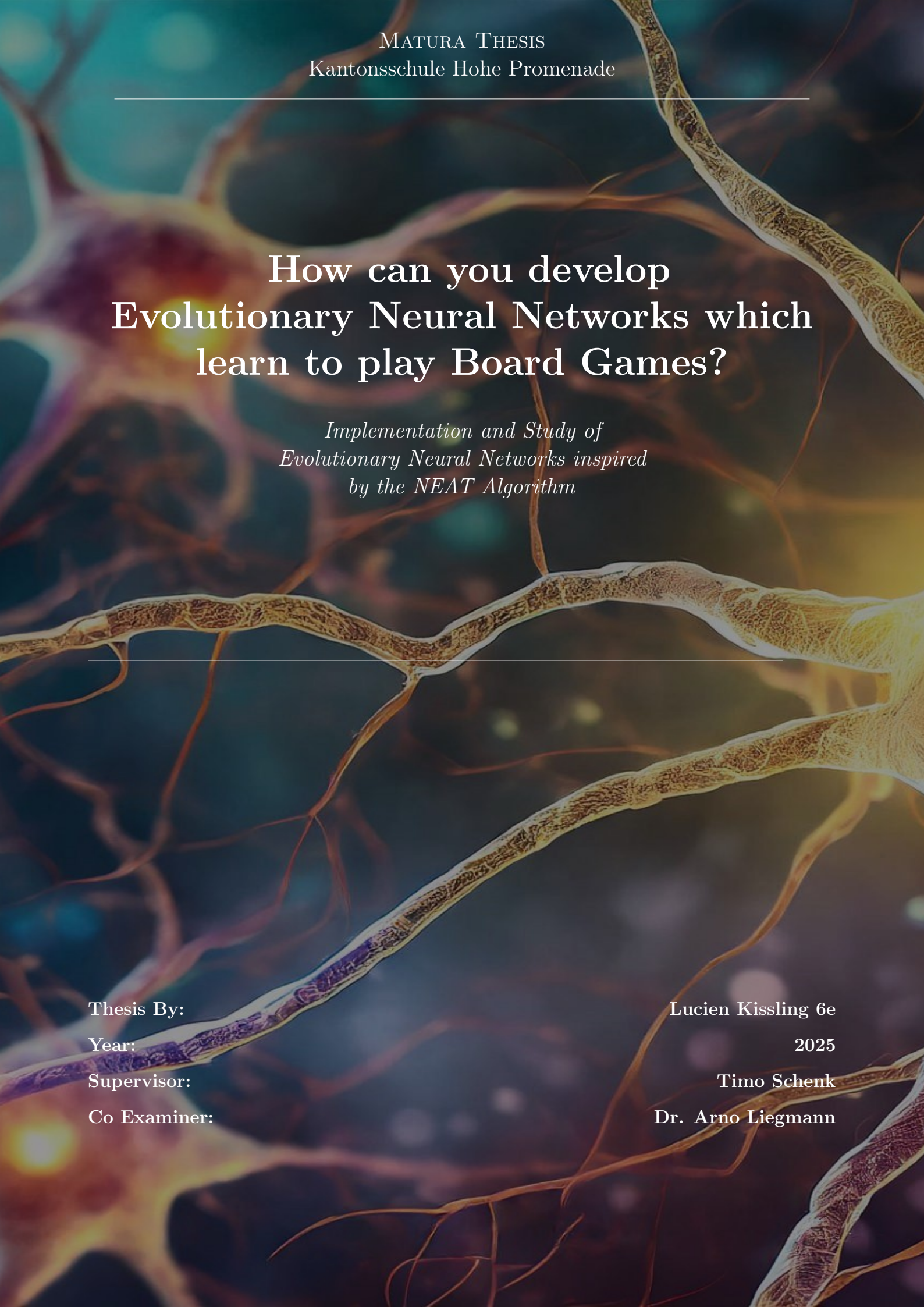
Matura Thesis
Kantonsschule Hohe Promenade

How can you develop Evolutionary Neural Networks which learn to play Board Games?

