NOVA IMS
Information Management School

MMAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

Towards The Deep Semantic Learning Machine Neuroevolution Algorithm

An exploration on the CIFAR-10 problem task

Olivier Jean Marie Hofman

Dissertation submitted in partial fulfillment
of the requirements for the Master's degree
Data Science and Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão da Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Towards The Deep Semantic Learning Machine Neuroevolution Algorithm

An exploration on the CIFAR-10 problem task

Olivier Jean Marie Hofman

December 2020

Dissertation submitted in partial fulfillment of the requirements for the Master's degree Data Science and Advanced Analytics

Supervisor:
Dr. Ivo Gonçalves, Universidade de Coimbra
Co-Supervisor:
Assoc. Prof. Mauro Castelli, Universidade Nova de Lisboa

Towards the Deep Semantic Learning Machine Neuroevolution Algorithm
An exploration on the CIFAR-10 problem task

This document was created using the (pdf)LaTeX processor, based on the NOVAthesis template, developed at the Dep. Informática of FCT-NOVA by João M. Lourenço.

Acknowledgements

I would like to thank everyone who has been involved directly and indirectly in my academic path so far, and particularly the people who have helped during my thesis.
I would like to thank professor Mauro Castelli and professor Ivo Gonçalves for their guidance, shared knowledge and the effort they put in this project.
Also, I would like to thank professor Leonardo Vanneschi for his inspiring lectures on Machine Learning and Evolutionary Algorithm, and Illya Bakurov for his always entertaining tutorials throughout the Master's degree.
Much love and appreciation for my parents, for giving me the opportunity to study in Portugal and always supporting me in what I do and where I go, no matter what happens.
I would like to thank all my Portuguese friends in Lisbon and elsewhere for teaching the Portuguese language and showing me their beautiful country!
And, thanks to Lucas, for teaching me a lot about Data Science and life in general, and for all the heated discussion we had on topics I actually had no clue about;)
Finally, I would like to thank Mother Nature for providing so many amazing waves and surf sessions at all the beautiful beaches of Portugal throughout my study period!
Obrigado!

Abstract

Selecting the topology and parameters of Convolutional Neural Network (CNN) for a given supervised machine learning task is a non-trivial problem. The Deep Semantic Learning Machine (Deep-SLM) deals with this problem by automatically constructing CNNs without the use of the Backpropagation algorithm. The Deep-SLM is a novel neuroevolution technique and functions as stochastic semantic hill-climbing algorithm searching over the space of CNN topologies and parameters. The geometric semantic properties of the Deep-SLM induce a unimodel error space and eliminate the existence of local optimal solutions. This makes the Deep-SLM potentially favorable in terms of search efficiency and effectiveness.

This thesis provides an exploration of a variant of the Deep-SLM algorithm on the CIFAR-10 problem task, and a validation of its proof of concept. This specific variant only forms *mutation node* $\rightarrow$ *mutation node* connections in the non-convolutional part of the constructed CNNs. Furthermore, a comparative study between the Deep-SLM and the Semantic Learning Machine (SLM) algorithms was conducted. It was observed that sparse connections can be an effective way to prevent overfitting. Additionally, it was shown that a single 2D convolution layer initialized with random weights does not result in well-generalizing features for the Deep-SLM directly, but, in combination with a 2D max-pooling down sampling layer, effective improvements in performance and generalization of the Deep-SLM could be achieved. These results constitute to the hypothesis that convolution and pooling layers can improve performance and generalization of the Deep-SLM, unless the components are properly optimized.

Keywords: Supervised Machine Learning; Deep Learning; Convolutional Neural Networks; Neuroevolution; Deep Semantic Learning Machine;

Resumo

Selecionar a topologia e os parâmetros da Rede Neural Convolucional (CNN) para uma tarefa de aprendizado automático supervisionada não é um problema trivial. A Deep Semantic Learning Machine (Deep-SLM) lida com este problema construindo automaticamente CNNs sem recorrer ao uso do algoritmo de Retro-propagação. A Deep-SLM é uma nova técnica de neuroevolução que funciona enquanto um algoritmo de escalada estocástico semântico na pesquisa de topologias e de parâmetros CNN. As propriedades geométrico-semânticas da Deep-SLM induzem um unimodel error space que elimina a existência de soluções ótimas locais, favorecendo, potencialmente, a Deep-SLM em termos de eficiência e eficácia.

Esta tese providencia uma exploração de uma variante do algoritmo da Deep-SLM no problemo de CIFAR-10, assim como uma validação do seu conceito de prova. Esta variante específica apenas forma conexões *nó de mutação* → *nó de mutação* na parte non convolucional da CNN construída. Mais ainda, foi conduzido um estudo comparativo entre a Deep-SLM e o algoritmo da Semantic Learning Machine (SLM). Tendo sido observado que as conexões esparsas poderão tratar-se de uma forma eficiente de prevenir o overfitting. Adicionalmente, mostrou-se que uma singular camada de convolução 2D, iniciada com valores aleatórios, não resulta, directamente, em características generalizadas para a Deep-SLM, mas, em combinação com uma camada de 2D max-pooling, melhorias efectivas na performance e na generalização da Deep-SLM poderão ser concretizadas. Estes resultados constituem, assim, a hipótese de que as camadas de convolução e pooling poderão melhorar a performance e a generalização da Deep-SLM, a não ser que os componentes sejam adequadamente otimizados.

Palavras-chave: Aprendizado Automático Supervisionada, Aprendizado Profundo, Rede Neural Convolucional, Neuroevolução, Deep Semantic Learning Machine

Contents

List of Figures

List of Tables

Acronyms

AM	Adaptive Mutation
ANN	Artificial Neural Network
BP	Backpropagation
CE	Cross-Entropy
CL	Convolution Layer
CNN	Convolutional Neural Network
CP	Convolution Part
Deep-SLM	Deep Semantic Learning Machine
EA	Evolutionary Algorithm
FCL	Fully Connected Layer
GD	Gradient Descent
GP	Genetic Programming
GSC	Geometric Semantic Crossover
GSGP	Geometric Semantic Genetic Programming
GSM	Geometric Semantic Mutation
ML	Machine Learning
MLP	Multilayer Perceptron Network
NCP	Non-Convolution Part
NE	Neuroevolution

NN	Feedforward Neural Network
OA	Optimization Algorithm
OP	Optimization Problem
PL	Pooling Layer
SLM	Semantic Learning Machine
SLP	Single Layer Perceptron Network

Chapter 1

Introduction

The 21th century marked itself as the century in which large amount of *data* ("big data") and the ability to process big data became available. This phenomenon was driven by the ever increasing trend in advancements in computational resources, which started in the early 1970's and evolved like this over the past decades (Schaller, 1997). These advancements allowed the successful application of sophisticated Machine Learning algorithms on big data for the first time in history.

A key role in this digital revolution is played by Artificial Neural Networks (ANNs) (Rebala et al., 2019). ANNs encompass one of the most popular classes of ML algorithms and have a wide range of industry applications, which include: the detection of patterns and objects in images (Khan et al., 2018; Anwar et al., 2018), interpretation and translating natural language (Lopez et al., 2017) and supervising autonomous vehicles (Bojarski et al., 2016). Thereby, ANNs have shown to provide excellent performance and efficiency in learning complicated patterns in various difficult problem settings.

ANNs are a sophisticated function approximation method, yet, they consist of simple structure inspired by the human brain. ANNs form networks of internally connected *nodes* ("neurons") and have a set of *weights* ("synapses") defining the strength of the internal connections. Similar to a biological neuron, an *artificial neuron* accepts an input and produces an output through a set of transformations involving an *activation function*.

A more advanced type of ANN is a Convolutional Neural Network (CNN). CNNs are characterized by additional Convolution Layers (CLs) and Pooling Layers (PLs) incorporated into their structure (LeCun et al., 1988). These layers extract *spatially invariant features* from the input data through operations called *convolution* and *pooling*. Regular artificial neurons do not consider the spatial information in the input data. For this reason, CNNs are a powerful method due to their ability to extract spatially invariant features, which is relevant when solving spatially structured problems (e.g. input data based on images, videos or any other type of multidimensional data).

To achieve good performance on a certain task, a CNN needs to learn a set of weights to induce the desired output of the problem at hand. The Backpropagation (BA) algorithm iteratively fine-tunes the weights in the learning phase, by propagating backwards the network's error on a provided set of training examples (Rumelhart et al., 1986).

Currently, the approach with the BA algorithm is the state-of-the-art approach, however, to the question on what network topology to use, the BA algorithm does not provide an answer, nor, it will always provide a "sufficient"good solution. The training procedure with the BA algorithm involves usually a tedious and difficult optimization, and is often the bottleneck in industry applications for ANNs (Sharma et al., 2017).

Neuroevolution (NE) deals with these issues by using Evolutionary Algorithms to derive the topology and weights of an ANN. Recently, a novel NE technique was proposed: the Semantic Learning Machine (SLM) - a stochastic hill-climbing algorithm searching over the space of ANN typologies (Gonçalves et al., 2015b). The SLM constructs fully-functioning ANN without the use of the BA algorithm. The SLM has shown to be efficient in terms of performance and generalization ability against other NE techniques (Jagusch et al., 2018).

An extension of the SLM algorithm to the search space of CNNs - the Deep Semantic Learning Machine (Deep-SLM) algorithm - has not been provided yet. Therefore, this thesis will explore the Deep-SLM algorithm on the CIFAR-10 problem task, and validate its proof of concept. The geometric semantic properties of the Deep-SLM induce a unimodal error space and eliminate the existence of local optima. This makes the Deep-SLM potentially favorable in terms of search efficiency and effectiveness.

To provide an exploration of the Deep-SLM on the CIFAR-10 problem task and validate its proof of concept, this thesis is organized as follows: chapter 2 introduces the theory and background, chapter 3 outlines the experimental methodology, chapter 4 presents and discusses the experimental results, and chapter 5 derives the conclusion.

Chapter 2

Theory & Background

This chapter introduces the theory and background of this thesis, including all concepts required for understanding the research conducted in this work. Section 2.1 starts by introducing the field of Supervised Machine Learning, building on top of that, section 2.2 introduces the concept of Artificial Neural Networks, consecutively, section 2.3 introduces the concept of Evolutionary Algorithms and the Semantic Learning Machine, and, finally, in section 2.4 the Deep Semantic Learning Machine algorithm will be defined.

2.1 Supervised Machine Learning

Learning is the act of acquiring new knowledge or modifying existing knowledge over time, based on previous experiences, events or a set of examples (Holt et al., 2012).

The application of computational algorithms with the objective to induce a general pattern or relation (a *function*) from a provided set of examples (*instances*) is a form of *computational learning*. The term "learning"in this context implies that there is no explicit information provided about the underlying function that has to be learned. The computational algorithm has to discover this hidden function in an automated and independent fashion, using only the information provided by the set of instances. The result is a mathematical model or a set of rules, which not only model the underlying relation or pattern, but also allow for accurate predictions or decision on new given data instances. The algorithm was not explicitly programmed to do so, but it has

"learned"the hidden function and now it successfully applies its acquired intelligence on never-before-seen data instances. This process of computational learning, with the objective to acquire *computational intelligence*, is known as Machine Learning (ML).

ML is either performed in a *supervised*, *unsupervised* or *semi-supervised* fashion. Which ML strategy is selected, depends on the nature of the provided set of instances, the type computational algorithm used and the learning strategy selected by the user.

The instances of the provided set of examples can either be labeled (supervised), unlabeled (unsupervised) or partially labeled (semi-supervised) with their respective target labels or values.

In a supervised ML setting, the target labels are known beforehand and allow for a direct and qualitative feedback on the performance of the algorithm, during and after the learning phase. This is in contrast to an unsupervised ML approach, in which the absence of true target labels make it difficult, or in most cases impossible, to asses the quality of the learned patterns or relations directly during and after the learning phase. A disadvantage of supervised ML is that there might not be a labeled set of instances available for the given problem at hand, and it might be costly and time consuming to generate such a dataset.

In this thesis, only the the supervised ML approach will be considered.

2.1.1 Supervised Learning Theory

A supervised learning setting is defined by an input space X, an output space Y and a *target function* $r : X \rightarrow Y$, which characterizes an unknown joint probability distribution $P = (X,\ Y)$.

The objective of the supervised learning algorithm is to learn a *model h:* $X \rightarrow Y$, which approximates the target function r with arbitrary close precision. In essence, the model h models a conditional probability distribution $P = (Y|X)$, which describes a relation that maps $X \rightarrow Y$.

To learn any model h, the learning algorithm can only consider the available empirical information. The available emperical information is the labelled set of instances $\mathbb{X}$ of size n, organised as $\mathbb{X} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \ldots (\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}$, where $\mathbf{x}^{(i)} \in X$, $\mathbf{y}^{(i)} \in Y$ and $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ represent the ith instance in $\mathbb{X}$ drawn i.i.d. from the distribution $P = (X,\ Y)$.

Let the set H hold all valid models mapping $X \rightarrow Y$, which include only models which can be derived from the empirical information in $\mathbb{X}$. Then, $\forall\ h \in H$ can be considered by the learning algorithm. In essence, the learning algorithm "searches over all valid

models"in the model space H with the objective to find the most appropriate one; the model h_{best}, which approximates the target function r the closest as possible.

A given supervised learning problem is either a *regression* or *classification* task. Both tasks exist in single and multiple target forms. The targets in a regression task are continuous real numbers, whilst for a classification task, the targets have a categorical meaning and can be represented by natural integers.

Let $\mathbf{x}^{(i)}$ be the ith feature vector in $\mathbb{X}$ of d dimensions, with its corresponding target vector $\mathbf{y}^{(i)}$ of k dimensions. Then, the matrix $\mathbf{X}^{\mathrm{n}\times\mathrm{d}}$ contains all feature vectors of $\mathbb{X}$ and, for a multi-target regression, the matrix $\mathbf{Y}^{\mathrm{n}\times\mathrm{k}}$ represents the target matrix containing all target vectors of $\mathbb{X}$. For a single target regression where $k = 1$, the target matrix $\mathbf{Y}$ is then the single column vector $\mathbf{Y}^{\mathrm{n}\times 1}$.

Assume a model h_i is a solution to the supervised learning task at hand. The model h_i is then characterized by the correctness of its approximation to the target function r.

First, define $\hat{\mathbf{Y}}_i$ as applying the model h_i to the feature matrix $\mathbf{X}$:

$$\hat{\mathbf{Y}}_i = h_i(\mathbf{X}). \tag{2.1}$$

Then, the *loss function* L is used to quantify the quality of the approximation of h_i to r. It quantifies the distance between the prediction matrix $\hat{\mathbf{Y}}_i$ and target matrix $\mathbf{Y}$. Define loss function L: $\hat{\mathbf{Y}} \times \mathbf{Y} \rightarrow \mathbb{R}$ as:

$$L(\hat{\mathbf{Y}}_i, \mathbf{Y}) = M(h_i(\mathbf{X}), \mathbf{Y}) \tag{2.2}$$

where M is some *metric function* $M : \hat{\mathbf{Y}} \times \mathbf{Y} \rightarrow \mathbb{R}$.

The *true risk* R_{true} (or more commonly the *true error*) associated with the given model h_i is the integral of L with respect to the joint distribution $P = (X,\ Y)$ (Vapnik, 1992). Define R_{true}, given the model h_i and the loss function L, as:

$$R_{\text{true}}(h_i) = \int L(h_i(X), Y)\, dP(X, Y). \tag{2.3}$$

Note that the true risk R_{true} in equation 2.3 is equivalent to $\mathbb{E}[L(h_i(X), Y)]$, the expected value of the loss function L over $\forall\ \mathbf{x} \in X$ (Vapnik, 1992).

By definition, the target function r is unknown and so is the distribution $P(X, Y)$. Therefore, it is impossible to solve the integral in equation 2.3 or calculate the true risk R_{true} for any given h_i.

Given a model h_i, only the *empirical risk* R_{emp} over the empirical observations in $\mathbb{X}$ can be calculated. This is done by normalizing the loss L for $\forall\ \mathbf{x} \in \mathbb{X}$ by, for example,

taking the average of the loss function L over all empirical observations (Vapnik, 1992). This can be done as follows for $\forall\ (\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in \mathbb{X}$:

$$R_{\text{emp}}(h_i) = \frac{1}{n} \sum_{i=1}^{n} L(h_i(\mathbf{x}_i), \mathbf{y}_i). \tag{2.4}$$

In essence, a supervised learning algorithms performs *empirical risk minimization* by searching over the model space H for a model h_{min} that minimizes the empirical risk function of equation 2.4. Formally, this can be written as:

$$h_{min} = \arg\min_{h \in H} R_{emp}(h), \tag{2.5}$$

and is in principle an optimization problem, which can be solved by an optimization algorithm (section 2.1.3).

