptron networks and progressive learning through stochastic gradient descent. Both techniques will be employed in the implementation and validation sections (sections 2.3 and 3.2).

### 2.2.1 Artificial intelligence and machine learning

Nilsson defines artificial intelligence as the "activity devoted to making machines intelligent", where *intelligence* is defined as the "quality that enables an entity to function appropriately and with foresight in its environment" (Nilsson 2009, p. 13). As Nilsson also points out, this covers a large range of sys-

tems such as recommender systems and image recognition tasks, which do not display higher cognitive functions (Nilsson 2009, p. 13). In the context of this paper, the above definition suffices to cover basic learning progression.

### 2.2.2 Learning in perceptron networks

Inspired by the biological brain, models for neural networks have been applied in computer science since the 1950s, and have become an integral part of machine learning (Nilsson 2009; Russell and Norvig 2002). Neurons are essentially functions that operate on some input and respond with some output (Russell and Norvig 2002). The inputs for a neuron are weighted to give different input channels - or input dimensions - varying significance. These weighted inputs arrive to an activation function that decides whether the neuron should 'fire' or not (Nilsson 2009). Sigmoid or hyperbolic tangent are popular activation functions for numerical predictions because of their steep logistic properties while retaining differentiability. Neural networks excel in their adaptability and have been applied to many domains with great success (Schmidhuber 2015; Russell and Norvig 2002; Pedersen 2018b).

Simple neural networks can be stacked in layers, where each neuron will assign weights to the input, to determine its significance for the activation function (Nilsson 2009). By tuning the weights of the neurons, such groupings can learn to fire on certain input patterns (Russell and Norvig 2002). However, this structure has proved to be brittle and difficult to train because the neurons do not retain their weights for long (Nilsson 2009; Russell and Norvig 2002). (Rumelhart, Hinton, and Williams 1988) suggested a method to avoid this instability by adjusting the weights of the neurons in retrospect with a method called back-propagation (Rumelhart, Hinton, and Williams 1988; Nilsson 2009). This optimisation is operationalised by a function that can describe the *loss* of efficiency in a network, and then adjust the weights correspondingly using gradient descent (Russell and Norvig 2002).

## 2.3 Language abstractions

A domain specific language (DSL) is a generalisation of a domain into a collection of tokens (Mernik, Heering, and Sloane 2005). A DSL generally offers more expressiveness and requires less programming expertise (Mernik, Heering, and Sloane 2005; Sestoft 2017). However, designing DSLs requires both deep domain knowledge and language development expertise and bad DSL design can become limiting and counterproductive (Mernik, Heering, and Sloane 2005; Sestoft 2017).

Compared to the development of general programming languages, DSL development is less standardised within the literature (Mernik, Heering, and Sloane 2005; Deurson and Klint 2002). However, there are strong recommendations to perform rigorous analyses of the target domain and extract precise features necessary for the DSL construction (Mernik, Heering, and Sloane 2005; Deurson and Klint 2002).

DSLs can be implemented either as a standalone language or embedded in a more general programming language (Mernik, Heering, and Sloane 2005). A standalone approach leverages the freedom of an independent syntax, but avoids the benefits of the environment and tools that normally follow a programming language (Mernik, Heering, and Sloane 2005; Sestoft 2017). Conversely, an embedded approach is more restricted with regards to syntax, but retains the infrastructure of the host language (Mernik, Heering, and Sloane 2005).[1]

Further, a DSL can either reuse the syntax and structure of other, older languages or completely invent new constructions (Mernik, Heering, and Sloane 2005; Deurson and Klint 2002). DSLs without any resemblance to previous languages and paradigms are difficult to develop and hard to memorise, especially for the user (Wile 2004).

The target for the DSL can, but does not have to be, executable (Mernik, Heering, and Sloane 2005). One common approach for executable DSLs is to generate code that can be compiled by a separate environment, thus promoting the reuse of code (Wile 2004).

[1] Mernik, Heering, and Sloane 2005 provide a detailed list of benefits and downsides for both approaches. See (Mernik, Heering, and Sloane 2005, pp. 330-331).

Lastly, it is important to note that the introduction of a DSL is not only introducing the language abstraction itself, but can impose up to five innovations: artifacts, language, tools, infrastructure and methodology (Wile 2004). For that reason it is important for the DSL to be as close to the original domain as possible, without introducing unnecessary overhead (Wile 2004; Mernik, Heering, and Sloane 2005).