Implementation and Study of Evolutionary Neural Networks inspired by the NEAT Algorithm



Thesis By: Lucien Kissling 6e

Year: 2025

Supervisor: Timo Schenk

Co Examiner: Dr. Arno Liegmann

Contents

Chapter 1

Introduction

1.1 Preface

Ever since I got into Computer Science a few years ago, I was fascinated by the Idea of Algorithms that solved various Problems. Therefore, I participated in the SOI (Swiss Olympiad in Informatics) where we were taught everything about developing and programming Algorithms and their Data Structures.

In recent years although, a new field of Computer Science has gained a lot of attention, where those Algorithms are not programmed by humans, but evolved by a computer. This field called Machine Learning immediately got my excitement and two years ago a friend of mine and I had our first practical experience with it. We developed a simple Neural Network, which helped us predict the color of a lego brick in front of a color sensor based on the RGB values in various lighting conditions.

A Neural Network (NN) forms the basis of most Machine Learning Models and I will therefore explain it in much detail in the following Chapters. In simple terms, a NN is a strongly simplified artificial model of the human brain as a NN consists of an interconnected web of Neurons through which information flows and gets computed.

Although the NN we developed two years ago already learned on its self, we still had to provide data for it to learn from. This meant that we had to manually scan the RGB values of the lego bricks and then label them with the color they represented. The aim of my Matura Thesis therefore is to take the idea of self learning a step further by developing Neural Networks, which don't need this kind of data with solutions predefined by humans.

1.2 Thesis Statement

As mentioned in the previous Section, this Thesis will explore the field of Neural Networks (NNs) that learn without data provided by humans, which is called unsupervised learning. The solution this Thesis will focus on are Evolutionary Neural Networks (ENNs), a combination of Neural Networks and Genetic Algorithms.

In this Thesis, I develop my own simplified implementation ENNs and then train them on Board Games to see how well it can learn to play them using different Parameters and Features. My approach draws inspiration from the NEAT Algorithm developed by Kenneth O. Stanley and Risto Miikkulainen in 2002, where the NNs start minimally in the first generation and then develop complexity over time.[1] The first game I will train the ENNs on is Nim, a simple game where two players take turns removing matches from different stacks.

In specific, this Thesis aims to answer following questions:

- How can you develop Evolutionary Neural Networks (ENNs) that learn to play Board Games?
- How do different parameters and features of ENNs affect the learning process?

[1] Stanley and Miikkulainen 2002, p.105-106.

- How does this Implementation of ENNs compare to other Machine Learning Algorithms?

Chapter 2

Background

2.1 Neural Networks (NNs)

As already touched on in the Preface(1.1), an artificial neural network (ANN or NN) is a mathematical model for data processing, initially inspired by the structure of the brain[1]. Therefore, we will first have a brief look at the functioning of a brain.

2.1.1 Neurons in the Brain

Inside the brain, around 86 million neurons[2] form connections to each other through which they activate other neurons. In a neuron, the signals of connected neurons add up and when they reach a certain threshold, the neuron is activated and fires a signal to its own connections[3]. The neuron then resets after a certain amount of cooldown time. With this web of neurons inside the brain, animals can process information from nerve signals from the body and output them again as nerve signals instructing the body.

2.1.2 Feed Forward Neural Networks (FNNs)

Now how do we apply those ideas about Neural Networks learned from the biology of a brain to a program that runs on a computer? The first step is to simplify the chaos of neurons in the brain into layers of neurons. We get an input layer, an output layer and optional so-called hidden layers in between. As a next step, in each layer, we connect its neurons to neurons of following layers, typically exclusively to neurons of the next layer.
Now, the structure of our network looks something like this:

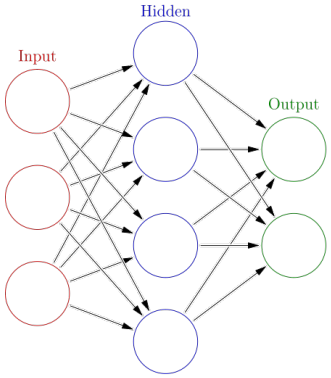


Figure 2.1: A simple Feed Forward Neural Network (FNN) with three layers: Input, Hidden and Output.[5]

The Structure of a NN is also called Topology.

[1] Chandra 2022.
[2] Caruso 2023.
[3] Newman 2023.