2.1.2 Overfitting, Underfitting & Generalization Ability

The assumption that minimizing the empirical risk R_{emp} of equation 2.4 results directly in a minimization of the true risk R_{true} of equation 2.3 is wrong. A model that minimizes the empirical risk R_{emp} is not guaranteed to minimize the true risk R_{true}.

For example, high capacity models have the capability of simply memorizing the empirical observations in $\mathbb{X}$. These models will perform extremely well on the minimization of the empirical risk R_{emp}, but will fail to perform well on the objective of minimizing the true risk R_{true}.

In practise, these models are useless and are said to not *generalize* well to new data instances, and this is called *overfitting*. If the opposite is the case, and the derived model h is too "simple"and does not grasp the overall trend of the relation within $\mathbb{X}$, the model is said to be *underfitting*.

In both cases, there exists a *bias-variance* difference. This bias-variance issue might arise when supervised learning is performed. The most common way to counter act this phenomenon is to divide the empirical observations of $\mathbb{X}$ in a *training set* and a *test set* (and in some cases a third *validation set*) and monitor the loss function L over these subsets, during and after the learning phase. The loss information over those subsets is then taken into account when a final model is selected by the computational learning algorithm or the end user.[1]

[1] There exist various techniques to counter act the bias-variance issue. Some of them are algorithm specific, others are not. For an elaborate discussion on how to prevent overfitting and underfitting, the reader can refer to Rebala et al., 2019.

2.1.3 Optimization In The Context Of Supervised Learning

Empirical risk minimization, as presented in equation 2.4, is an example of an Optimization Problem (OP). The objective in equation 2.4 is to find a model h_{min} out of many, which minimizes the loss function L of equation 2.2. The loss function L is in this case the *objective function* that has to be minimized by the Optimization Algorithm (OA).

More generally, an OP is defined by a *search space* S with a corresponding *fitness space* f. Formally, The OP tries to *minimize* (or *maximize*) the fitness function f. Define the general definition of a minimization problem as follows:

$$s_{\text{global}} = \arg\min_{s \in S} f(s). \tag{2.6}$$

Maximizing the fitness function is the inverse of minimizing the fitness function. A minimizing problem can be transformed into maximization problem by multiplying the fitness function simply -1. The search space S is the domain containing all possible solutions to the OP. The correctness of a solution s is quantified by a *fitness function f:* $S \rightarrow \mathbb{R}$ and scores the quality of $\forall\, s \in S$. The OA traverses the search space S through a *neighbourhood function N:* $s_{\text{current}} \rightarrow s_{\text{next}}$, with the objective to find the best fit solution s_{global} for the OP at hand.

Any supervised ML problem can be considered an OP with the objective to minimize the loss function L over a search space S. For the given supervised ML problem the search space S is then $\forall\, h \in H$.

A practical example of this is the optimization of the topology, weight values and bias values of an Feedforward Neural Network (section 2.2). Here, the search space S consists of all possible structures of the network, as well as for each structure, all possible combinations for the weight values and bias values. In this example, the search space S is infinitely big due to the fact that theoretically the network can have an infinite number of neurons and layers.

Besides that, the corresponding fitness landscape induced by the fitness function f is usually highly *multimodal* (Goodfellow et al., 2016). This means there exist multiple sub optimal solutions to the OP. Also, it can be expensive to evaluate the individual solutions of the OP, especially when thousands of solutions need to be evaluated by the OA when the search space S is explored.

How the search space S is explored depends on the nature of the OA used in the optimization. Some OAs allow only for the optimization of the weight values and bias

values of the network, and not for the structural component. The numerical optimization method Gradient Descent (section 2.2.3) is an example of this. This is in contrast other OAs, such as Neuroevolution of Augmenting Topologies and the Semantic Learning Machine (section 2.3.3), which have the capability of optimizing the weight values and bias values of the network at the same time as evolving the topology.

2.2 Artificial Neural Networks

ANNs are graphs of internally connected information processing units, capable of solving complex computational problems in various domains, including: computer vision, forecasting and clustering (Khan et al., 2018; Anwar et al., 2018; Lopez et al., 2017; Bojarski et al., 2016; Goodfellow et al., 2016). ANNs form an independent class of ML models and are widely applied in the fields of supervised, unsupervised and reinforcement ML. This section will only cover the relevant theory for this thesis, which is the ANNs in the form of a Feedforward Neural Network.

A Feedforward Neural Network (NN) is a network of *artificial neurons* ("neurons") which are partially inspired by the biological neurons in the human brain. The information flow in a NN is unidirectional, from input neurons to output neurons, and there exist no cyclic connections. The neurons are organised in consecutive layers, with in between the layers connections ("synapses") connecting two neurons at a time. Every connection is characterized by a weight value, which defines the strength of that connection. This weight value is not fixed but can be adjusted - it is a parameter of the network. Accordingly, the variable θ will be used to represent all degrees of freedom (parameters) of a NN.

By accepting an input and producing an output, a NN, in essence, represents a mathematical function. By altering the network's parameters θ, a NN functions as sophisticated function approximation method. A NN can be interpreted as a model h that approximates some real function r (section 2.1.1). Each NN is then characterized by the its unique parameters θ, and the structure in which the parameters θ are organised. Therefore, each model h_i representing a NN will be referred to as h_θ.

The application of NNs are characterized by a *training phase* and a *generalization phase*. In the training phase, the NN "learns"how to process the input information correctly. The network's parameters are adjusted until it produces the desired output to solve the

supervised ML problem at hand. The alternation of the weights is done by executing the Backpropagation algorithm (section 2.2.3). Then, after, the "trained"NN is used to output accurate predictions on new incoming information in in the generalization phase,.

2.2.1 Single Layer Perceptron Networks

A Single Layer Perceptron Network (SLP) belongs to the family of NNs. It contains a single layer and it has n output neurons. An SLP containing only a single output neuron is the most elementary form of a modern NN, and is known as the Rosenblatt's Perceptron Model (Rosenblatt, 1958).

The single neuron in the Rosenblatt's Perceptron Model is connected to all input neurons and each connection is represented by a weight w. Additionally the single neuron has a bias connection. The input neurons are special neurons which do not perform any information processing, they only facilitate the information entering the network.

The single output neuron processes the incoming information by calculating the linear combination of its input activations, multiplied with the corresponding weight values and the bias value. Then, the neuron applies its (non-linear) *activation function* and produces an output in the form of a single real value - the neuron's *output activation*. Formally, given n number of input neurons, let the output activation y_{out} of a neuron be defined as:

$$y_{\text{out}} = f_{act}(\sum_{i=1}^{n} w_i x_i + bias) \tag{2.7}$$

where f_{act} is the activation function of the neuron, x_i is the input activation of the ith input neuron, w_i the weight the connection from the ith incoming neuron to the neuron and $bias$ the bias value of the neuron. A schematic overview a single neuron SLP with three input neurons is provided in figure 2.1.

The activation function of the output neurons of a NN depend on the supervised ML task at hand. For a binary classification task, the non-linear Sigmoid function (equation 2.8) is commonly used. The Sigmoid activation transforms the neuron's activation into the range $[0, 1]$, which allows for a probabilistic interpretation of the network's binary predictions. Let the Sigmoid activation function be defined as follows:

$$f_{\text{sigmoid}} = \frac{1}{1 + e^{-(w^T x + b)}}. \tag{2.8}$$

For a multi-class classification task, with n separable classes, n output neurons are required to cover all output probabilities. The Sigmoid activation function is replaced with the Softmax activation function, which normalizes all n output probabilities of the n output neurons into a n dimensional vector, which allows for a correct probabilistic interpretation of the output of the SLP network. Let the Softmax activation function be defined as follows:

$$f_{\text{softmax}}(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}. \tag{2.9}$$

In the case of a regression task, the output activation function of the neurons is replaced by the Identity function. The Identity function is defined as follows:

$$f_{\text{identity}}(x) = x \tag{2.10}$$

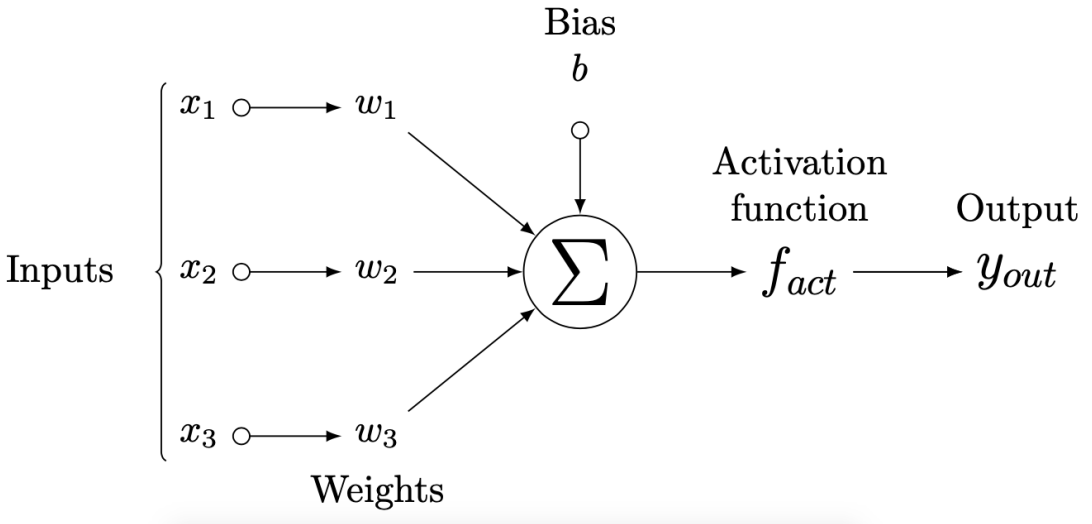

Figure 2.1: Schematic overview of Rosenblatt's Perceptron Network with three input neurons and single output neuron

2.2.2 Multilayer Perceptron Networks

The Multilayer Perceptron Network (MLP) extends the SLP by having m Fully Connected Layers (FCLs), in which each FCL has n hidden neurons. The neurons in layer m_i receive the output activations of the neurons in layer m_{i-1} and propagate their activations into the neurons of layer m_{i+1}. This process continues until the output neurons in the output layer m_{output} are reached and the MLP's final output prediction is produced.

Except for having multiple FCLs, the same principles of the SLP (section 2.2.1) apply the each individual neuron in the MLP. A schematic overview of the structure of a MLP consisting of three input neurons, two hidden layers a single output neuron, is provided in figure 2.2.

The MLP design is a deeper and more sophisticated one, which allows for the approximation of more complex functions. In fact, the Universal Approximation Theorem of NNs states that, a single hidden layer with a finite number of hidden neurons, is sufficient to approximate any function with a finite number of discontinuities, under the constraint that the activation function of the hidden neurons is non-linear (Hartman et al., 1990). Although this is a remarkable property of MLPs, the Universal Approximation Theorem does not state anything about the learnability of the parameters θ for this approximation with arbitrary close precision. In practise, this is commonly a non trivial task which will be discussed further in section 2.2.3.

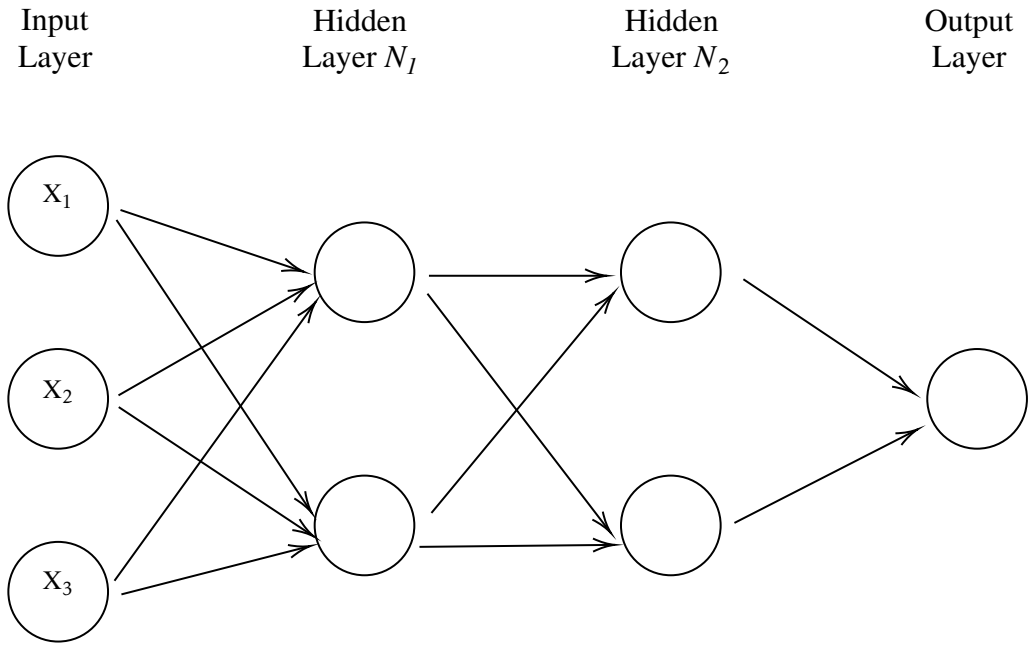

Figure 2.2: Schematic overview of an MLP network with three input neurons, two hidden layer with hidden neurons and single output neuron

Deeper NNs require the use of continuous and differentiable activation functions inside the neurons of the network. This constraint is the result of the commonly used Backpropagation Algorithm (section 2.2.3), required for learning the correct weights of

the connections, to solve the supervised ML problem at hand. The four most commonly used non-linear activation function in deeper NNs are provided in figure 2.3.

An important motivation for the use of non-linear activations functions lies in the fact that these functions introduce favourable properties into the NN - their non-linearity. The non-linear nature of these functions introduce non-linear dependencies into the deeper network, which allow for approximating more sophisticated and non-linear target functions and decision boundaries by the MLP. Without non-linearity, deeper networks would effectively be linear combinations of linear combinations, and thus, incapable of approximating non-linear functions and decision boundaries (Rojas, 2013). [2]

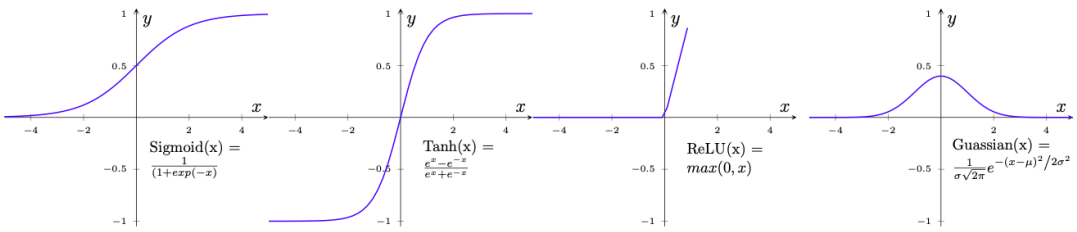

Figure 2.3: Four commonly used non-linear activation functions in the Multilayer Perceptron Network; the Sigmoid, the Hyperbolic Tangent, the Rectified Linear Unit and the Gaussian

2.2.3 The Backpropagation Algorithm

The Backpropagation (BP) algorithm is an iterative algorithm for learning the correct parameters θ of a NN in the training phase (Rumelhart et al., 1986; LeCun et al., 1988). In each iteration, all parameters θ are updated once by the BP algorithm, under the constraint of minimizing the *error function* E. This achieved by first propagating forward the network the current activations (*propagate_activations_forward* in algorithm 1), and then, propagating backwards the network's prediction error on a single input activation, or a batch of input activations (a *mini-batch*) (*propagate_errors_backward* in algorithm 1).

The error function E is similar to the loss function presented in equation 2.2. It also quantifies the difference between the network's predictions $\hat{\mathbf{Y}}$ and target matrix $\mathbf{Y}$ by some metric M, but, additionally, it depends on the parameters θ of the network.

