FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

DEPARTMENT OF COMPUTER SCIENCE

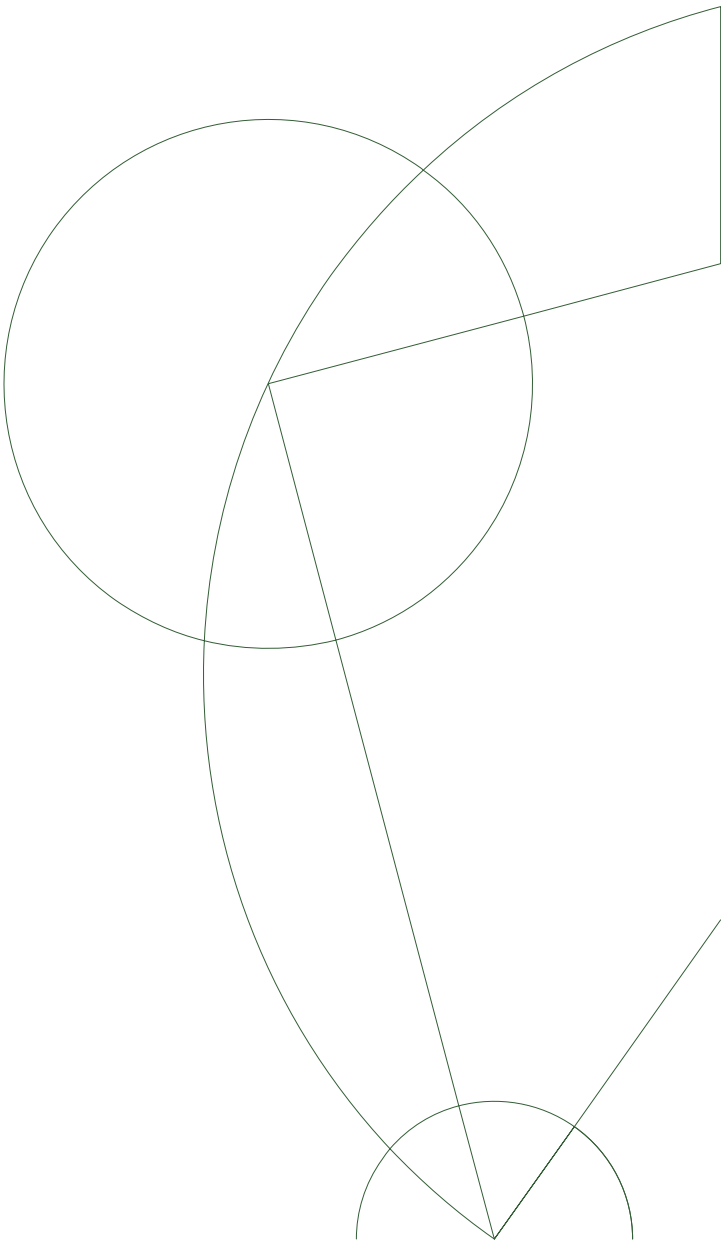

# Modelling learning systems

## A DSL for cognitive neuroscientists

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

**Supervisor**

Martin Elsman <mael@di.ku.dk>



# 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 resembles those from the domain of neuroscience.

The latter 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

# Theory

This section accounts for the theoretical foundation of 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 2.3. In the final part the theoretical background for linguistic abstractions and the construction 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 — whether when the brain acquires new information or recovers 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 2011) 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 2011) 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 et, 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 (that, in turn, consists of a number of EFs) to a given task (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 machine (Nilsson 2009; Russell and Norvig 2002). It is a large and active research field, 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 2.3).

### 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 does 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 integral part of machine learning (Nilsson 2009; Pedersen 2018; 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 long been applied to the field of language processing (Jurafsky and Martin 2000; Russell and Norvig 2002).

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 have proved 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). DSL generally offers more expressiveness and require less programming expertise (Mernik, Heering, and Sloane 2005; Sestoft 2017). However, designing DSLs requires both deep domain knowledge and language development expertise (Mernik, Heering, and Sloane 2005). Bad DSL design can become limiting and counterproductive (Mernik, Heering, and Sloane 2005).

Contrary to the development of general programming languages, DSL development is less standardised within the litterature (Mernik, Heering, and Sloane 2005; Deurson and Klint 2002). However, there are strong recommendations to perform rigorous analysis of the target domain and extract precise features necessary for the DSL construction (**Deurson2002**; Mernik, Heering, and Sloane 2005).

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 follows a programming language (Mernik, Heering, and Sloane 2005; Sestoft 2017). Conversely, an embedded approach will be more restricted with regards to syntax, but retains the infrastructure of the host language (Mernik, Heering, and Sloane 2005).[1]

Futher, a DSL can either exploit the features of previous languages os invent new constructions (Mernik, Heering, and Sloane 2005; Deurson and Klint

[1]Mernik, Heering, and Sloane 2005 lists a number of benefits and downsides for either approaches. They are too detailed to be reported here, but see (Mernik, Heering, and Sloane 2005, pp. 330-331).

2002).

# Chapter 3

# Volr: A DSL for learning systems

Embedded vs. standalone

Futhark: ("Futhark: purely functional GPU-programming with nested parallelism and in-place array updates" 2017)

OpenCL

# Chapter 4

# Case study: Krechevsky maze

As pointed out by Mernik, Heering, and Sloane 2005 quantitative evaluation of a DSL is hard, which is why this section will....

# Chapter 5

# Discussion

## 5.1 Conclusion

## 5.2 Future work

# 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, 9

**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

**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, 7

# 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.

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

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.

Jurafsky, Daniel and James H. Martin (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR. ISBN: 0130950696.

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.

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

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

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.

# Appendix A

# Volr EBNF

```
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 A

# Learning network with two features in Volr

```
stimulus input [2]
  file: examples/maze_x100.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_y100.txt
  learning_rate: 0.5
```