Neural Network Topology. *The Topology of a Neural Network is its distinct arrangement of neurons, layers and the connections between the neurons.*

For the neural network to perform functions, we also need to assign a weight to all the connections. This weight value ranges from -1 to 1 and can be thought of as the strength of a connection between neurons.
In terms of computer science, this structure now resembles a directed, weighted graph, which is why we will call the neurons nodes and their connections edges from now on.

Now let's see, how information gets computed by a neural network by using the example of a computer vision NN, that recognizes digits on a black/white image: First, we need a way to encode the information into values for the nodes of the input layer, in our example the Brightness values of the single pixels. We assign these values to the nodes of the input layer.
Then, for each node of the input layer, we look for all edges connected to that node. For each of these edges we multiply its weight by the value of the input node and add the result to the other node connected by the edge.
After iterating through all the nodes of the one layer, we move the next layer. Now, the value of the nodes in this layer is the sum accumulated by the value of all connected nodes multiplied the weight of that connection:

$$v_x = \sum_{i=0}^{N} v_i * w_i \tag{2.1}$$

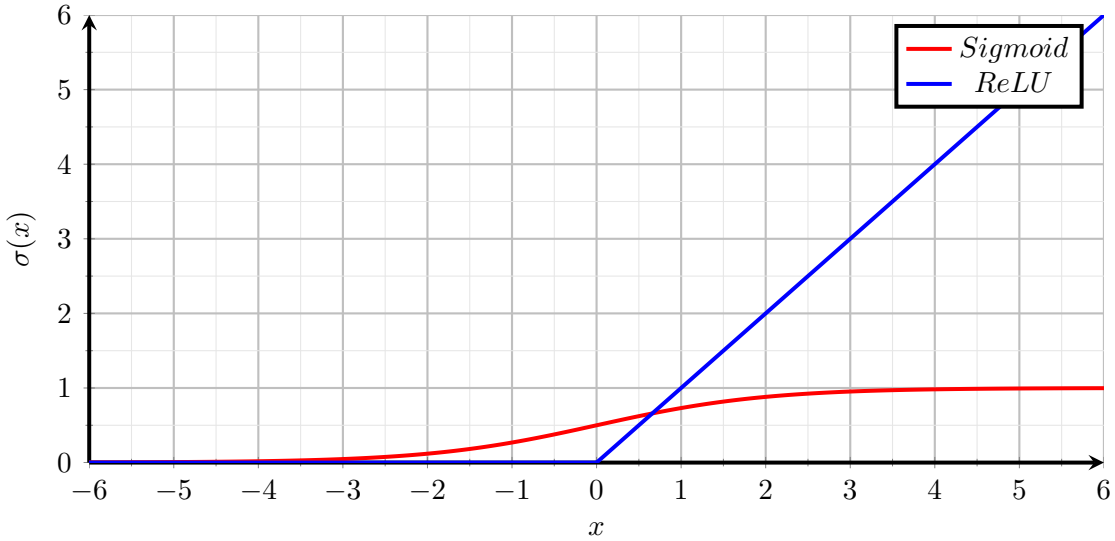
Where:

- N *is the number of connected nodes*
- v_x *is the value of the current node*
- v_i *is the value of the current node*
- w_i *is the weight of the edge connecting the current node to node* i

Additionally, we can add a bias value r_x ranging from -1 to 1 onto the value of the nodes. Afterward, an activation function is applied to the value of the nodes to fit the value of the node inside a preferred range. This activation function can also be thought of as the threshold of stimulation for a neuron to fire. Two examples for activation functions are:

- Sigmoid Function: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Rectified Linear Unit (ReLU): $\sigma(x) = \max 0, x$

Plot of the activation functions:



The final function for the value of one node is therefore:

$$v_x = \sigma\left(\sum_{i=0}^{N} v_i * w_i + r_x\right) \tag{2.2}$$

Where (additionally to equation 2.1):

- r_x *is the bias of the node*
- σ *is the activation function*

Now this value v_x can be used in upcoming layers. When we reach the final layer after iterating through all the nodes and layers of the NN, we can read out its computed output. This output again is encoded in node values like the input and therefore needs to be decoded for the result. In the example of a digit detecting NN, we could encode the output with 10 output nodes, each representing one digit. The result could be decoded by using the output node with the highest value as result. This kind of output encoding, where all possible result get their own node is called One-hot encoding. With this kind of output encoding, the values of the output nodes can be interpreted as a certainty value for a specific result to be correct.