[2] There exist other non-linear activation functions and in principle every continuous function can be used, for more information about NN activation functions, the reader can refer to Rojas, 2013 for more information

Define the error function as follows:

$$E(\mathbf{X}, \theta) = M(h_\theta(\mathbf{X}), \mathbf{Y}). \tag{2.11}$$

Algorithm 1: Pseudo code of the Backpropagation algorithm

```
Set t_current = 0;
Initialize t_max;
Initialize network parameters θ;
while error E ≠ 0 or t_current ≤ t_max do
    propagate_activations_forward(X, θ);
    propagate_error_backwards(activations);
end
```

Due to the minimization constraint on the error function E, a numerical optimization method is required in the BP procedure to determine the weight's update value. In principle, an OP (section 2.1.3) is being solved in the BP procedure. Although various numerical methods are valid for this optimization, the Gradient Descent (GD) has proven to be the most suited for this specific task (Goodfellow et al., 2016).[3]

GD uses the gradient of the error function, in combination with a learning rate γ, to update each individual weight in the network. Let n be the number of weights in the network, then, the gradient of E, with respect to the weights, is defined as:

$$\nabla E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n}). \tag{2.12}$$

The learning rule for updating a weight w_i, given the learning step γ, is then

$$\Delta w_i = -\gamma \frac{\partial E}{\partial w_i}. \tag{2.13}$$

.

The backpropagation approach with the underlying GD optimizer is well suited if the problem at hand is linearly separable. A linear separable problem induces a unimodal error surface E, with respect to the weights in the network. This guarantees that the GD optimizer will converge to the global optimal solution, if the the learning step γ is chosen correctly (Gori et al., 1992).

[3] For an extensive overview of numerical methods for optimizing NNs, the reader can refer to Goodfellow et al., 2016 for more information.

In practise, real-world ML problems are often not linearly separable and induce a non-convex error surface. In this case, the GD optimizer is not guaranteed to converge to the global optimal solution. The GD optimizer not converging to a "good enough"solution is a major problem in the application of deep NN (Goodfellow et al., 2016).

Some strategies can be applied to improve the convergence of GD, which are; improving the initial starting point of the optimization, adjusting the learning rate γ and introducing stochastic variation in the weight's update value.

The initial starting point of the optimization is the initial parameters θ. An appropriate initialization is known to speed up convergence of the GD (Goodfellow et al., 2016). Also, having a adjustable learning rate γ will speed up the convergence (Goodfellow et al., 2016). Introducing stochastic variation into the weight's update value has also shown to be effective technique to improve convergence. This has led to the introduction Stochastic GD, without momentum parameter α and without momentum (Goodfellow et al., 2016).

Stochastic GD introduces stochastic variation into the weight's update value by sampling a mini-batch of the training set of examples, and derives the gradient over this mini-batch. The momentum parameter α introduces another factor of stochastic variation to the weight's update value. It does so by introducing stochastic variation based on the previous weight's updated value from iteration $t-1$. The tweaked equation for Stochastic GD with momentum parameter is provided in equation 2.14

$$\Delta w_i = -\gamma \frac{\partial E}{\partial w_i} + \alpha \Delta_{t-1} w_i. \tag{2.14}$$

Other variants have been introduced for the momentum parameter α. For example, the Nesterov's momentum, which reduces the "snowball effect"induced by the standard momentum parameter in equation 2.14. Additionally, there have been introduced numerous other algorithms to heuristically derive the Nestrov's momentum. Famous examples of these are: AdaGrad, RMSProp, and Adam optimization algorithms.[4].

The optimization of the topology and the parameters θ of a NN is in general a non-trivial task. Distributed computing power, time, domain knowledge and a trial and error hyper parameter tuning approach is the most "efficient"way of deriving the appropriate set of parameters for a NN, to solve a given supervised ML problem. This process has

[4]For an extensive overview of possible techniques to improve the convergence of GD, the reader can refer to Goodfellow et al., 2016 for more information

always been, and still is, the main bottleneck in the application of NNs, and is essential in deploying a practical model.

This shows also the importance of gradient-free optimization techniques, such as Genetic Programming (section **??**) and especially the potential of neuro-evolution algorithms (section 2.3.2), which can optimize the structure and the parameters θ of NNs without relying to on domain knowledge and gradient information of the error function.

2.2.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) (LeCun et al., 1989) are a family of more sophisticated NNs. CNNs extend the standard MLP network architecture with CLs and PLs. The CLs an PLs compose the Convolution Part (CP) of the CNN, and this part is located in front of the Non-Convolution Part (NCP) of the CNN, which is composed by regular FCLs of a MLP network (section 2.2.2).

The CLs and PLs transform the input activations of the CNN through a serial combinations of mathematical operations called *convolution* and *pooling*. The result of this are convolved and down sampled representations of the original input activations, which are called *feature maps*. The feature maps are then flattened and fed into the NCP of the CNN, in which normal classification or regression is performed according to the principles of the MLP network (section 2.2.2).

In essence, the CP of a CNN performs feature extraction and feature reduction for the NCP of the CNN. An example of a CNN architecture is provided in figure 2.4. This network is know as the LeNet5 network and consists of 3 CLs and 2 PLs with a 2 hidden FCL structure in the NCP's MLP (LeCun et al., 1998).

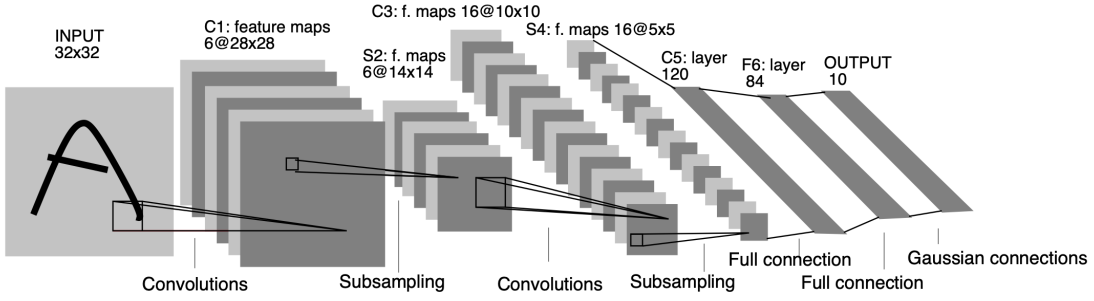

Figure 2.4: LeNet5 architecture for digit recognition task designed by LeCun et al. which consists of 3 CLs and 2 PLs with a 2 hidden FCL structure in the MLP of the NCP.

CNNs should be applied when the input data holds any type of spatial information, which could be relevant for the solution of the problem at hand. The spatial structure within the data can either be a grid-type topology, such as image data (satellite, computerized tomography scans or a sensor data), or any other type of n dimensional data volume, such as sequence data (video, audio or time-series data).

CNNs manage to take in consideration the spatial information present in the input data due its CLs and PLs. The serial combination of CLs and PLs in the CP are capable of extracting *spatially invariant* features. Here, the term spatially invariant implies that the position or the rotation of the target feature in the input data is not relevant for the detection of the feature. If the feature is present "somewhere"in the input data, the corresponding neuron sensitive for that particular feature in the CP will detect the feature and be activated. This ability of detecting spatially invariant features in the CP of the CNNs is what makes CNNs excellent for the exploitation of spatial information hidden within the input data.

Regular neurons in a FCL of an MLP network do not take in consideration the spatial information present in the input data. The linear combination within the formula of the activation of a regular neuron (presented in equation 2.7) shows that any spatial information will be lost due to the summation of all the incoming activations. All incoming activations are treated equally, with respect to the corresponding weights of the connections, but any spatial information is lost. Therefore, the capability of automatically detecting and extracting spatially invariant features in the CP, in combination with a fully connected MLP network in the NCP, is what makes CNNs a powerful method that has shown ground-breaking results on image classification competitions and other difficult tasks (Rawat et al., 2017).

The CL is the characteristic component of a CNN and is part where the incoming data is convolved over m dimensions. In principle, the convolution operation is the application of a m dimensional *filter* (or *kernel*) on the incoming data volume. The output a convolved representation of the input into the kernel.

This process can also be described as an m dimensional filter sliding over the input data, and performing element wise multiplication and additions of the filter values with the overlapping input data values (the kernel's *receptive field*). In figure 2.5 shows a practical example of the convolution operation on a $10x10$ input data volume, convolved by three 2D convolution kernels with receptive field dimensions $3x3$ and stride values of $1x1$.

Formally, given the incoming data data volume I with $m = 2$ dimensions, K the

convolution kernel and S the resulting feature map, then, the convolution operation is defined as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{2.15}$$

In equation 2.15, note that the dimensions of feature map S will be always smaller than the dimensions of the input data volume I (if any of the receptive field dimensions of $K > 1$).

In the CNN architecture, the CL is commonly followed up by a PL. Similar to the CL, the PL contains a kernel with dimensions pxp, which is the receptive field. Let the feature map S be fed into this PL, then, the feature map S can be represented as an assemblage of local pooling regions of size k as $S = (s_1, s_2, \ldots, s_{PxP})$ (Zhao et al., 2018).

The resulting individual pooling region values depend on the type of pooling that is applied on the feature map S. Figure 2.5 shows a practical example of the pooling operation on a $3x8x8$ input data volume, pooled by a 2D pooling kernel with dimension $2x2$ and stride values of $2x2$.

There exist various types of pooling, but the two commonly used pooling operations are *max pooling* and *average pooling*. Max pooling selects the "most active"element within its the receptive field of the pooling kernel and will therefore be more sensitive to texture information (Zhao et al., 2018). Define the max-pooling operation, given the incoming assemblage of k pooling regions S and pooling kernel dimensions of pxp, as:

$$Output\ Pool_{\max} = \max_{1 \le k \le PxP}(s_k). \tag{2.16}$$

Average pooling takes in consideration the complete receptive field and is therefore more sensitive to the background information (Zhao et al., 2018). Define average pooling operation, given the incoming pooling region S and pooling kernel dimensions of pxp, as:

$$Output\ Pool_{\text{avg}} = \sum_{1}^{pxp}(s_k). \tag{2.17}$$

A practical example of a serial 2D convolution - 2D pooling operation is provided in figure 2.5.

CNNs are considered as one of the best success stories of biologically inspired computational techniques (Bengio et al., 2009). The deep CNN architecture mimics the functioning of the virtual cortex of a primate's ventral pathway. With its deep hierarchical structure, it is similar to the functioning of earlier sensor areas of the visual

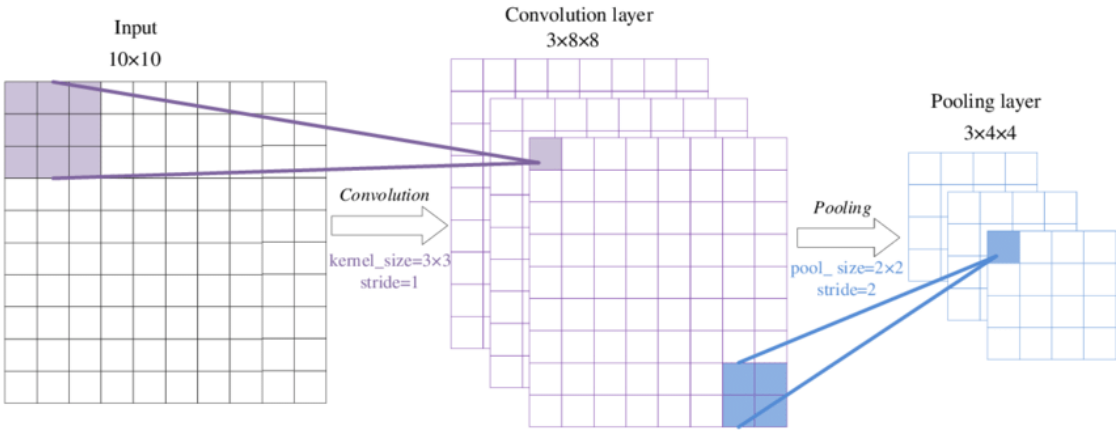

Figure 2.5: Practical example of a convolution operation on a $10x10$ input data volume, followed up by a 2D pooling operation. First, three individual 2D convolution kernels with dimensions $3x3$ and strides of $1x1$ convolve the input volume. Then, the resulting feature map with dimensions $3x8x8$ is pooled by a 2D pooling kernel with dimension $2x2$ and strides of $2x2$, resulting in the final output feature map with dimensions $3x4x4$ (image taken from Zhou et al.).

cortex of the human brain (Bengio et al., 2009). The different levels in the pyramid architecture corresponds to different levels of abstraction, which enables the network to learn complex representations at each level abstraction with the BP.

Deeper architectures have shown to be more efficient than shallower architectures in solving complex non-linear problems (Goodfellow et al., 2016). The accumulation of linear and non-linear layer transformations enables the network to learn non linear patterns and representations from the input data, and this allows the network to solve complex problems. An example of CNN pyramid architecture is the "LeNet5"network, designed by LeCun et al., 1998 provided in figure 2.4

The commercial application of deep CNNs dates back to 1998, when LeCun et al.'s influential handwritten digit recognition model called "LeNet-5"was effectively deployed by NRC, and was responsible for reading over 10 percent of all the financial paychecks in the United States.

The big boom in commercial and scientific interest happened in the past decade and was initialized by the success of "AlexNet"in the ImageNet Large Scale Visual Recognition Challenge in 2012 (Krizhevsky et al., 2012). AlexNet won the competition by achieving an error rate of 15.3 percent on classifying 1.3 million high resolution images, over a 1000 distinguishable classes, and beat the second place's error rate by a

remarkable 10.8 percent.

The research in the field of CNNs accelerated in the years after and resulted in numerous advancements in in the deep CNN architectures, such as the "ResNet"architecture (He et al., 2016), "Xception"architecture (Chollet, 2017) and the "Inception"architecture (v4) (Szegedy et al., 2017).

By 2015, the CNN architecture of He et al. surpassed the human error rate of 5.1 percent on the ImageNet classification task, by reaching a groundbreaking error rate of 4.94 percent. This was, again, a significant improvement of 26 percent relative to the previous year's winner "GoogLeNet", which had reached an error of 6.66 percent on the ImageNet classification task (Szegedy et al., 2015).

2.3 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are bio-inspired meta-heuristic search and optimization algorithms. EAs mimics biological processes in nature, such as evolution, selection, mutation and recombination, to solve complex computational problems (Back, 1996).

The study of EAs is part of the broader field of Evolutionary Computation, and leading sub fields within EAs are Genetic Programming (Koza, 1992), Genetic Algorithms (Holland, 1975), Evolutionary Strategies (Beyer et al., 2002) and Evolutionary Programming (Fogel et al., 1966). The field of Neuroevolution (section 2.3.2) and Semantic Learning Machine algorithm (section 2.3.3) are part of the field of Evolutionary Computation as well, and will be discussed in this section.

For n generations, an EA evolves a population of individuals, which represent a pool of solutions to a given OP. By applying probabilistic *artificial selection* and *artificial variation* operators on its population, it evolves the pool of solutions towards the best fit solution for the OP at hand (Back, 1996).

Let P be the population of size n, such that $P = \{i_1, i_2, \ldots, i_n\}$, where i_n represents the nth individual in P. Each individual i in P represents a solution s for the subject OP at hand. The artificial selection pressure of the EA will favour the survival of individuals with a better fit solution, but, the selection is a stochastic process, which enables less fitted individuals to have a probability of survival onto the next generation.

A fitness function f is used to score the quality of each solution and determines the probability of survival the individual. Each individual i_n is characterized by its *genotype* s_n - the actual representation of the solution, and its *phenotype* f_n - the fitness score of the solution.

The objective of an EA is to derive an individual that either maximizes or minimizes the fitness function, after n generations. This is done by evolving an individual i_n with the goal to find a particular i_n for which its solution $s_n = s_{global}$. Here, the solution s_{global} maximizes (or minimizes) the fitness function f, and this is in principle an OP. Define the OP of an EA as follows:

$$s_{global} = \arg\max_{s \in S} f(s). \tag{2.18}$$

Besides the stochastic selection procedure, there is also stochastic variation introduced into the population P. At each generation, the genetic material of the selected individuals are recombined and randomly mutated. In this manner, new potential solutions are created and the search space S is explored.

The general procedure of an EA is as follows: Step 1; initialize a starting population $P_{initial}$ and promote $P_{initial}$ to the current population $P_{current}$, Step 2: stochastically select individuals from $P_{current}$ for the next population P_{new}, Step 3: stochastically vary the genotypes of the individuals in P_{new} to create new genetic data and replace $P_{current}$ with P_{new} (Back, 1996). The above steps are continued until a termination criteria is met or until N generations have evolved. Figure 2.6 illustrates an schematic overview of the general evolutionary process of an EA.