# Chapter 3

# Volr: A DSL for learning systems

Following the advise of Mernik, Heering, and Sloane 2005 and Wile 2004, the theory of reorganisation of elementary functions from section 2.1.1, has been applied to the domain of machine learning as rigorously as possible. The resulting DSL has been dubbed `Volr`.

Before accounting for the features and design decision of the language, it is important to motivate the reasoning for constructing a DSL. Research within the domain of (cognitive) neuroscience is juggling increasingly complex models, and is to an increasing degree overlapping with the domain of computer science.[1] Researchers within cognitive sciences are rarely equipped with strong programming knowledge, and the rapid development of machine learning in recent years has made it hard to keep track of state-of-the-art frameworks.[2]

Another fact driving the decision to design a DSL is the particular need to build and extend specific features for cognitive neuroscience, such as neuron clusters (without layering) and lesions in the networks. These features would never enter regular machine learning libraries, because of their strong focus

[1] This claim is implicit in a number of articles from both domains, and explicit in more recent literature. From the perspective of computer science, see for instance (Nilsson 2009; Walter, Röhrbein, and Knoll 2015; Schmidhuber 2015; Russell and Norvig 2002). From the perspective of cognitive science, see (Hohwy 2009; Dennett 2017; Dayan and Abbot 2001; Thagard 2014).

[2] Within machine learning alone there have been a surge of new projects in the last few years. Of particular note are Google's Tensorflow (from 2015), Microsoft Cognitive Toolkit (from 2016), Python's Scikit-learn (from 2007) and Pytorch (from 2016).

on accuracy and performance (Nilsson 2009; Schmidhuber 2015).

## 3.1 Volr design decisions

Because of the requirement to design a few, but unique features, a standalone approach was employed. The syntax has been kept simple and resembles that of the YAML (YAML Ain't Markup Language, Evans 2011). This style was chosen for its simplicity and widespread, use along with its ability to express declarations.

The language is parsed in Haskell using the open-source parser-combinator library `megaparsec` (Karpov 2018), and is available on GitHub (Pedersen 2018c).

The DSL was required to be executable and to allow users to evaluate and analyse cognitive models. The only backend supporting this currently is implemented in the data-parallel array programming language Futhark (Henriksen et al. 2017), running on an OpenCL backend. The implementation uses perceptron networks with gradient descent backpropagation learning. The source code is available on GitHub (Pedersen 2018a).

The syntax of the DSL is described in EBNF notation and visualised in a railroad diagram available in appendix A.

## 3.2 Volr concepts

Transferring the concepts of the REF model to a DSL is a complex task that necessitates simplification. To limit the pitfalls of developing the DSL as warned by Mernik, Heering, and Sloane 2005, the present model has been developed in collaboration with the Unit for Cognitive Neuroscience in Copenhagen and the author of the REF model, J. Mogensen.

Three main concepts have been derived from the REF model: A stimulus which serves as the input for a given learning system, a strategy which contains a number of (elementary) functions and finally a response, which outputs the activations of one or more strategies. The DSL allows for multiple strategies, but only a single stimulus and response. Both strategies and responses can however, use any combination of stimuli and strategies as their input. This allows for a graph-like structure, which represents the algorithmic

strategies as described in the REF model (see section 2.1.1). Figure 3.1 shows an example of such a graph.

As shown in figure 3.1 stimulus, strategies and response are all associated with an integer that denotes their dimensionality. The dimensionality is necessary for several reasons: First, the network needs to learn from a given input, which is required to adhere to the dimensionality restriction. Second, the strategies can be combined to provide networks of different sizes, but the input for those strategies have to depend on the outputs of the previous layers, which requires them to combine the features. The dimensionality of the strategies entirely depends on the number of elementary functions (`functions`) associated with the strategy. Figure 3.1 shows an example of this. Here, a singly dimensional input is fed into two strategies with four and three EFs respectively. They are then combined into a 7-dimensional input, which is reduced to five in the last strategy, and forwarded to the response.

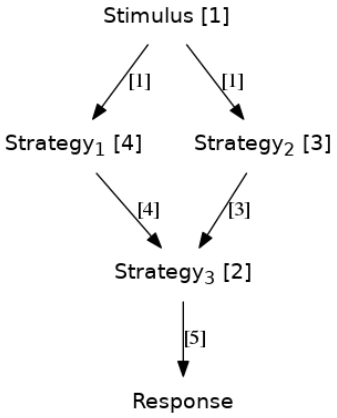



Figure 3.1: An example learning graph in Volr.

Stimuli and response sections further specify a file, from which the input data and expected output data is read. This is used to train the network according to the given data.

Responses also allow to specify learning rates (`learning_rate`) to adjust the learning rate of the underlying implementation.