2.1.3 Remark: Functioning of NNs

Now that we have established how a NN works, it is important to understand why this kind of algorithm is revolutionary to computer science.

An algorithm is a set of functions applied in a certain order onto an input to receive a result. Traditionally, this algorithm was programmed with the help of mathematical operations, logic functions, loops, system functions and data structures, which are then converted into binary code for the processor. Of course, this also applies in the context of NNs, however, there is also a third layer of abstraction on top, which simulates the functioning of a brain with neurons. Since the traditional algorithm only enables the neurons in the NN, the third neuron-based layer is what computes the actual function of the NN. Therefore, NNs resemble more the functioning of a brain than a traditional algorithm. This difference results in implications for the functioning of an NN: Traditional algorithms represent actual mathematical calculations and are therefore deterministic. With NNs, the algorithm is based on different parameters for nodes and edges, which make it an unpredictable blackbox. Additionally, NNs don't represent actual functions but try to approximate them instead. This means it is hard to prove a neural network to be always accurate and is the reason why we often call the result of NNs its prediction.

2.1.4 An Example of NN learning: Backpropagation

We already know how a NN can make a prediction for a given input with the help of parameters, which encode for the structure and weights/biases of the NN.But how do we find such parameters for a given problem? The answer is to use machine learning, which trains the model to perform a certain task with the help of training data.

Machine Learning. *Machine Learning is a subset of artificial intelligence that involves the use of algorithms and statistical techniques to optimize artificially intelligent models for a given problem.*

One machine learning algorithm for NNs is called backpropagation, which is one of the simplest and most efficient ways to train a NN.

Backpropagation. *The backpropagation algorithm is an algorithm that can optimize parameters for weights and biases of a NN within a fixed topology for a certain problem with the help of labeled data.*

Backpropagation uses a training and test set containing unlabeled data for a problem. But what does a test set with unlabeled data mean in the context of machine learning?

(Un)labeled Data. *Labeled data for a machine learning problem is a set of data with sample problems and the respective solutions. Unlabeled data instead only contains the sample problems.*

The Machine Learning Process starts with an NN with fixed topology and random weights/biases. Then, you give the NN the training problems, whose predictions will be random at first. However, with the help of the corresponding solutions, you can refine the predictions of the NN by adjusting the parameters in a way that would let the NN make the correct prediction for the given problem. This is done by starting from the prediction of a certain problem its solution and propagating back through the whole NN until reaching the input layer. More specifically, you compute a cost function of The NN shouldn't be adapted only to a single problem but should be able to make accurate predictions for the whole training set as well as for unknown problems. To therefore prevent overcorrecting a NN for a single problem, we have to factor in a learning rate significantly smaller than 1 onto our correction. No we can train the NN on the whole training set for many generations, until the NN starts to make accurate predictions for the whole. To test the performance on unknown problems we can use a separate test set, which the NN wasn't trained on.

2.2 Evolutionary Computation (EC) & Genetic Algorithms (GAs)

Now we will have a look at the Machine Learning Technique, this Thesis focuses on. Again, we will draw inspiration from Nature:

Evolutionary Computation. *Evolutionary Computation (EC) is an Algorithm that optimizes a set of parameters for a problem with the help of natural selection.*

One scenario for EC might be for example if you have a large set of data points of an unknown polynomial function with noise and outliers. If you want to find the underlying polynomial function, you could then employ Evolutionary Competition to find the best fitting parameters for such polynomial. So how do you find these parameters?

You start with an initial population of agents with a random set of parameters. For each generation of your population you will then repeat following steps:

- Fitness: First, you need to find out how good each agent (and its parameters) performs in the function they try to optimize for. The fitness evaluation can be done with an objective cost function or a competition between the agents, which evaluates their relative performances. In our example, the cost function might be the sum of the absolute differences between the data points and the output of the polynomial function.
- Selection: As a next step, you rank the agents performances to then pick the ones that have performed the best.
- Reproduction: The best agents pick is the part of the population that survives. These agents will then be replicated by copying their parameters (asexual reproduction) of by merging parameters from different agents (sexual reproduction).
- Mutation: The agents parameters will then be mutated by either completely overwriting certain parameters with new random parameters or by shifting the existing parameters by a random number.