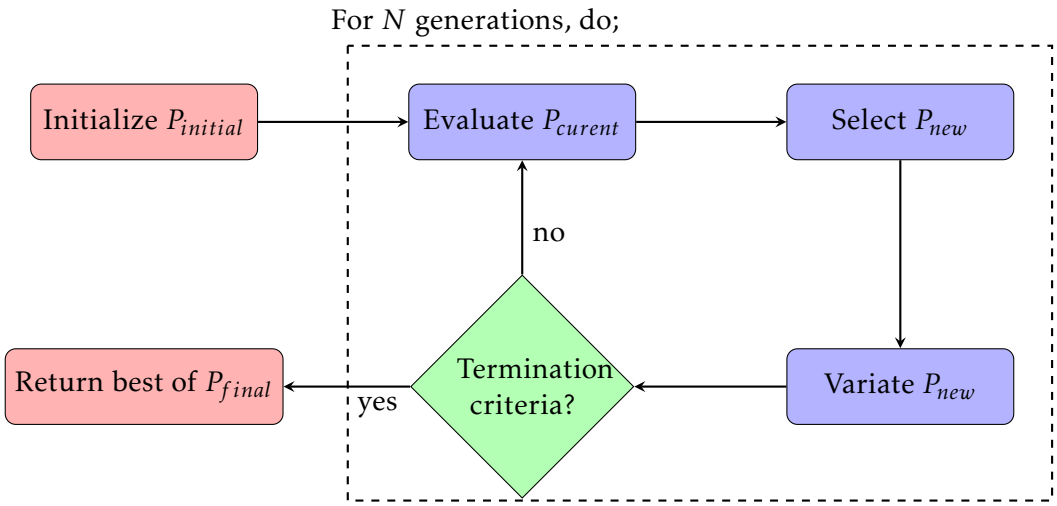

Figure 2.6: Schematic overview of the evolution procedure of an EA (Back, 1996).

2.3.1 Geometric Semantic Genetic Programming

Genetic Programming (GP) (Koza, 1989) is a particular sub-field of EA, in which the genotype of an individual i in the population P defines a computer program or a mathematical function. Accordingly, GP allows the automatic generation and evolution computer code by searching the underlying space of computer programs.

The genetic computer code of a GP can be represented as sequential code instruction (Brameier et al., 2007) or as graphs (Miller et al., 2008), but the most common approach is the syntax tree based GP(Koza, 1992)

The *syntax tree based* GP is constructed from a primitive set of mathematical operators and variables. The primitive set encompass the fundamental building blocks for the genetic computer code in GP. The primitive includes a set of *terminals* - the terminating nodes ("leaves") of syntax tree, and a set of *functions* - the internal nodes of the syntax tree. The function set determines how the sub trees ("branches") of the syntax GP trees will be combined mathematically. Figure 2.7 provides two examples of syntax GP trees constructed from the terminal set $[a, b, c, x]$ and the function set $[+, *]$.

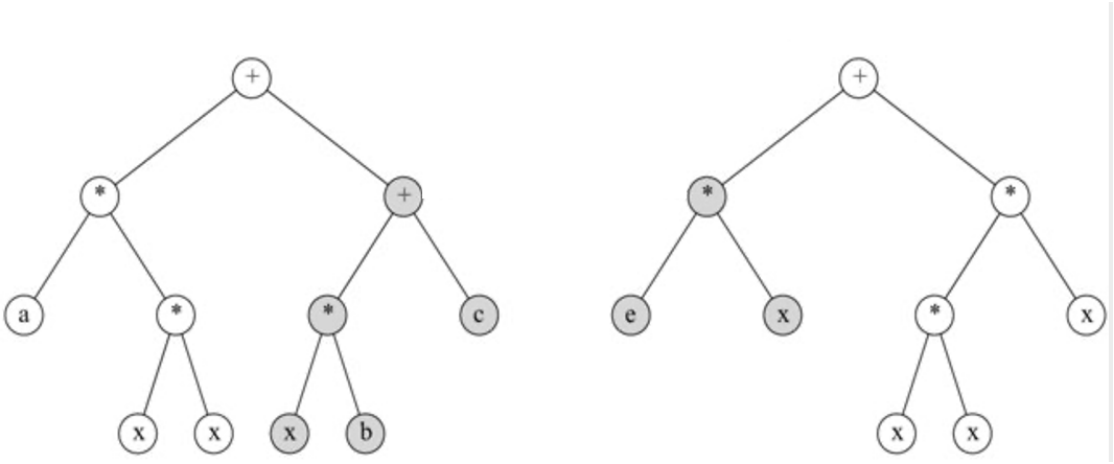

Figure 2.7: Example of two GP syntax trees constructed from the terminal set $[a, b, c, x]$ and the function set $[+, *]$ representing the computer program $a*(x*x)+((x*b)+c)$ (left) and $c*x+(x*x)*x$ (right). (Image taken from Kaufmann, 2013)

Stochastic variation is introduced into the population P by recombination of two individuals ("the mating' process") or by mutating a single individual ("a mutation"). Both operations have an independent probability P_{mut} and P_{crossr} of taking place, at each

generation. These probabilities are also referred to as the *mutation rates* and *crossover rates*.[5]

The semantics of a GP program is the behaviour of the program on the problem it is meant to solve (Moraglio et al., 2012). In a supervised ML setting, the semantics of a GP individual translates to the prediction vector $\hat{\mathbf{Y}}$ that the program generates (Moraglio et al., 2015). Accordingly, each GP individual can be interpreted as a point in the semantic space S by its semantic vector $\hat{\mathbf{Y}}$. Note that the target vector $\mathbf{Y}$ also exists in the same semantic space S.

The distance from any GP individual i to the target vector can be calculated, which results in an error vector $\mathbf{E}_i$ existing in some error space E (Moraglio et al., 2012). Note that, in the origin of the error space E exists an individual E_{origin} for which its semantics $\hat{\mathbf{Y}} = \mathbf{Y}$, which is the global optimal solution for the given supervised ML problem.

The GP variation operators operate directly on the syntactic trees of the GP individuals and naively searches the the underlying functional space of computer programs. The GP operators do not take in consideration their effects in semantic space S and the error space E. A relative small change in the functional space of a GP can have a significant change in the semantic space, and vice-versa. In other words: the resulting offsprings do not have to be related to their parents, in either the underlying functional or semantic space.

This is an inconvenient characteristic of standard GP and results in a rugged genotype $\rightarrow$ phenotype mapping, which can make the search of GP inefficient and limit the possibility of reaching the global optimal solution (Vanneschi, 2017; Moraglio et al., 2012).

To tackle this particular issue, variation operators for standard GP have been designed to include "semantic awareness"into the variation process.[6] Also, operators which produce offsprings more similar to their parents, in both semantic and functional space, have been designed. And finally, this quest has led to the introduction of Geometric Semantic Genetic Programming (GSGP) by Moraglio et al. in 2012.

GSGP is a novel form of GP in which the stochastic variation operators hold specific geometric properties in the semantic space (Moraglio, 2007). The geometric variation

[5] There exist various different strategies on how to perform recombination and mutation and they can have an influence on the performance of GP, the reader can refer to Koza et al., 2008 for more information.

[6] For a detailed literature review on geometric semantic techniques in GP, the reader can refer to Vanneschi et al., 2014 and Vanneschi, 2017 for more information.

operators for GSGP induce their geometric properties onto the semantic space S, if the following two conditions are satisfied:
Condition 1. The problem at hand is a supervised ML problem, where the target semantics satisfy $\mathbf{Y} \in \mathbb{R}^n$
Condition 2. The error of individual i in population P is calculated with some distance metric M between the program's semantics $\hat{\mathbf{Y}}$ and the target semantics $\mathbf{Y}$ (Moraglio, 2007).

By satisfying these two conditions, the Geometric Semantic Crossover (GSC) operator returns an offspring with a better fitness than at least one of its parents. The Geometric Semantic Mutation (GSM) operator returns an offspring with a small perturbation in the semantics of its parent. The distance of this perturbation is controlled with the mutation step (*ms*) parameter of the GSM operator. The formal definitions of GSC and GSM are as follows:

Definition 1. Geometric Semantic Crossover: Given two parent functions T_1 and $T_2 : \mathbb{R}^n \rightarrow \mathbb{R}$, the geometric semantic crossover returns the real function $T_{XO} = T_1 \cdot T_R + (1 - T_R) \cdot T_2$, where T_R is a random real function whose output values falls in the interval $[0, 1]$.

Definition 2. Geometric Semantic Mutation: Given a parent functions $T : \mathbb{R}^n \rightarrow \mathbb{R}$, the geometric semantic mutation with mutation step *ms* returns the real function returns the real function $T_M = T + ms \cdot (T_{R1} - T_{R2})$, where T_{R1} and T_{R2} are random real function.

The GSM operator outputs a mutated syntactic GP tree by linearly combining two independent syntactic GP trees. In definition 2, the resulting mutated tree T_M is a linear combination of the parent tree T with two random trees T_{R1} and T_{R2}. The linear combination is done by weighing the random trees T_{R1} and T_{R2} by some factor, which is the mutation step *ms*.

The important aspect of the geometric semantic mutation operators is that the combined trees are independent. Independent implies that the semantics of one tree does not effected the semantics of the other trees. If this constraint is fulfilled, the semantics of the mutated tree T_M can simply be derived by linearly combining the semantics of the parent tree T and the two random trees T_{R1} and T_{R2}, taking into consideration the weighing factor *ms* (Moraglio et al., 2012).

The interesting property of the GSGP and the geometric semantic variation operators is that they induce a unimodal error error space E. This means; for every point in the

semantic space S, there exists at least a single neighbouring point (individual) with a better fitness value, which can be reached effectively through application of the GSC and GSM operators. This particular individual will be closer in distance to the desired target semantic vector $\mathbf{Y}_{target}$.

This special property eliminates the multimodal nature of error spaces, which are commonly associated with optimization problems and algorithms (section 2.1.3).

By using GSM and GSC operators, GSGP searches directly in the semantic space of computer programs without ever reaching a local optimal solution. This is a result of the induced unimodal and linear nature of the corresponding error space E. This is, by itself, an interesting property in terms of potential search efficiency and effectiveness.

2.3.2 Neuroevolution

The search for the optimal topology and hyperparameters of an ANN for a given supervised ML problem is a difficult task. This optimization comes with an infinitely big search space, as a consequence of the parameters θ, number of neurons and number of layers of an ANN are theoretically not bound by any limit (section 2.2).

Neuroevolution (NE) deals with this kind of optimization by using EAs to search underlying space of the topologies and hyperparameters of an ANN (Stanley et al., 2019). The advantage of NE lays in the fact that the underlying EAs facilitate the optimization over a discontinuous search space, which is a necessity for optimizing the topology and hyperparameters of an ANN. The search space of ANN topologies and the corresponding fitness landscape is undifferentiable and discontinuous, which make numerical optimization methods unsuited for this kind of optimization (Stanley et al., 2019).

Besides this, the performance of ANNs before and after the training phase depend heavily on the initial conditions. As well, slight changes in topology of the network change the semantics of an ANN significantly. All these factors combined result in an irregular, ragged and highly multimodal fitness landscape, which has to be traversed by an OA. These characteristics make EAs and NE an excellent candidate for tackling such OP (Floreano et al., 2008).

Various NE strategies have been proposed since the late 1980s and differentiate in the way how the structural components, the parameters θ and other hyperparameters of the ANN are encoded into the EA. Three fundamental approaches can be identified within the field of NE ; one in which the the NE algorithm is used to only train the ANN

with a fixed topology, second in which the NE algorithm is only used to optimize the topology of the network and trained with another optimization method, an a third in which the NE is applied to both train and optimize the topology of the ANN at the same time, during the evolutionary procedure of the NE (Stanley et al., 2019). [7]

The pioneering work of Miller et al. showed that the bias values and connection weight values of an ANN can be evolved using a EA under a fixed topology structure. Consecutively, in 2002, Stanley et al. showed that evolving the topology and weight connections during the evolutionary process is beneficial and leads competitive performance compared to conventional fixed-topology approach trained with the BP. Stanley et al. achieved these results by proposing a novel NeuroEvolution of Augmenting Topologies (NEAT) algorithm, which is considered as a milestone in the field NE up until today.

After the boom of interest in CNN research in 2014 (section 2.2.4), various attempts to evolve CNN architectures with NE have been reported. In 2017, Real et al. showed that it was possible by applying of various evolutionary techniques to design CNN models for the CIFAR-10 and CIFAR-100 data sets. Real et al. started from trivial initial conditions, but reached up to 94.6% 77.0% accuracy respectively. The authors stressed that no human participation was involved after the initialization of the evolution procedure, and that the output was fully-trained CNN model. The authors did place an emphasis on the repeat-ability and variability of results, and the computational requirements necessary for obtaining them.

In the same year, Desell introduced the Evolutionary Exploration of Augmenting Convolutional Topologies (EXACT) algorithm. The EXACT algorithm was modeled after the NEAT algorithm to extend to CNNs, but contained major differences to enable scaling to a large scale distributed computing system, to speed up the evolutionary process.

Meanwhile, Such et al. demonstrated that it was possible to evolve over 4 millions weights of a deep ANN with a population-based genetic algorithm. The evolved network performed well on various deep reinforcement learning problems and it was the the largest neural network ever evolved with a traditional EA. It expanded the sense of on what scale genetic algorithms can operate successfully.

The NE algorithm of Suganuma et al. automatically constructed CNN architectures based on Cartesian GP, and applied this strategy successfully on the CIFAR-10 image

[7] For an extensive overview on NE techniques and recent trends, the reader can refer to Stanley et al., 2019 and Floreano et al., 2008 for more information

classification task. The experimental results on the CIFAR-10 showed that the proposed Cartesian GP method was competitive with state-of-the-art models for designing CNN architectures.

Starting in 2019, Miikkulainen et al. introduced an extension of NEAT, Coevolution DeepNEAT (CoDeepNEAT), which reported results comparable state-of-the-art benchmarks achieved by human designed networks in multiple fields, including object recognition, language modeling image classification.

Zhu et al. implemented an NE solution based on Artificial Bee Colony (ABC) for automatic design of CNN topologies. Results reported on the MNIST dataset proved competitiveness with the state-of-the-art NE techniques for the MNIST dataset.

Consecutively, Badan et al. introduced EA4CNN (Evolutionary Algorithms for Convolutional Neural Networks) - an EAwhich evolves and optimizes the CNN architecture with respect to both the classification error and the model complexity. This was done in an attempt to avoid deriving extremely big and complicated models to reduce training time. The complexity was expressed as the number of tuneable parameters in the CNN.

Sun et al. proposed a new method using a genetic algorithm to evolve CNN architectures and corresponding connection weight initialization distributions. A variable-length gene encoding strategy was used, together with a new representation scheme for the initialization of the connection weights, as well as a novel fitness evaluation method to reduce computational resources. The proposed algorithm was compared against 22 existing algorithms on 9 image classification tasks and showed competitiveness against state-of-the-art algorithms in terms of classification error and number of parameters.

Additionally, Sun et al. proposed a genetic algorithm designing CNN architectures based on ResNet and DenseNet blocks. The genetic algorithm was evaluated on the CIFAR-10 and CIFAR-100 benchmark data sets, against 18 state-of-the-art peer contestants. The proposed algorithm outperformed the manually designed state-of-the-art CNNs and CNNs designed by automatic peer contestants. The evaluation was done in terms of the classification performance on CIFAR-10 and CIFAR-100 data sets.

Important to note is that all the above mentioned NE approaches, with the exception of Such et al., 2017, used the conventional training approach with the Backpropagation algorithm with the underlying Gradient Descent method to optimize the parameters θ of the evolved CNN models. All the presented scientific papers used extensive computational resources to train hundreds up to thousands of CNNs on distributed systems. There has not been a single method reported, up until today, which allows for evolution of the CNN topology, parameters θ and other hyperparameters, without the use of

Gradient Descent and the Backpropagation algorithm. Also, Such et al. worked with a fixed topology CNN and only evolved the weights the network.

2.3.3 Semantic Learning Machine

The Semantic Learning Machine (SLM) is a novel NE algorithm, recently proposed by Gonçalves et al., which constructs fully functioning feedforward NNs without the use of the Backpropagation algorithm. By applying the GSM operator (definition 2) on NNs, the SLM functions as a stochastic hill-climbing algorithm in the search space of NNs. The SLM's evolutionary procedure is therefore based upon the principles of of GSGP and accordingly, the geometric semantic properties defined by Moraglio apply to the stochastic search of the SLM.

As described in section 2.3.1, the geometric semantic properties imply that the SLM searches over the search space of feedforward NN topologies and hyperparameters, under a unimodal error landscape, and thereby eliminating the presence of any local optimal solutions. This makes the SLM more attractive in terms of search effectiveness and efficiency.

Section 2.3.1 introduced the GSM operator for GP, which resulted in the definition of GSGP. In definition 2, the GSM operator outputs a mutated syntactic Genetic Programming (GP) tree by linearly combining two independent syntactic GP trees. This concept of the GSM operator, including its geometric semantic properties, can be extended to feedforward NNs, in which the syntactic GP trees are replaced by feedforward NNs.