A 3- and 4-layer network can be seen in appendix B.

# Chapter 4

# Case study: Krechevsky maze

This case study is based on the mazes built by Krechevsky in the 1930's

| | **3 layers** | **4 layers** |
|---|---|---|
| Runtime | 40.505ms | 49.137ms |
| Accuracy | 100% | 100% |

Table 4.1: Runtimes of volr networks in milliseconds, averaged over 100 runs.

(Krechevsky 1932). Krechevsky researched the behaviour of rats by guiding them through carefully constructed mazes that gave the rats clues in the form of shapes and colours of light (Krechevsky 1932). Each fork showed one or more of the features, leading either to success (reward) or failure (blocked path). To illustrate such a maze, imagine a bright lamp being consistently installed at paths leading to reward, while complete darkness consistently points to dead ends. Here the rats would quickly learn to follow the lit paths to claim their reward (Krechevsky 1932). For the cognitive neurosciences this experiment is interesting, because it is feasible to introspect each maze decision and compare the decisions of the virtual system with real rats.

Incidentally the mazes are easy to model because the features can be represented with simple boolean flags when they are present (`1`) or absent (`0`). Therefore, a simple maze with only T-junctions can be generated by showing the "rat" an array of input features and expecting it to make a choice: left or right.

## 4.1 Experiment result

The Krechevsky maze in the experiment is a simple two-feature maze with a single branching option, to give an immediate feedback for the learning network. For the sake of the experiment, the features in the maze are entirely consistent. A perfect network is thus expected to achieve a 100% accuracy.

Two networks were constructed to calculate the data: a 3-layer and 4-layer maze. Both use a sample size of 500.

The results are shown in table 4.1. Both networks achieve a 100% accuracy, which indicates that they are capable of learning the task at hand. Information about the execution and runtime environment can be found in appendix C.

# Chapter 5

# Discussion

This paper set out to examine whether a DSL can model meaningful machine learning tasks for the cognitive neurosciences. Building on relevant theory from cognitive science and computer science, a DSL, Volr, was presented. A backend executing to OpenCL was implemented and a Krechevsky-like maze was constructed and evaluated through two 3- and 4-layered neural networks with backpropagation and gradient descent learning.

The evaluation showed that Volr successfully modelled the Krechevsky-maze task with different network configurations, and with the expected accuracy.

The result is encouraging because it is easily scaleable to more complex mazes and other problems. The present study only examined a perfect maze where each feature was consistent. A possible extension of this would be to randomize the features and study the behaviour of the system. Similarly, this system could allow to support a form of "memory" where previous decisions are stored as *breadcrumbs*, that can assist the virtual rat in its decision.

With the concept of memory, the virtual model would be even closer to the biological model and its learning mechanism. This underlines the present work as an initial effort towards more elaborate modelling tools.

Reviewing the hypothesis two shortcomings became apparent during the evaluation: accurately representing the 2.1.1 model in the DSL and modelling the mechanisms of the biological substrate.

Notably two features from the 2.1.1 model were ignored in the paper: the GAS and the comparator. The GAS was partly implemented by the supervised learning strategy, in which the stochastic gradient descent learning continually improved the network towards the training data. But the model does not have control over the GAS and the mechanisms with which the network is improved. A more fitting approach to capture the GAS could be to implement

reinforcement learning, where the goal is more clearly stated. The comparator was omitted because it required the implementation of multiple and competing strategies. This is however easier to mitigate by simply constructing multiple models and choosing the best network, based on some criterion.

Another shortcoming of the approach is the Futhark implementation backed by traditional machine learning perceptron networks. These networks cannot accurately model neurons in the biological brain. The layered approach does not allow for the continuous feedback from neighbouring strategies, nor for the asynchronous workings of the neurons. A mediation of this could be to write a backend which supports such a clustered neuronal base, similar to the idea of the EF in the REF model. Neuromorphic hardware would be particularly interesting to examine here, because they are capable of simulating exactly these properties.