After a certain number of generations, you will then have the best performing set of parameters for a given problem. There is also one popular addition to EC, which draws inspiration from nature again:

Genetic Algorithm. *A Genetic Algorithm (GA) is an implementation of EC that uses a genetic representation of the parameters.*

Its main idea is to have genes, that only encode indirectly for the parameters, and that can be on and off.

2.3 Evolutionary Neural Networks (ENNs)

After having covered all the basics about NNs and EC, we can now combine those concepts to create the algorithm, that this thesis is about.

> **Evolutionary Neural Networks.** *Evolutionary Neural Networks (ENNs) are Neural Networks that use Evolutionary Computation to optimize for its parameters for the NNs weights/biases and its topology.*

ENNs are useful for complex machine learning problems and also work for unlabeled training data as long as there is a fitness function. Let's bring our knowledge about EC into the context of Neural Networks: The parameters ENNs encode for are for the weights, biases, nodes and edges of a Neural Network. You could either directly encode the NNs parameters as a graph or indirectly as genes, which would make it a Genetic Algorithm. ENNs still start with a random population of agents that are evaluated and selected by a fitness function. Then they replicated (sexually or asexually) and finally mutated in following ways:

1. Change weights/biases: A new random value is set for the weight or bias of a random existing edge or node.
2. Shift weights/biases: The value for an existing weight or bias of a random existing edge or node is altered by a random change but is kept close to the old value.
3. Add edges: A new edge is added between two random unconnected nodes.
4. Add nodes: A new node is added in between of a random existing edge. The old edge is removed and two new edges with random weight are added between the new node and the two other nodes each.
5. Remove nodes/edges: A random node or edge in a way that doesn't cut off the input from the output layer.

2.4 Drawing Inspiration from the NEAT-Apporach

As already stated in the Thesis Statement(1.2), Neuroevolution of Augmenting Topologies is a ENN machine learning algorithm by Kenneth O. Stanley and Risto Miikkulainen in 2002[6]. One of its main innovations is that the initial population starts with NNs of lowest complexity and only increase topological complexity as its useful for the problem. In specific, you start with NNs that have only the nodes of the input layer connected to the nodes of the output layer. This should result in more efficient NNs for the problem, as the complexity is only increased when it improves the NNs performance. The NEAT algorithm therefore only needs the mutations 1. to 4. showed in the last chapter(2.3) and doesn't need to remove complexity (mutation 5.)as it is already minimal.

2.5 Games

Let's also have a quick look at the games, this Thesis will train the ENNs on.

2.5.1 Simple Nim

This game is a simplified version of Nim, where two players take turns removing an arbitrary amount of matches from a single stack with some number of matches. The player who removes the last match loses. Therefore, the winning strategy simply is to remove all matches from the stack but one, which forces the opponent to remove the last match, which makes them lose.

[6]Stanley and Miikkulainen 2002, p.105-106.

2.5.2 Nim

This game works similarly to Simple Nim (described in the last Section(2.5.1)) with the difference that there are multiple stacks with matches. Now, the player who removes the last match from the last unemptied stack loses. You can read up the winning strategy for Nim on Wikipedia[7]. The winning strategy is much more complex, which forces the ENNs to learn a more complex strategy.

2.6 Related Work

[7] contributors 2023.

Chapter 3

Building my ENN

3.1 Basic Implementation

Now, we will have a detailed look at how I used the concepts described in the last Chapter(1) to build an Evolutionary Neural Network that plays board games. First, I will explain the general Structure of my Code:

3.1.1 Project Overview

As already mentioned in the Thesis Statement(1.2), this Project is written in the Rust Programming Language. If you don't understand the syntax of the code snippets, you can consult the online "The Rust Programming Language" booklet under: https://doc.rust-lang.org/book/

My Codebase is divided into different modules as well as a bin folder with main files, one of which has to be selected to be executed at the beginning. The other modules are divided into one module with the ENN algorithm, the other modules handle the different games, the ENNs learn to play. The ENN module is divided into two files:

- agent.rs: This file handles the NNs and the mutations on the NNs
- population.rs: This file handles the natural selection process and the data saving

The game modules provide the problems for the NNs handle the predictions of the NNs and finds the new game state after a move. It might also include an objective performance evaluation function to measure how well the NNs perform. I will now explain all the functions bottom up, starting with the Neural Network. I work with Rusts Structs which are similar to classes in other programming languages.