In this use case, the GSM operator is applied on a given parent feedforward NN and adds to the parent network structure a randomly initialized feedforward NN. The random feedforward NN only receives incoming connections from the parent network, and the random feedforward NN cannot feed its activations back into the parent's network hidden nodes. The output neurons of the parent network receive the last hidden layer activations of the random network and linearly combines the output activations with its current existing output activations. The weights for the last hidden layer nodes of the random network is then defined by the mutation step *ms*. The other weights in mutated part of the feedforward NN are initialized at random. If this architectural design is followed, the semantics of the parent network will not be affected by the semantics of the random network, which results in an independent combination of networks and GSGP's constraint of independently combining networks and their corresponding semantics are fulfilled (Gonçalves et al., 2015c).

To provide a practical example of the GSM operator on a NN, a single mutation of the GSM operator is performed on the the parent network T_{parent}, as presented in figure 2.2. This NN consists of three input neurons, two hidden layers with each two hidden neurons and a single output neuron. A single application of the GSM operator adds a randomly initialized network T_{random} to the parent network T_{parent}. Here, in figure 2.8, the green nodes with green incoming connections represent the added T_{random} network and the dotted black dotted connections represent the existing T_{parent} network connections. The added network T_{random} consists of two added nodes in the first hidden layer and a single hidden node in the second hidden layer. As can be observed from figure 2.8, the internal activations of T_{random} do no feed into the hidden nodes of the parent network T_{parent}, with exception of the output neuron, which is shared between the T_{parent} and T_{random} networks. The weights of the last hidden layer of the random network T_{random} are defined by the mutation step ms.

The SLM was introduced by Gonçalves et al. in 2015, in which the algorithm was tested on several real-life multidimensional symbolic regression datasets. This study showed that the SLM was able to outperform GSGP in terms of learning the training data, but did not statistically improve over GSGP on the test data.

Consecutively, Jagusch et al. showed that various SLM variants outperform other NE approaches in terms of learning the training and test data in 9 real-world regression and classification datasets. Even the best performing SLM variant outperformed various topologies of a feedforward NN trained with the backpropagation-based approach. In this research, the SLM was also used as a base estimator to build an ensemble estimator which proved to outperform the Random Forest algorithm in two classification problems. This work clearly showed the diversity and potential of the SLM.

The SLM was also used to evolve the NCP of a state-of-the-art CNN and was tested against a state-of-the-art CNN trained with the conventional Backpropagation approach (Lapa et al., 2019). The performance was assessed on the PROSTATEx dataset composed of multispectral MRI sequences for the detection of prostate cancer. This study showed that CNN evolved with the SLM was outperforming the the state-of-the-art in terms of Area Under the Receiver Operating Characteristic (AUROC) curve performance. Besides increased performance, there was a 14 time computational speed up reported for the SLM procedure.

Also Teixeira reported outperforming results of the NE approach with the SLM compared to a conventional Backpropagation approach with NNs with a common parametrization framework for equal comparison between the two methods.

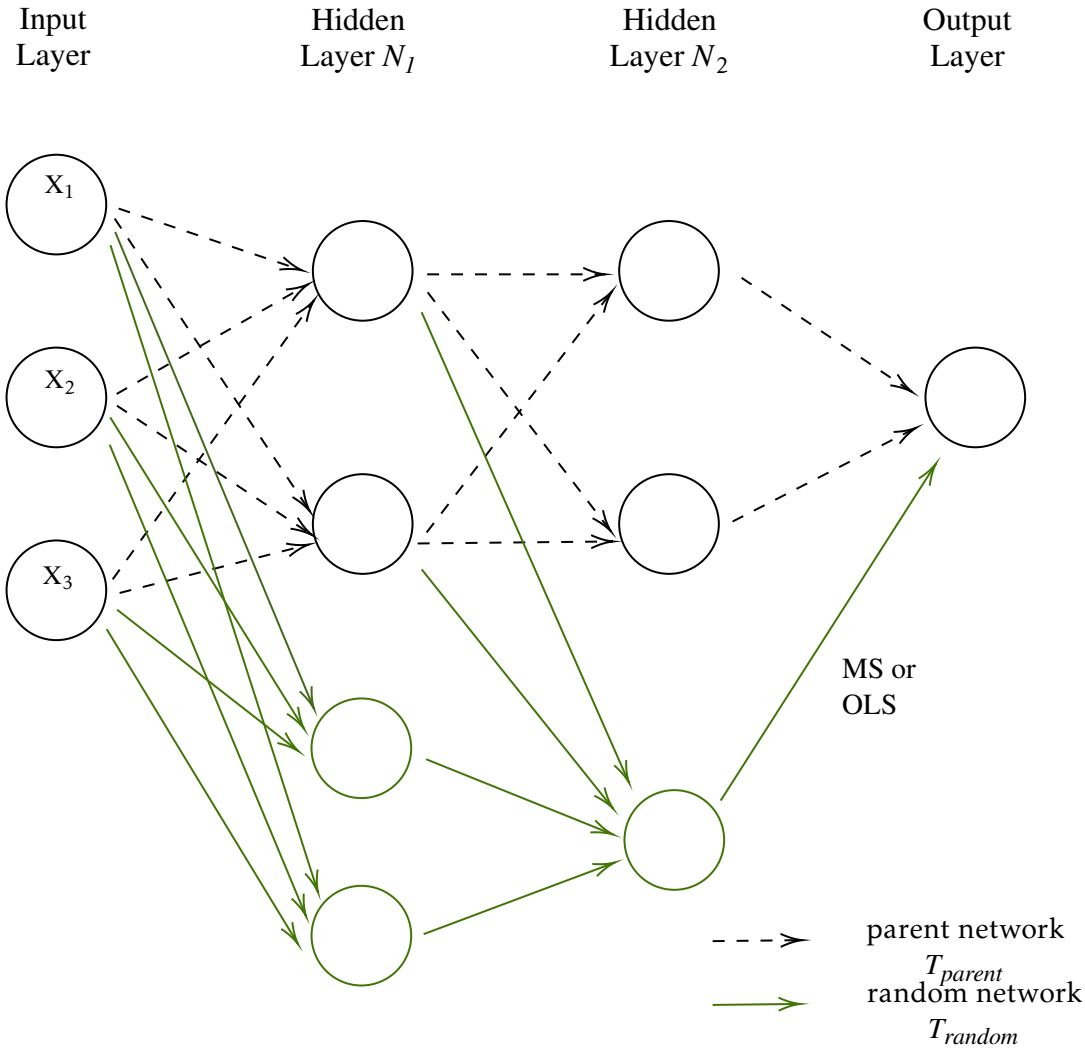

Figure 2.8: Overview of the application of the GSM operator on the MLP network of Figure 2.2 by adding two hidden neurons to the first hidden layer and a single hidden neuron to the second hidden layer (green labelled connections).

Finally, Gonçalves et al. studied how dynamic subsets of the training data can be used to improve generalization ability of the SLM . Over fifteen real-world binary classification data sets, the dynamic use of training data resulted in superior generalization of the SLM. As well, the SLM outperformed, with statistically significance, the conventional MLP network approach in thirteen of the fifteen studied datasets. Furthermore, the work showed that the stochastic nature of the SLM introduces sufficient diversity into the base learner, wherefore an averaging ensemble approach with the SLM as base

learners is an effective technique to improve performance, with respect to other ensemble approaches such as Bagging and Boosting.

2.4 Deep Semantic Learning Machine

This section introduces Deep Semantic Learning Machine (Deep-SLM) algorithm. The Deep-SLM extends the principles of the SLM algorithm (section 2.3.3) to the search space of CNN topologies and parameters. The Deep-SLM is a stochastic search and CNN constructing algorithm, which functions as a stochastic hill climbing algorithm operating in the search space of CNN topologies and parameters. The Deep-SLM automatically builds and evolves CNNs by applying the GSM operator (definition 2) on existing CNNs in its evolution process.

The Deep-SLM originates from the elementary SLM algorithm, which is the Deep-SLM's analog for the search space of NNs. Therefore, the Deep-SLM also comes forth out of the principles of GSGP (section 2.3.1) and shares the same geometric semantic properties. From the geometric semantic properties, the most appealing one is; for any given supervised ML problem, the Deep-SLM executes its search over a unimodel error space and eliminates the presence of any local optimal solutions.

Another notable property of the Deep-SLM is that it derives fully functioning CNNs without the use of the Backpropagation algorithm. These properties make the Deep-SLM potentially favourable and more effective in terms of search effectiveness and efficiency.

A formal description of the Deep-SLM algorithm is formulated in this section by organising the section as follows: section 2.4.1 extends the GSM operator to CNNs, section 2.4.2 provides a description and pseudo code Deep-SLM algorithm, section 2.4.3 discusses Adaptive and Optimal Learning Step of the Deep-SLM and finally, section 2.4.4 discusses the code implementation of the Deep-SLM.

2.4.1 Geometric Semantic Mutation For CNNs

The GSM operator defined in definition 2 can be extended to apply to any given CNN. The GSM operation starts from any given CNN - T_{parent} - and linearly combines T_{parent} with a randomly initialized CNN - T_{random}. The addition of the T_{random} to T_{parent} is weighted by some weighting factor ms - the mutation step. In this manner, at every application of the GSM operator, a new CNN T_M is produced.

To induce the geometric semantic properties defined by Moraglio et al. described in section 2.3.3, the linear combination of CNNs has to be independent. Therefore, no connections from T_{random} to T_{parent} can be formed, exclusively connections from T_{parent} to T_{random}. The semantics of T_{parent} are simply obtained by linearly combining the semantics of T_{random} and the semantics of T_{parent} weighted by the ms.

To clarify the GSM procedure, a practical example of a single GSM application on a CNN is provided in figure 2.9. The single application of the GSM is performed on a parent CNN T_{parent} with the following structure: a single 2D CL with 10 kernels, with kernel dimensions of 5x5 and stride 1x1, followed by a 2D Max PL with kernel dimensions of 2x2 and stride 2x2, and finally a single FCL with 50 nodes with an output layer consisting of 10 nodes. The input into this network consists of a data volume of 32x32x3 (which could represent a 32x32 RGB colour image) and the connections of this parent network T_{parent} are indicated with the black dotted arrows in figure 2.9.

Then, the GSM operator for CNNs adds random nodes to each layer and these connections are indicated with the green arrows. To the first 2D CL, 2 kernels are added, which are down sampled by the 2D Max PL and then fed into the single FCL layer. The single FCL has been extended with 20 nodes and the resulting output activation of this layer are fed into the final output nodes of the network. In figure 2.9, the new formed connections of the T_{random} network are indicated with the green arrows.

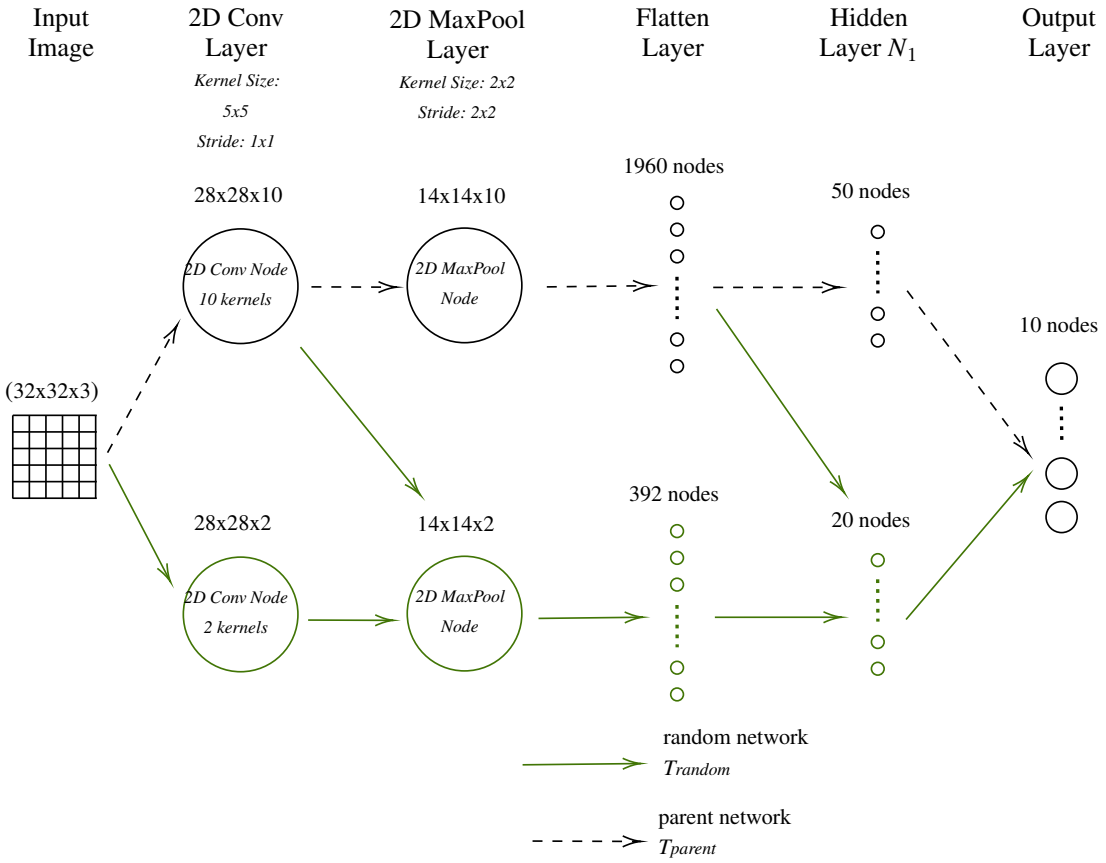

Figure 2.9: Single application of the GSM operator of the Deep-SLM on a T_{parent} network consisting of a single 2D CL , 2D Max PL and a single PL. The black dotted arrows represent the connections of T_{parent} and green arrows represent the connections of T_{random}

2.4.2 The Deep-SLM algorithm

The pseudo code of the Deep-SLM algorithm is provided in algorithm 2. Here, P_0 represents the initial population of CNNs and the starting point Deep-SLM's search. The CNN T_{best} represents the current best individual at iteration i and the CNN T_i represents the ith neighbor of the T_{best}.

The hyper parameters that have to be defined for the initialization of the Deep-SLM algorithm are presented and clarified in table 2.1. Certain hyperparameters should be defined for both the Convolution Part (CP) and Non-Convolution Part (NCP) of the Deep-SLM, these parameters include: *max nodes*, *max layers*, *probability add nodes*, *probability add layers*, *sparseness*, *skip-connections*, *node activations*.

Algorithm 2: Deep Semantic Learning Machine

```
initialize Deep-SLM parameters (table 2.1);
initialize P_0;
set T_best;
iteration i = 0;
while i < i_max do
    Apply n time the GSM to T_best to generate neighbourhood N of T_best;
    for T_i ∈ N do
        evaluate T_i;
        if fitness T_i > fitness T_best then
            T_best = T_i;
        else
            continue;
        end
    end
end
```

Table 2.1 introduces two new parameters of the Deep-SLM (which also apply to the SLM but not discussed in section 2.3.3); the *sparseness* and *skip-connections* parameters. The sparseness of a given neuron i defines the sparsity of the incoming connections of that neuron. Neuron i receiving all possible incoming connections is said to have a sparseness of 0%. If all weights of all incoming connections are set to a value of 0, which means all incoming connections are inactive, then, the sparseness of that neuron is said to be 100%. Define the sparseness for a given neuron i as follows:

$$Sparseness = \frac{N_{\ connections\ where\ w \neq 0}}{N_{\ all\ possible\ connections}}\ x\ 100\%/ \tag{2.19}$$

Skip-connections parameter for a given neuron i in the layer m_i works according to same principle as the sparseness parameter, but, it applies only to the incoming connections from neurons in the layers m_j for which $m_j \leq m_{i-2}$. For a given neuron i in the layer m_i, which receives only connections from the neurons in the previous layer m_{i-1}, the skip-connections parameter is said to be 0%. If the given neuron i receives connections from all previous neurons in the network, the skip-connections parameter for neuron i is said to be 100%. Define the skip-connections parameter in % for a given

neuron i as follows:

$$Skip\ Connections = \frac{N_{\ incoming\ connections(layers\ M_j \leq M_{i-2})}}{N_{\ all\ possible\ connections(layers\ M_j \leq M_{i-2})}}\ x\ 100\%. \tag{2.20}$$

Parameters:	**Data type:**	**Clarification:**
number iterations	*int*	*number of iterations of the evolution procedure*
neighbourhood size	*int*	*size of the neighborhood to be explored at each iteration*
max layers	*int*	*max number of layers to be added in the mutation*
max nodes	*int*	*max number of nodes to be added in the mutation*
mutation step	*float*	*size of the mutation step*
probability add nodes	*float* $[0 \leq x \leq 1]$	*probability of adding nodes in the mutation*
probability add layers	*float* $[0 \leq x \leq 1]$	*probability of adding layers in the mutation*
sparseness	*float* $[0 \leq x \leq 1]$	*max fraction of sparse connections of a node*
skip-connections	*float* $[0 \leq x \leq 1]$	*max fraction of skip-connections of a node*
conv kernel dim	*(int, int)*	*range of the 2D convolution kernel dimensions*
conv strides	*(int, int)*	*range of the 2D convolution strides*
pool kernel dim	*(int, int)*	*range of the 2D pooling kernel dimensions*
pool strides	*(int, int)*	*range of the 2D pooling strides*
node activation	*function*	*the activation function to be used in the node*
weight initialization	*distribution*	*distribution for weight initialization*
bias initialization	*distribution*	*distribution for bias initialization*

Table 2.1: List of hyperparameter definitions for the Deep-SLM. Parameters that apply to both the CP and NCP of the Deep-SLM are: *max nodes*, *max layers*, *probability add nodes*, *probability add layers*, *sparseness*, *skip-connections*, *node activations*.