# Glossary

**artificial intelligence** Artificial intelligence (AI) covers the broad discipline in computer science that is concerned with replicating intelligent behaviour in computational systems. The exact definition is controversial for historical reasons (Nilsson 2009). 4, 13

**domain specific language** A DSL is a language used to model concepts from a certain domain. DSLs are usually simpler than more general programming languages in that they contain fewer concepts and less complex syntax. 2

**Futhark** A programming language geared towards performance in parallel environment such as graphics processors (GPUs). Futhark is a purely functional array language and is developed by HIPERFIT research center under the Department of Computer Science at the University of Copenhagen (DIKU). 1, 2

**machine learning** Machine learning is a sub-field within artificial intelligence that is concerned with developing systems that "progressively improves their performance on a certain task" (Wikipedia 2018). 4, 7

**OpenCL** An open standard for cross-platform parallel programming, which allows software to be executed on CPUs, GPUs or other processors or hardware accelerators. See `https://www.khronos.org/opencl/`. 1, 8

**REF** A model for rehabilitation in patients with brain lesions, developed by Mogensen 2011. An extension in the form of the REFGEN model was developed by Mogensen and Overgaard 2017. 3, 8, 9, 12

# Appendix A

# Volr EBNF

A railroad diagram of the below EBNF is available online at `https://github.com/Jegp/volr-report/blob/master/railroad.png`.

```
model = ( stimulus | strategy | response )
      , { [ space ] , "," , [ space ] , ( stimulus | strategy | response ) };

stimulus = "stimulus" , [ space ] , name , [ space ] , dimensionality
         , [ space ] , "file:" , [ space ] , name;
strategy = "strategy" , [ space ] , name , [ space ] , input
         , [ space ] , "functions:" , [ space ] , integer;
response = "response" , [ space ] , dimensionality , [ space ] , input
         , [ space ] , "file:" , [ space ] , name
         , [ space ] , "learning_rate:" , [ space ] , number;

input = "from" , [ space ] , name , { [ space ] , name };
dimensionality = "[" , [ space ] , integer , [ space ] , "]";

name list = name , { [ space ] , "," , name };
name = letter , { letter | digit };
letter = ? non-whitespace Unicode character ?;
space = space character , { space character };
space character = ? white space character ?;

number = integer , ["." , digit , {digit}];
integer = ["-"] , digit , {digit};
digit = "0" | "1" | "2" | "3" | "4"
      | "5" | "6" | "7" | "8" | "9";
```

# Appendix B

# 3- and 4-layer learning networks in Volr

```
stimulus input [2]
  file: examples/maze_x500.txt
strategy b1 from input
  functions: 30
strategy b2 from b1
  functions: 10
response from b2
  file: examples/maze_y500.txt
  learning_rate: 0.5
```

---

```
stimulus input [2]
  file: examples/maze_x500.txt
strategy b1 from input
  functions: 30
strategy b2 from b1
  functions: 10
strategy b3 from b2
  functions: 2
response from b3
  file: examples/maze_y500.txt
  learning_rate: 0.5
```

# Appendix C

# Benchmark informations

The benchmark was executed on an Asus X550VX with an Intel i7 6700HQ 2.6GHz quad-core processor, 8 GB of DDR4 2133 MHz SDRAM and an NVIDIA GeForce GTX 950M (640 CUDA cores at 914MHz).

The benchmark was run by the following command (standing in the directory of the volr project):

```
volr -f examples/4layer_500.volr
```

Runtimes was extracted using a specific `-t` flag in the Futhark runtime, and re-run 100 times with the Futhark `-r` flag. The above command produces binaries in the `bin` folder, allowing us to execute 100 iterations and put the results into the `times.txt` file:

```
cat examples/maze_x500.txt examples/maze_y500.txt \
  | bin/futhark-model -r 100 -t times.txt
```

# Bibliography


Block, Ned (2007). "Consciousness, accessibility, and the mesh between psychology and neuroscience". In: *Behavioral and Brain Sciences* 30.5-6, pp. 481–499. DOI: 10.1017/S0140525X07002786.

Dayan, Peter and L. F. Abbot (2001). *Theoretical neuroscience - Computational and Mathematical Modeling of Neural Systems*. MIT Press. ISBN: 978-0-262-04199-7.

Dennett, Daniel C. (2017). *From bacteria to Bach and back*. ISBN: 978-0-393-24207-2.

Deurson, Arie von and Paul Klint (2002). In: *Journal of Computing and Information Technology* 1, pp. 1–17.

Evans, Clark C. (2011). *The Official YAML Web site*. [Online; accessed 31-March-2018]. URL: %5Curl%7Bhttp://yaml.org/%7D.

Henriksen, Troels et al. (2017). "Futhark: purely functional GPU-programming with nested parallelism and in-place array updates". In: *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 556–571.

Hohwy, Jakob (2009). "The Neural Correlates of Consciousness: New Experimental Approaches Needed?" In: *Consciousness and Cognition* 18.2, pp. 428–438.