3.1.2 Neural Network

The struct *NeurualNetwork* (defined in agents.rs) most importantly contains a two-dimensional vector of nodes, which encodes all the information of the NN:

```
1    pub struct NeuralNetwork {
2        [...] // redundant side data about the NN
3        pub nodes: Vec<Vec<Node>>,
4    }
```

The Vector inside (`Vec<Node>`) represents a layer of the NN and the outside Vector (`Vec<Vec<Node>>`) contains all layers of the NN. A *Node* contains its bias and a vector for the incoming and outgoing edges:

```
1    pub struct Node {
2        pub bias: f64,
3        //edges stored in an adjacency list
4        pub incoming_edges: Vec<Edge>,
5        pub outgoing_edges: Vec<Edge>,
6    }
```

An *Edge* contains the weight of the edge and its input/output node:

```rust
pub struct Edge {
    input: [usize; 2],
    out: [usize; 2],
    weight: f64,
}
```

The most important functions of the *NeuralNetwork* struct are:

- `new(input_nodes: usize, output_nodes: usize) -> Self {}`, which initializes the NN with all input and output nodes connected to each other with random weights and biases.
- `predict(&self, input: Vec<f64>) -> Vec<f64>`, which computes the output of the NN for a given input with forward propagation algorithm described in the last Chapter(2.1.2).
- `new(input_nodes: usize, output_nodes: usize) -> Self {}`, which computes the neuron activation described in the last Chapter(2.1.2). I use a variant of the ReLU function, since the linearity for inputs above 0 of the ReLU function allows for more variety of neuron activation, which makes the NN more flexible and efficient. The variant I use is the ELU function, which additionally tackles the problem of dead neurons in ReLU by allowing for negative values.
- All mutation functions for the NN, which we will need in the next Subsection(3.1.3).

3.1.3 Mutation

The mutation of the NNs is handled in the *Agent* struct (defined in agent.rs), which includes a NN, its fitness and its Rank:

```rust
pub struct Agent {
    pub nn: NeuralNetwork,
    pub fitness: f64,
    pub rank: isize,
}
```

The function `mutate(&mut self, mutations: usize) -> Self {}` has a number of mutations as a parameter which it applies to the NN of the Agent. For each mutation, it randomly selects one of the following mutations: Change weights/biases, Shift weights/biases, Add edges, Add nodes. All of these mutations have already been described in the last Chapter(2.3), although there are some implementation details to mention:

- oh actually that's kind of stupid, gotta change something

3.1.4 Natural Selection

3.1.5 Nim

3.1.6 Data Saving

3.2 First Findings

3.2.1 Simple Nim

3.2.2 Nim

3.3 Complexification

Chapter 4

Wrapping Up

4.1 Auto Review

4.2 Future Work

Chapter 5

Appendix

5.1 Code

5.2 Data

5.3 Documentation

5.4 References

Bibliography

Stanley, Kenneth O. and Risto Miikkulainen (June 2002). "Evolving Neural Networks through Augmenting Topologies". In: *Evolutionary Computation* 10.2, pp. 99–127. ISSN: 1063-6560. DOI: 10.1162/106365602320169811. eprint: https://direct.mit.edu/evco/article-pdf/10/2/99/1493254/106365602320169811.pdf. URL: https://doi.org/10.1162/106365602320169811.

Chandra, Akshay L (2022). *McCulloch-Pitts Neuron - mankind's first mathematical model of a biological neuron*. URL: https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1.

Caruso, Catherine (2023). *A New Field of Neuroscience Aims to Map Connections in the Brain*. Accessed: 2024-11-03. URL: https://hms.harvard.edu/news/new-field-neuroscience-aims-map-connections-brain.

Newman, Tim (2023). *Neurons: What are they and how do they work?* URL: https://www.medicalnewstoday.com/articles/320289#carry_message.

Commons, Wikimedia (2023). *Colored neural network*. Accessed: 2024-11-03. URL: https://commons.wikimedia.org/wiki/File:Colored_neural_network.svg.

contributors, Wikipedia (2023). *Nim - Wikipedia*. Accessed: 2024-11-03. URL: https://en.wikipedia.org/wiki/Nim.