2.4.3 Adaptive Learning Step

In the GSM application, the mutation step - *ms* - determines the size of the random perturbation in the semantic space of the T_{parent} network. In other words, the *ms* the size of the n dimensional ball mutation when thinking of the GSM as a ball mutation in the semantic space of T_{parent} (Vanneschi, 2017).

The random perturbation in the semantics of T_{parent} can be bounded or unbounded. This depends on the random individual T_{random} in the GSM operation being bounded by a bounding function or not. For example, in GSGP, if the random tree is bounded by a logistic function, which maps the output in the interval $[0,1]$, then the maximum perturbation in the semantics is given by the interval $[-ms, ms]$.

The *ms* can also be optimally computed for each application of the GSM operator and is referred to as the Adaptive Mutation (AM) (Moraglio et al., 2013). The linear combination obtained from the application of the GSM operator can be written as equation 2.21, in which *RI* represents the random tree or network T_{random}, t the target vector and P the parent tree or network T_{parent}

$$P + RI \cdot ms = t, \tag{2.21}$$

with the objective to reach the target semantic vector t. Equation 2.21 can be rewritten to

$$RI \cdot ms = (t - P), \tag{2.22}$$

which satisfies the condition of a general linear system

$$A \cdot x = y. \tag{2.23}$$

Thus, the linear system of equation 2.23 can be solved deterministically using the Moore-Penrose pseudo inverse.

Also, the parent tree or network P can be weighted by another factor resulting in a little adjustment of equation 2.21 to equation 2.24

$$P \cdot w_p + RI \cdot ms = t. \tag{2.24}$$

This is referred to as Double Adaptive Mutation (Gonçalves et al., 2015a). Gonçalves et al. experimented for the first time with the adaptive mutation step on real world multidimensional data sets.

The linear system in equation 2.23 can be solved with a computational method and results in a solution x. The solution x is the mutation step *ms* in equation 2.22 for that particular application of the GSM operator.

Consider the the application of the GSM for CNNs, the semantics of the last hidden layer of the randomly generated CNN - T_{random} - is considered as *RI* in the the linear system of equation 2.22, then, the solution of this linear system - *ms* - represent the values for weights of the connections of the last hidden layer of T_{random} to the final output nodes of T_{parent}.

This means that, with every application of the GSM for CNNs, the optimal weights of the last hidden layer of T_{random} to the output nodes of T_{parent} can be calculated by solving the linear system of equation 2.22.

The fact that the last hidden layer connection weights (and theoretically all the weights of the network) can be optimally computed in the GSM for CNNs, is an interesting property of the GSM operator which could make the Adaptive Mutation Step a favourable property of the Deep-SLM.

2.4.4 Code Implementation

For experiments conducted in this thesis described in chapter 4, the Deep-SLM was implemented in the Python Programming language. The source code can be downloaded from: $https://github.com/olehofman/deepSLM\_src$.

The CP of the Deep-SLM was implemented with support of the Tensorflow library and the NCP of the Deep-SLM was implemented with support of the Numpy library.

Due to limited computational resources, only a specific variant of the Deep-SLM was used in the experimental study. This variant only forms $parent\ node \rightarrow mutation\ node$ connections in the CP of the CNNs constructed. In the NCP of the CNNs constructed, only $mutation\ node \rightarrow mutation\ node$ connection were formed. Therefore, the final CNNs have a fully connected CP structure, whilst, a sparse NCP structure. The motivation for this design was the lack of Random-access Memory (RAM) caused by the nature of the GSM operator.

The GSM operator linearly combines two networks producing an offspring network (definition 2). This offspring network will always be bigger in terms of size compared to its parent network T_{parent}.

To derive the semantics of the offspring network, the internal node activations of the parent network T_{parent} are required, if the offspring network is fully-connected between all layers. When only the mutation nodes are connected to previous mutation nodes, only the activations of the previous mutation nodes are required for calculating the semantics of the new offspring network. Therefore, it was decided to only connect the mutation nodes in the NCP internally. With this implementation, it was achieved to perform significant longer runs without running into RAM storage issues.

Chapter 3

Experimental Methodology

This chapter presents the experimental methodology for the conducted research in this thesis. This includes: a description of dataset (section 3.1), a validation of the metrics (section 3.2) and an explanation of the adapted experimental framework to tackle the research objectives.

Chapter 2 introduced the relevant theory and background for this thesis. Here, the Deep-SLM algorithm was introduced by extending the principles of the SLM to the search space of CNNs. The Deep-SLM is therefore a novel NE algorithm which has never been validated in a scientific experiment before, which has led in chapter 1 to the establishment of main research objective of this thesis:

Research Objective *Validation of the proof of concept of the Deep-SLM*

3.1 Data sets

To qualitatively asses performance the Deep-SLM algorithm and to directly compare the Deep-SLM algorithm's performance against the performance of the SLM algorithm, one needs to decide on an appropriate problem setting first. This is, in essence, the dataset (or sets) on which the performance of the algorithms will be evaluated in the various benchmark experiments. A higher number of datasets and a higher variety between

them is favourable and allow for a more fair and valid assessment and comparison of the two algorithms.

The Canadian Institute For Advanced Research 10 (CIFAR-10) dataset was selected to be used to benchmark the performance of the Deep-SLM and SLM algorithms (https:// www.cs.toronto.edu/ kriz/cifar.html). The CIFAR-10 dataset is one of the most commonly used datasets for Computer Vision benchmarks experiments and therefore is a valid choose to validate the proof of concept of the Deep-SLM.

The CIFAR-10 dataset is a balanced dataset consisting of 60000 32x32 labelled colour images of each 10 classes, with each class having exactly 6000 images. By default the dataset is divided in a training set of 50000 training images and a test set of 10000 test images. All the classes are listed in Figure 3.1 and are completely mutually exclusive, there exist no overlap between the classes. Figure 3.1 displays an image sample per class of the CIFAR-10 dataset.

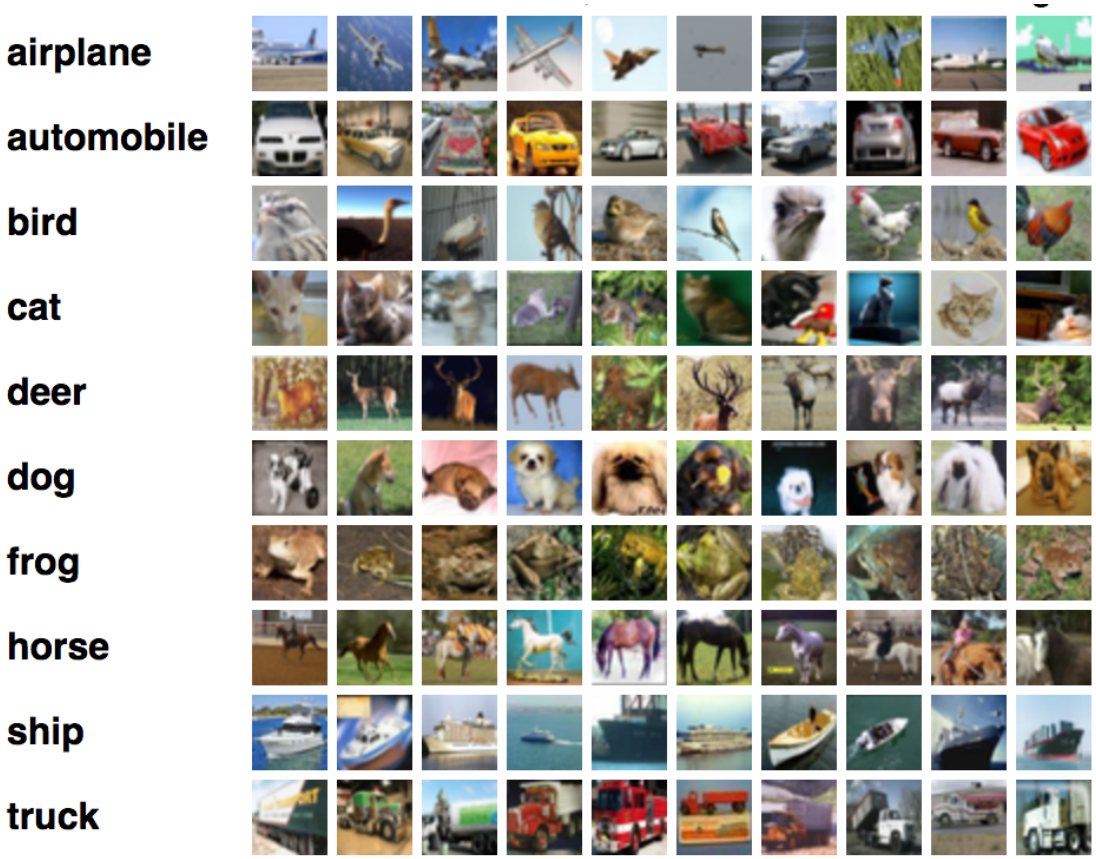

Figure 3.1: Sample illustration of the 10 classes of the CIFAR-10 dataset (taken from https://www.cs.toronto.edu/ kriz/cifar.html)

3.2 Metrics

For the performance assessment of the Deep-SLM and the comparative benchmarks between the Deep-SLM and the slm algorithms, an appropriate metric needs to be chosen which assigns a quantitative scores to each benchmark run. This metric scores quality of the performance of each specific run under the specific set of experimental conditions used in that run.

The appropriate metric is dependent on the problem setting at hand, which is in this case a *balanced* image classification task. Also, important to note is that every error in the algorithm's prediction is of equal importance, as opposed to certain other image classification applications. For example, in cancer detection methods, a *false negative* prediction is an error which should be avoided at all times, whilst a *false positive* is an

error of less importance.

In the CIFAR-10 problem setting, all errors are from equal importance and the dataset is balanced. Thus, the CIFAR-10 dataset allows for the use of the most conventional performance metric in classification problem setting - the *accuracy*.

The accuracy metric simply provides the percentage of correctly classified classes in a classification task. Equation 3.1 provides the formula of the accuracy for multi-classification setting, in which T_p represents *True Positive* and N the number of total instances. The accuracy is usually given in terms of percentage where multiplied with a factor of 100.

$$Accuracy\,\% = \frac{T_p}{N}\, x\, 100\,\% \tag{3.1}$$

Also, the Cross-Entropy (CE) loss will be monitored during the benchmark experiments. The CE loss is the objective function minimized by the LBFGS-algorithm and is, therefore, relevant to be monitored during the evolution process. The formula for multiclass CE loss is given in equation 3.2, in which C represents all the classes of the multi-classification task, $f_{output}(s)_i$ the output probability of the model for the class C_i and t_i the corresponding target

$$CE\ Loss = -\sum_{i}^{C} t_i \cdot log(f_{output}(s)_i) \tag{3.2}$$

3.3 Experimental Framework

An experimental framework was designed to tackle the research objective presented in chapter 1. Since the Deep-SLM has not been tested before in a scientific experiment, the experimental framework will consist of exploratory parts. Each individual exploratory part is then followed up by a more comprehensive analysis of the selected hyperparameter settings for the SLM and Deep-SLM algorithms and includes statistical assessing.

In the exploratory parts, only the behaviour of the Deep-SLM and SLM under various hyperparameter settings will be explored. The objective of the exploratory parts is not to provide a comprehensive analysis of the hyperparameters and their effect, but rather to get an understanding of the behaviour of the algorithms on the CIFAR-10 dataset. In these exploratory experiments, only a single run will be used due to limited computational resources and due to the exploratory objective.

The exploratory parts are then followed up by a more profound analysis, in which statistical tests will be performed on a selected set of hyperparameters for both the Deep-SLM and SLM algorithms. The statistical tests will be conducted as follows: first, the obtained scores will be tested for normality with the non-parametric Kolmogorov-Smirnov Test, consecutively, the parametric Unpaired Students T-test or the non-parametric Mann-Whitney U will be used to analyse for potential differences between expected values of the score means of the sample scores. These experiments will be run using 30 runs each.

Framework

1. *Define the SLM baseline performance on the CIFAR-10 dataset*
    - Explore the baseline SLM's performance and generalization ability on CIFAR-10
    - Grid search various NCP structures of the SLM to establish the baseline SLM performance (done with 1 run)
2. *Explore possible improvements of the baseline SLM's performance and generalization ability on the CIFAR-10 dataset*
    - Perform hyperparameter tuning of the SLM to investigate if performance and generalization ability can be improved over the baseline SLM performer (done with 1 run)
    - Validate the potential improvement by assessing the results for a statistical significant improvement (done with 30 runs)
3. *Explore the effects of a 2D CL in the CP of the Deep-SLM on CIFAR-10 dataset*
    - Grid search various 2D convolution kernel sizes and stride values to explore the effect of a 2D CL as single random feature generation in the CP of the Deep-SLM (done with 1 run)
    - Validate the performance and generalization ability by assessing the results for a statistical significant difference (done with 30 runs)
4. *Explore possible improvements in performance and generalization the single 2D CL in the Deep-SLM's CP*

- Perform hyperparameter tuning to the 2D CL of the Deep-SLM to investigate if the performance and generalization ability can be improved (exploration done with 1 run)
- Validate the potential improvement of the performance and generalization ability by assessing the results for a statistical significant difference (done with 30 runs)

5. *Explore the effect of a 2D max-PL with the 2D CL in the CP of the Deep-SLM*

   - Grid search various 2D max-pooling kernel dimensions and stride values to explore the effect of the down sampling layer with the 2D CL in the CP of the Deep-SLM (exploration done with 1 run)
   - Validate the potential improvement in performance and generalization ability by assessing the results for a statistical significant difference (done with 30 runs)

Chapter 4

Results & Discussion

This chapter presents the experimental results obtained from the research conducted in this thesis. This includes a presentation of the result of the initial exploration phase, and, for a selected set of experiments, a deeper statistical analysis. The research conducted in this thesis was based upon the earlier defined experimental framework in section 3.3.

For each conducted experiment in this chapter, the used hyperparameter settings and methods are shortly discussed. Consecutively, the obtained results are presented and discussed. Note that in all following experiments, a variant of the Deep-SLM was used as explained in section 2.4.4. This variant only forms *parent node* → *mutation node* connections in the CP of the CNNs constructed. In the NCP of the CNNs constructed, only *mutation node* → *mutation node* connection are formed.

4.1 Defining the baseline SLM performance

The establishment of the SLM baseline performance on the CIFAR-10 dataset was done by grid searching various (and randomly selected) NCP structures of the SLM . The NCP structures which were included in the grid search were; [20], [50], [200], [400], [100, 50] and [200, 50] respectively. Here, the first number represents the number of nodes in the NCP first hidden layer. The second number represents the nodes in the second (and last) hidden layer. The + sign in the respective label (Figure 4.1) represents the number of nodes added to to each hidden layer at mutation respectively.

Weight values and bias values of the NCP connections were initialized using a random uniform distribution within the interval [-0.1, +0.1]. The the weight values and bias values of the last hidden layer connecting to the output layer were optimally computed using the LBFGS-numerical optimization algorithm. The CE Loss was used as the objective function to be minimized for the optimization. The numerical optimization was run for 50 iterations with default settings of the LBFGS-algorithm.

The results obtained from the grid search have been achieved with a single run, no statistical significant differences between the runs were assessed. Table 4.1 summarizes the default settings used in the following benchmarks.