Johnston, Michael V. (2009). "Plasticity in the developing brain: Implications for rehabilitation". In: *Developmental Disabilities Research Reviews* 15.2, pp. 94–101. ISSN: 1940-5529. DOI: 10.1002/ddrr.64. URL: http://dx.doi.org/10.1002/ddrr.64.

Karpov, Mark (2018). *Megaparsec on GitHub*. [Online; accessed 31-March-2018]. URL: %5Curl%7Bhttps://github.com/mrkkrp/megaparsec%7D.

Krechevsky, I (1932). "Hypotheses in Rats". In: 39, pp. 516–532.

Mernik, Marjan, Jan Heering, and Anthony M. Sloane (2005). "When and How to Develop Domain-specific Languages". In: *ACM Comput. Surv.* 37.4, pp. 316–

344. ISSN: 0360-0300. DOI: 10.1145/1118890.1118892. URL: http://doi.acm.org/10.1145/1118890.1118892.

Mogensen, J. (2011). "Reorganization of the Injured Brain: Implications for Studies of the Neural Substrate of Cognition". In: *Frontiers in Psychology* 2, p. 7. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2011.00007. URL: https://www.frontiersin.org/article/10.3389/fpsyg.2011.00007.

— (2012). "Reorganization of Elementary Functions (REF) after brain injury: implications for the therapeutic interventions and prognosis of brain injured patients suffering cognitive impairments". In: *Brain Damage: Causes, Management and Prognosis* 1. Ed. by A. J. Schäfer and J. Müller, pp. 1–40.

Mogensen, J. and M. Overgaard (2017). "Reorganization of the Connectivity between Elementary Functions – A Model Relating Conscious States to Neural Connections". In: *Frontiers in Psychology* 8, p. 625. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2017.00625. URL: https://www.frontiersin.org/article/10.3389/fpsyg.2017.00625.

Moravec, Hans (1998). "When will computer hardware match the human brain". In: *Journal of Transhumanism* 1.

Nilsson, Nils J. (2009). *The Quest for Artificial Intelligence*. 1st. New York, NY, USA: Cambridge University Press. ISBN: 9780521122931.

Norman, Donald and Tim Shallice (1986). *Attention to Action*.

Pedersen, Jens E. (2018a). *Futhark backend for Volr on GitHub*. [Online; accessed 31-March-2018]. URL: %5Curl%7Bhttps://github.com/volr/futhark-backend%7D.

— (2018b). *Sentiment analysis with linear regression and LSTM*. URL: %5Curl%7Bhttps://github.com/Jegp/langprocexam/blob/master/report/report.pdf%7D.

— (2018c). *Volr language implementation*. [Online; accessed 31-March-2018]. URL: %5Curl%7Bhttps://github.com/volr/lang%7D.

Robertson, Ian H. and Jaap M. J. Murre (1999). "Rehabilitation of Brain Damage: Brain Plasticity and Principles of Guided Recovery". In: *Psychological Bulletin* 125.5, pp. 544–575.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1988). "Neurocomputing: Foundations of Research". In: ed. by James A. Anderson and Edward Rosenfeld. Cambridge, MA, USA: MIT Press. Chap. Learning Representations by Back-propagating Errors, pp. 696–699. ISBN: 0-262-01097-6. URL: http://dl.acm.org/citation.cfm?id=65669.104451.

Russell, Stuart J. and Peter Norvig (2002). *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall. ISBN: 0137903952. URL: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20%5C&path=ASIN/0137903952.

Schmidhuber, J. (2015). "Deep Learning in Neural Networks: An Overview". In: *Neural Networks* 61. Published online 2014; based on TR arXiv:1404.7828 [cs.NE], pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

Sestoft, Peter (2017). *Programming Language Concepts*. Springer International Publishing. ISBN: 978-3-319-60789-4. DOI: 10.1007/978-3-319-60789-4.

Thagard, Paul (2014). "Cognitive Science". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Fall 2014. Metaphysics Research Lab, Stanford University.

Walter, Florian, Florian Röhrbein, and Alois Knoll (2015). "Neuromorphic implementations of neurobiological learning algorithms for spiking neural networks". In: *Neural Networks* 72.Supplement C. Neurobiologically Inspired Robotics: Enhanced Autonomy through Neuromorphic Cognition, pp. 152–167. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2015.07.004. URL: http://www.sciencedirect.com/science/article/pii/S0893608015001410.

Wikipedia (2018). *Machine learning — Wikipedia, The Free Encyclopedia*. [Online; accessed 31-March-2018]. URL: %5Curl%7Bhttps://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=833013504%7D.

Wile, David (2004). "Lessons learned from real DSL experiments". In: *Science of Computer Programming* 51, pp. 265–290.