Table 4.1: Default parameter settings of the SLM

Parameter:	
LBFGS Iterations	*50*
Weight Initialization	*RandomUniform [-0.1 <x <+0.1]*
Bias Initialization	*RandomUniform [-0.1 <x <+0.1]*
Sparseness	*0%*
Mutation Configuration	*Only previous mutation nodes*

4.1.1 Results

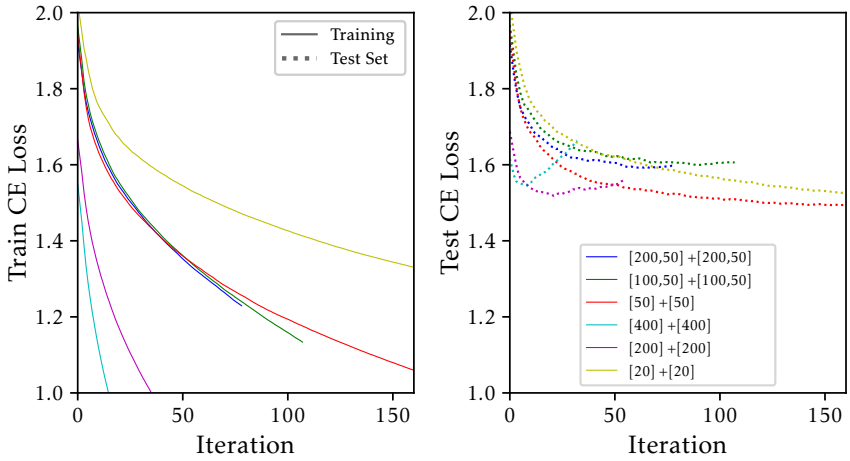

Figure 4.1: SLM performance on CIFAR-10 problem task with various NCP structures. The label $[X, X]$ represents the initialization of NCP structure. $+[X, X]$ represents the number of nodes added to each layer at each mutation. More nodes in the final hidden layer tends to show more overfitting. Deeper NCP structures did not have any beneficial effect.

4.1.2 Discussion

Figure 4.1 displays the results obtained from the grid search of the NCP structures of the SLM on the CIFAR-10 dataset. Figure 4.1 shows that all variants of the baseline SLM are capable of learning the training data, but overfit clearly on test data. All tested structures show a converging Test CE Loss around 1.5 and start overfitting afterwards.

There is a clear trend that indicates a higher number of nodes in the last hidden layer of the SLM results into more overfitting. More nodes in the final hidden layer of the NCP results in more degrees of freedom when solving the numerical optimization problem in the OLS calculation in the GSM operation. This could be an explanation for the more extreme and quicker overfitting trend.

Additionally, multiple hidden layers (deeper structures) do not seem to have a positive effect on the performance of the SLM. The application of the OLS could be an explanation for this behaviour as well. For this reasons, the single hidden layer SLM with +50 nodes added at each mutation was chosen as a baseline performer in the future benchmarks with the Deep-SLM .

4.2 Improving the baseline SLM performance

To improve the SLM baseline performance established in section 4.1, the effect of applying sparse connections in the NCP of the SLM was investigated. In addition to the SLM baseline performer, the SLM variant having 100 nodes in last hidden layer and adding +100 nodes at each mutation was included in the exploratory analysis.

The SLM with 0%, 50% and 99% sparseness in the NCP were tested and compared against the baseline SLM performance established in section 4.1. The experimental settings for the SLM were the same as explained in section 4.1 and table **??**. The effect of the sparse connections was explored using a single run exploration (figure 4.2).

Consecutively, the best performing variant was selected and benchmarked for 30 runs to asses for a statistical significant improvement over the the baseline SLM performance. This procedure was done by first testing for normality with non-parametric Kolmogorov-Smirnov Test, and consecutively, a non-parametric rank-based Mann-Whitney U Test was performed to asses for statistical significant differences between the sample means of both performance scores of the two SLM hyperparameter variants (figure 4.3).

4.2.1 Results

The results obtained from the experiments are visualised in figure 4.2 and 4.3.

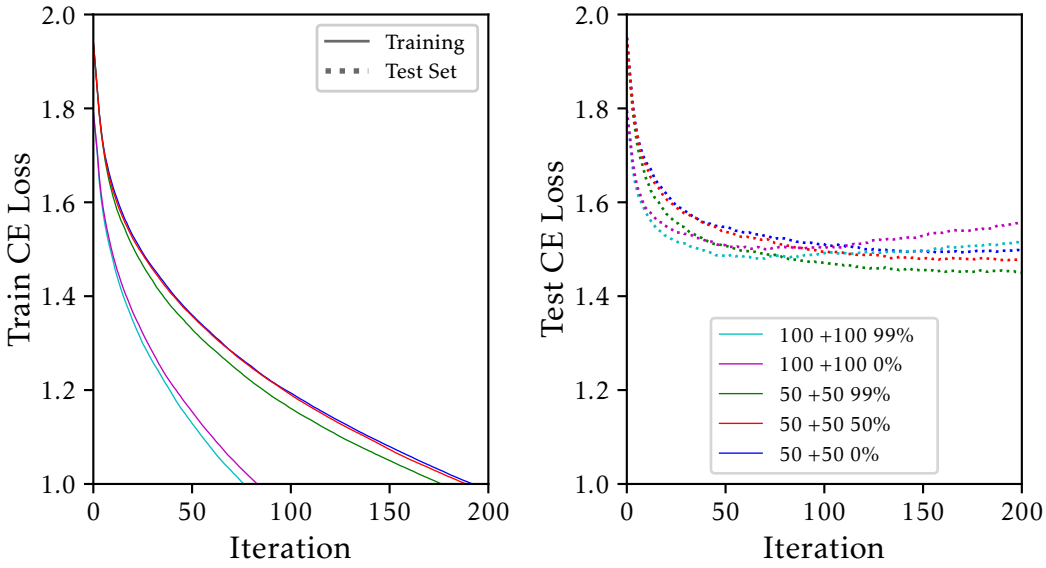

Figure 4.2: SLM performance on CIFAR-10 problem task with various degrees of sparseness. The same label structure of figure 4.1 applies to this figure. Additionally, the last % indicates the percentage of sparseness applied to the SLM. A higher percentage (50% and 99%) of sparseness tends to reduce overfitting in combination with fewer nodes (50).

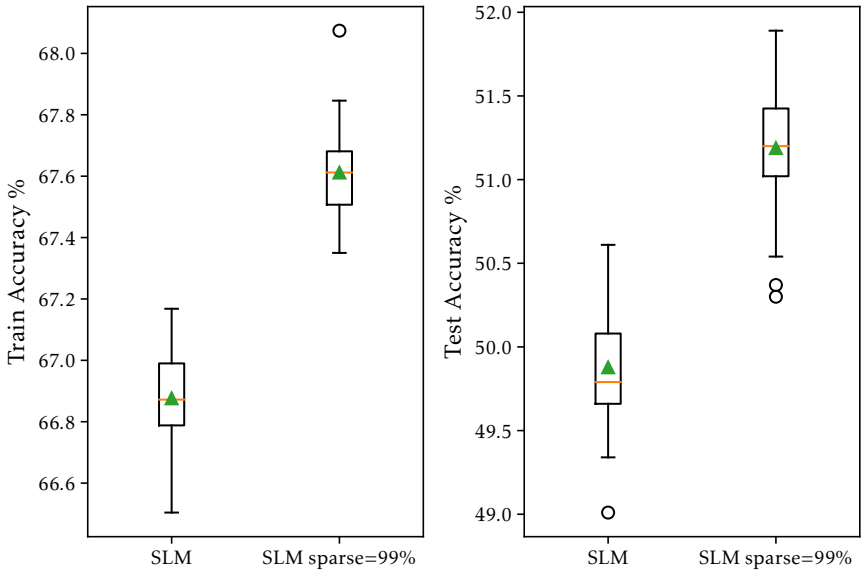

Figure 4.3: Box plot visualisation of the Train Set Accuracy and Test Set Accuracy of the SLM with 99% sparseness and without sparseness on the CIFAR-10 problem task. The experiment was done using 30 runs and the expected mean accuracies were found to be significantly different.

4.2.2 Discussion

Figure 4.2 displays the results of the exploratory study of the use of sparse connections in the NCP in the SLM . A more stable convergence of the Test CE Loss can be observed when a higher percentage of sparseness (50% and 99%) is applied to the NCP of the SLM. The SLM variants adding +50 nodes in each mutation, with percentage of sparseness of 50% and 99%, results in the lowest Test CE Loss of around 1.45. Therefore, a hypothesis can be formulated that applying sparse connections to the standard SLM can help reduce overfitting.

To asses if these improvements are indeed statistically significant, the SLM variants of [+50] with 99% sparseness and no sparseness were benchmarked for 30 runs. The resulting scores are visualized in the box plots in figure 4.3. The mean of the training set accuracies were 66.8% and 67.6% respectively. The test set accuracies resulted to be 49.8% and 51.2% respectively.

To validate if these values come from a normal distribution, the Kolmogorov-Smirnov test was performed. The p-values obtained from this test were respectively 4.45e-69 and 7.68e-69, and thus, suggesting that the alternative hypothesis, which states that the scores are not normally distributed, cannot be rejected. In this test, a significance level α of 0.05 was considered.

Consecutively, the non-parametric Mann-Whitney U test was performed to evaluate if the both score samples have equal means, and if the alternative hypothesis, which states that the SLM with no sparseness has a lower expected sample mean than the SLM with 99% sparseness, could be rejected. The p-value returned from this Mann-Whitney U test was 1.83e-11, and thus the alternative hypothesis could not be rejected, indicating that the SLM variant with 99% sparseness outperforms the baseline SLM performer with statistical significance on the test set accuracy metric. Also in this test, a significance level α of 0.05 was considered.

4.3 Exploration of the 2D convolution layer for the Deep-SLM

In this experiment, the application of a single 2D CL in the CP of the Deep-SLM on the CIFAR-10 dataset was explored. Various 2D CL configurations were tested and the accuracy and CE loss were monitored. This exploration was done using a single run.

At each generation, the GSM operator adds a 2D convolution node to the single CL in the CP of the Deep-SLM. The convolution kernels are initialized with random kernel

values wherefore the new convolution node will produce random convolved feature maps. The random convolved feature maps form the input activations for the NCP of the Deep-SLM. Like this, the random convolution nodes function as random feature transformers for the NCP of the Deep-SLM.

The NCP of the Deep-SLM consists of a single hidden layer with 50 nodes and in each mutation 50 new nodes are added to the single hidden layer. Therefore, NCP structure is equivalent to the previous experiments with the SLM in sections 4.1 and 4.2.

In the following experiments, the convolution kernel dimensions and number of kernels were varied. The kernel dimensions explored were (2,2), (5,5), (10,10) and (32,32) respectively. The different number of kernels were 1, 10, 25 and 100 respectively. The stride value of the 2D CL was kept constant at (1,1). Note that a higher number of kernels result in higher number feature maps, and thus, in a higher number of flattened input connections into the NCP. For the kernel dimensions, a bigger kernel induces a smaller feature maps and thus, less flattened nodes feeding their activations into the NCP of the Deep-SLM.

The convolution kernel values were initialized with a random uniform distribution within the interval [-0.1, +0.1]. The random convolved feature maps were fed directly into the single layer NCP with +50 nodes added to the single hidden layer. The weight connection values of the single hidden layer to the output nodes of the Deep-SLM were optimized by the LBFGS-algorithm. The default Deep-SLM parameters are displayed in Table 4.2. As well in this exploration, the effect of using sparse connections in the NCP of the Deep-SLM was explored. Various degrees of sparseness were tested, which were 0%, 50% and 99% respectively. This exploration was also done using a single run.

4.3.1 Results

The results obtained from the experiments are visualised in figure 4.5 and 4.4.

Table 4.2: Default parameter settings of the Deep-SLM

Parameter:	
LBFGS Iterations	*50*
NCP structure	*single fully connected layer*
NCP n nodes init	*50*
NCP n nodes mut	*+50*
NCP Weight Initialization	*RandomUniform [-0.1 <x <+0.1]*
NCP Bias Initialization	*RandomUniform [-0.1 <x <+0.1]*
NCP Sparseness	*0%*
NCP Mutation Configuration	*Only previous mutation nodes*
CP structure	*single 2D Convolution Layer (stride=(1,1))*
CP Weight Initialization	*RandomUniform [-0.1 <x <+0.1]*
CP Bias Initialization	*RandomUniform [-0.1 <x <+0.1]*
CP Sparseness	*0%*
CP Mutation Configuration	*Only previous mutation nodes*

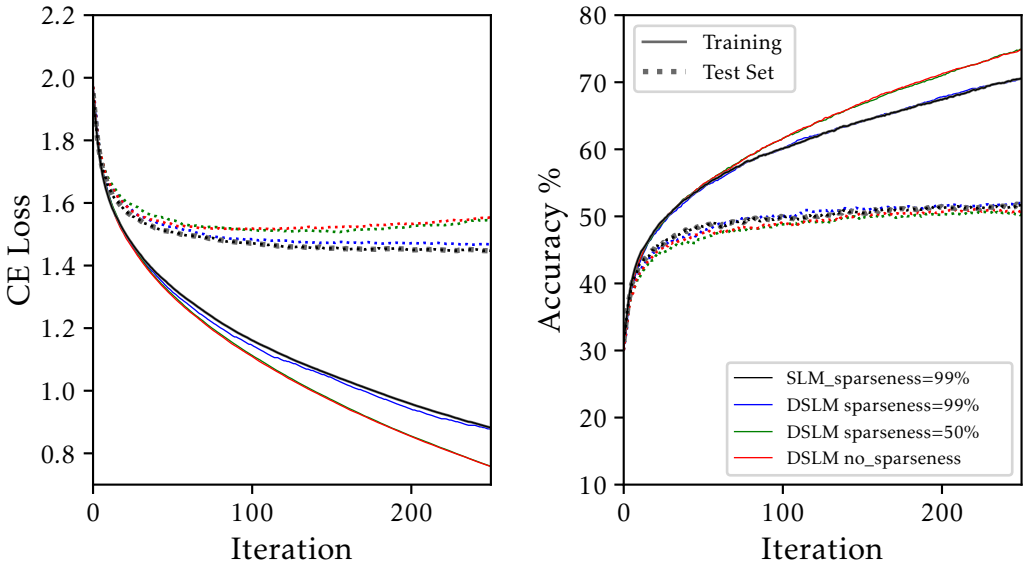

Figure 4.4: Deep-SLM benchmark on CIFAR-10 problem task. CP contains a single 2D CL with kernel dimensions=(5,5) and various percentages of sparseness applied to the NCP. Black curve represents the baseline SLM performance. Sparseness id not improve the performance or generalization ability of the Deep-SLM, as opposed to the SLM

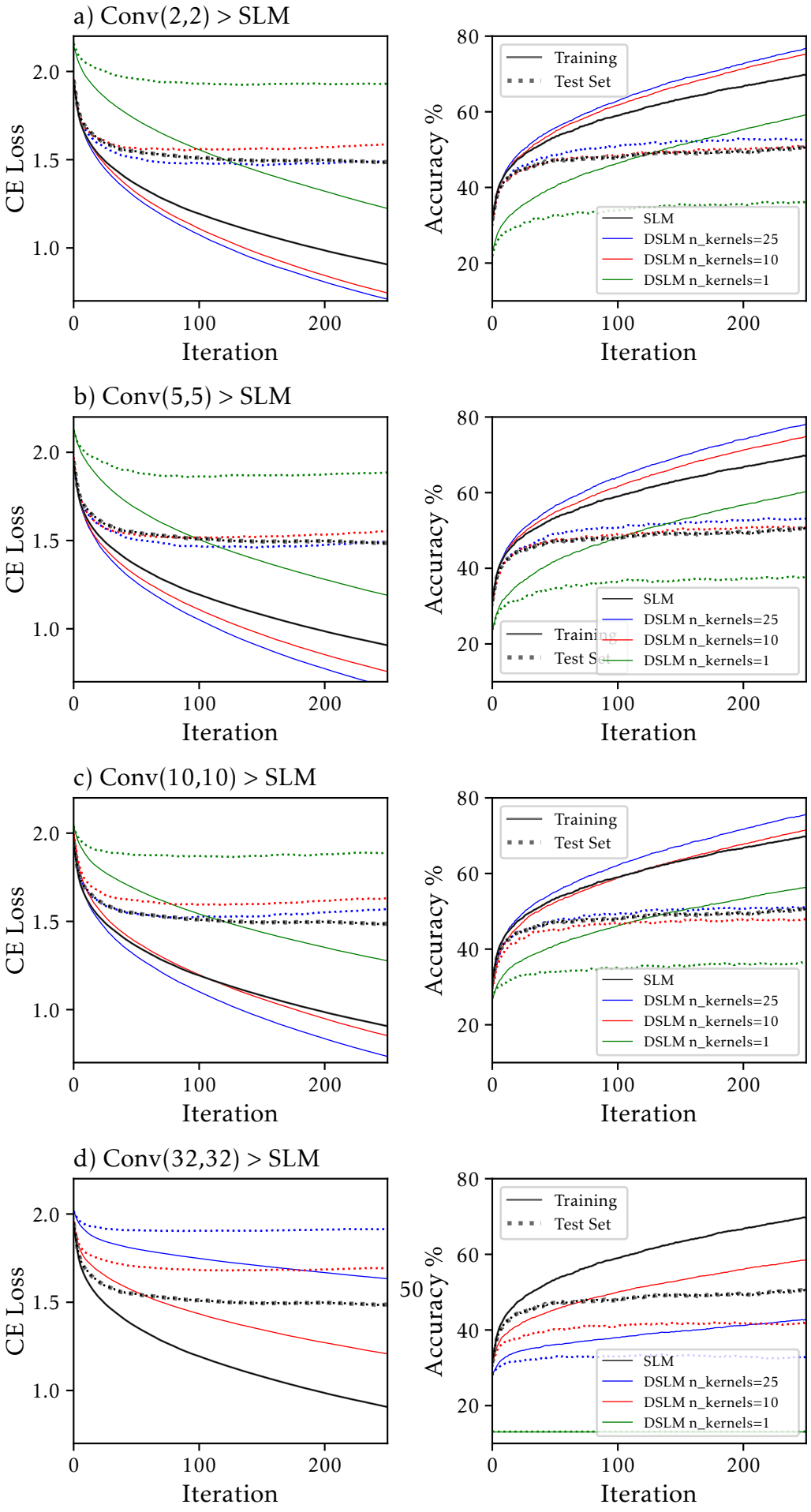

Figure 4.5: Deep-SLM benchmarks on CIFAR-10 problem task with kernel dimensions a) (2,2), b) (5,5), c) (10,10) and d) (32,32) with various number of kernels with constant stride (1,1). Black curve is the SLM baseline reference. Kernel dimension does not seem to have significant effect on performance of the Deep-SLM. Higher number kernels increase learning rate but not significantly improve overfitting.

4.3.2 Discussion

Figure 4.5 displays the results obtained from the exploratory analysis of using a single 2D CL in the CP of the Deep-SLM.

From the observations in this figure 4.5, it seems that the use of a single 2D CL, initialized with a random kernel values, did not improve the performance of the Deep-SLM. The black curve in figure 4.5 represents the baseline SLM benchmark performer and in all evolution graphs, the performance of baseline SLM performer was not exceeded, or only to a very limited extend. The CE loss and accuracy of all tested variants show minor to no improvements over the baseline SLM (black curve).

Different convolution kernel dimensions did not seem to have an effect on the performance of the Deep-SLM . Increasing the kernel dimensions (increasing the number of weights in the convolution kernels) did not seem to have any effect as well. The maximum kernel dimension possible for the CIFAR-10 dataset (32x32), which results in a single pixel feature map, is highly unfavourable (figure 4.5d) and did not allow for learning either the training data or test data.

Adding a higher number of convolution kernels (10, 25) in each mutation step will result in more feature maps in that mutation. This does seem to have a minor effect on generalization ability and performance of the Deep-SLM. More random feature maps also induce more input connections feeding into NCP of the Deep-SLM, although this effect of is minimal.

From this exploratory analysis, the hypothesis can be proposed stating that the random weight initialization for a 2D CL does not produce relevant random transformations (random feature maps), which generalize well to both the training set data and test set data. Potentially, an approach with optimizing the weights of the convolution nodes could be a solution to overcome this problem. But, in this case, the advantage of deriving fully function CNNs without the use of a numerical optimization method, would be lost.

Besides the exploration of a single 2D CL in the CP of the Deep-SLM , applying sparse connections to the NCP of the Deep-SLM was also explored. Figure 4.4 shows the results obtained from this analysis.

As previously observed in the SLM baseline, sparseness effectively improved performance and generalization ability of the SLM (figure 4.3). For the Deep-SLM, this effect can be observed, to a minimal extend, as well when analysing the evolution of the CE loss on the test set in figure 4.4. Both variants of the SLM and Deep-SLM with 99%

sparseness show a more stable convergence of the the Test CE loss, whilst the Deep-SLM variants with less sparseness start to increase their Test CE loss scores when the evolution is continued for more generations. These observations did not translate to accuracy metric.

Nevertheless, applying sparse connections to the NCP of the Deep-SLM did not seem to have any influence on the performance of the Deep-SLM. Note that due to the 2D CL in the CP of the Deep-SLM, the number of input connections feeding into the NCP is smaller as compared to the SLM variants. This could be a potential explanation for the absence of the positive effect when applying sparse connections in the Deep-SLM.

4.4 Exploration of 2D max-pooling layer for the Deep-SLM

In addition to section 4.3, the Deep-SLM was explored by adding a 2D max-PL to the CP of the Deep-SLM. By adding a 2D max-PL, the Deep-SLM's CP structure consisted of a single 2D CL followed up by a 2D max-PL.

The 2D CL randomly convolves the CIFAR-10 input data and produces random feature maps. Consecutively, the 2D max-PL down samples the random feature maps and feeds the activations into the NCP of the Deep-SLM. Here, likewise as all the other previous benchmarks, a single hidden layer in the NCP consisting of 50 nodes was used in the Deep-SLM.

The effect of down sampling of the random feature maps by the 2D max-PL was evaluated by testing two variants of the PL, whilst keeping the 2D CL parameters constant. Two different kernel dimensions and stride values of the max-PL were tested, which were respectively (2,2) and (4,4) for the kernel dimensions, and (2,2) and (4,4) for stride values. Note that the effect of pooling with bigger kernel dimensions and strides results in more down sampled features with smaller dimensions. This induces *"more general, less detailed and precise"* representation of the original random convolved feature maps.

The parameters of the 2D CL layer were kept constant and were the following; kernel dimension=(5,5), n_kernels=10, stride=(1,1). The same parameter settings as displayed in table 4.2 were used in the following benchmark experiments.

Additionally, the effect of using sparse connections in the NCP of the Deep-SLM was explored. This was done by applying 50% and 99% sparseness to the NCP of the Deep-SLM. These variants were also compared against the variants without sparseness and these exploratory benchmarks were performed using a single run (figure 4.6).

For all the following benchmarks, standard NCP and CP parameter settings were used as reported in table 2.1.

In a final experiment, the 2D max-PL with kernel dimensions=(4,4) and stride=(4,4) was benchmarked over 30 runs to asses for a statistical significant improvement (figure 4.7 and 4.8). This procedure consisted of first assessing the score samples for normality with non-parametric Kolmogorov-Smirnov Test, and secondly, the non-parametric rank-based Mann-Whitney U Test to asses for a statistical significant difference between the expected means of the sample scores.

4.4.1 Results

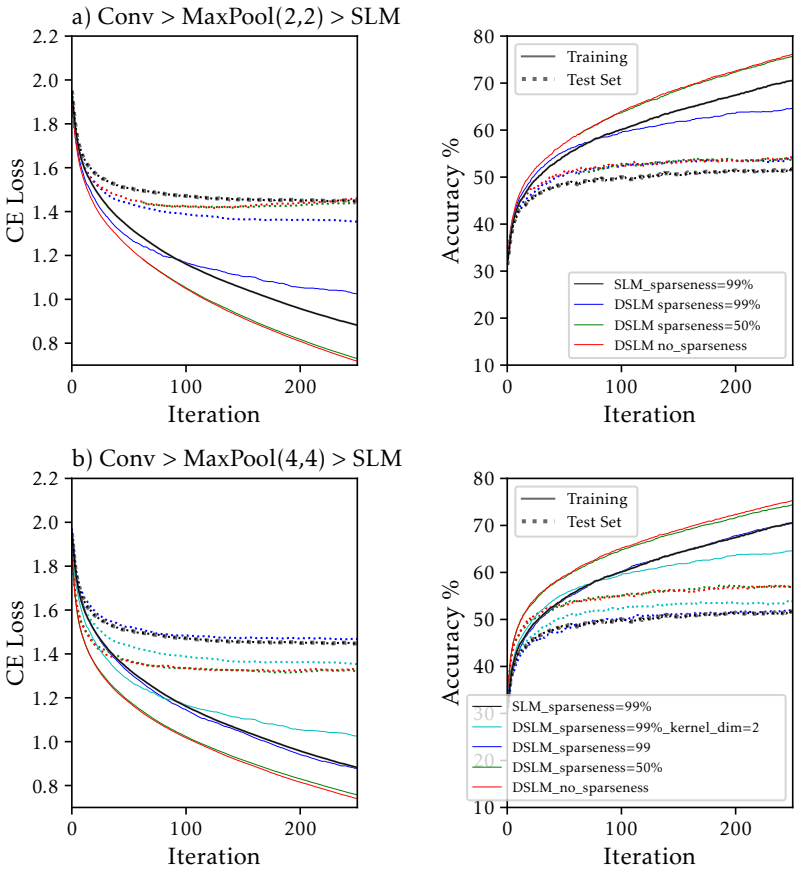

Figure 4.6: 2D max-PL with kernel dimensions (2,2) (a) and kernel dimensions (4,4) (b) with various percentages of sparseness applied the NCP of the Deep-SLM on the CIFAR-10 problem task. A bigger receptive field of the 2D max-PL seems to have a beneficial effect on the performance of the Deep-SLM

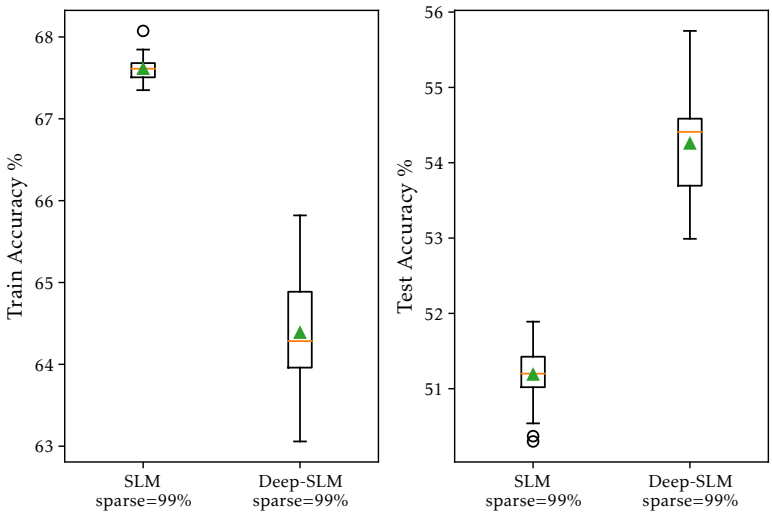

Figure 4.7: Box plot representation of baseline SLM performance and the Deep-SLM with 2D max-PL with kernel dimensions (2,2) and 99% sparseness on the CIFAR-10 problem task. The experiment was run with 30 runs. The 2D max-PL in combination with the 2D CL improves the performance of the Deep-SLM by increasing Test Set Accuracy from 51.5% to 54.2%. The results were found to be statistically significant.

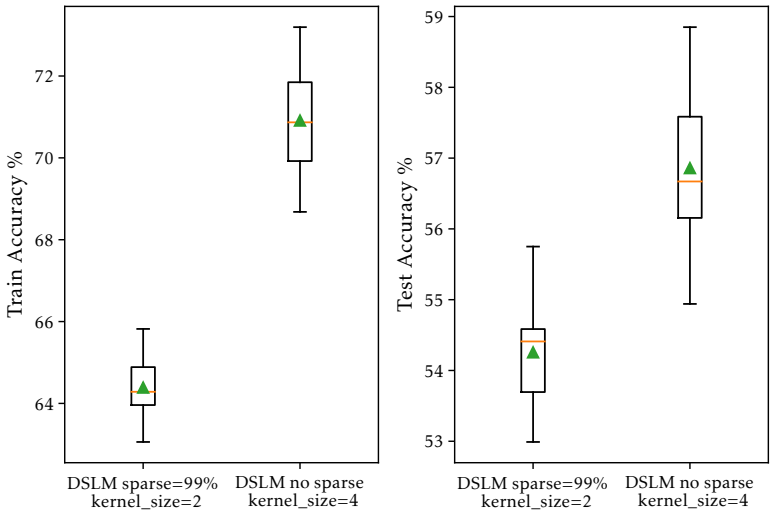

Figure 4.8: Box plot visualisation of the Deep-SLM with 2D max-PL with kernel dimensions of (2,2) and 99% sparseness and kernel dimensions of (4,4) with no sparseness. The experiment was performed on the CIFAR-10 problem task and was run with 30 runs. A bigger receptive field of the 2D max-PL improves the performance of the Deep-SLM by increasing test set accuracy from 54.2% to 57.2%. The results were found to be statistically significant.

4.4.2 Discussion

Figure 4.6 displays the results obtained from the exploration of the Deep-SLM with the 2D max-PL in the CP on the CIFAR-10 dataset. Figure 4.6 (a) shows the effect of using a smaller receptive field and stride in the 2D max-PL. Figure 4.6 (b) shows the effect of using a bigger receptive field and stride in the down sampling layer.

Figure 4.6 (a) shows that the use of a 2D max-PL with a 2D CL in the CP of the Deep-SLM does not seem to have an effect on the performance metrics (red and green curves figure 4.6 (a)), but, when sparseness is applied to the NCP, the CE loss on the test set is decreased from 1.45 to 1.40, and, the accuracy on the test set is increased 51.5% to 54.2% (blue curve figure 4.6 (a)).

To validate the improvements of the 2D max-PL, the Deep-SLM variant with 99% sparseness was benchmarked over 30 runs and the Kolmogorov-Smirnov test was performed to test the sample for normality, and consecutively, the rank-based Mann-Whitney U Test was used to asses for a statistical significant difference with the expected mean performance of the baseline SLM performer with 99% sparseness.

The results of this analysis are shown in figure 4.7 in box plot format. The Kolmogorov-Smirnov p-values obtained for both score samples were respectively 7.68e-69 and 3.52e-695, and thus, suggesting that the alternative hypothesis, which states that both scores are not normally distributed, cannot be rejected. In this test, a significance level α of 0.05 was considered. Consecutively, the p-values obtained from the ranked-based Mann-Whitney U Test was 7.00e-12, which suggest that the alternative hypothesis, which states that the SLM with 99% sparseness has a lower expected mean of the sample score than the Deep-SLM variant with 99% sparseness, cannot be rejected either. This indicates that the Deep-SLM variant with 99% sparseness outperforms the standard SLM with 99% sparseness, with statistical significance on the test set accuracy metric.

In figure 4.6 (b), the effect of a bigger receptive field and stride value in the 2D max-PL was explored. What can be observed from figure 4.6(b)) is that a bigger receptive field and stride value, in combination with a less sparseness (green curve) or no sparseness (red curve) applied to the NCP, decreases the Test CE loss to 1.37, and increases Test Set Accuracy up to 57.2%.

These observations are in contrary with the findings in figure 4.6 (a), in which a higher amount of sparseness in the NCP had a favourable effect on the performance of the Deep-SLM .

Nevertheless, increasing the amount of down sampling of the randomly convolved feature maps seems to have a favourable effect on the performance of the Deep-SLM , under the constraint that the other hyperparameters are properly tuned.

The 2D max-PL seems to increase generalization ability and performance of the Deep-SLM on the CIFAR-10 dataset. Note that in the configuration of the 2D max-pooling operation there exist no weights that can be optimized, as opposed to the 2D CL, which consists of tuneable weights. This property could support the hypothesis that randomly sampling the weights of a 2D CL does not result in the generation of *useful and well-generalizing random features* for the Deep-SLM. This is opposed to a non-tuneable down sampling component, which does seem to have a favourable effect on generalization ability and performance of the Deep-SLM .

To asses for the statistical significant improvements of these results, the Deep-SLM variant with pooling kernel dimensions of (4,4) was tested for 30 runs and the Kolmogorov-Smirnov test was used to asses the score samples for normality. Consecutively, the rank-based Mann-Whitney Test was used to asses if the scores have the same expected mean as the Deep-SLM variant with 99% sparseness and pooling kernel dimensions of (2,2), and, if the alternative hypothesis, which states that the expected mean accuracy score on the test set of the Deep-SLM variant with 99% sparseness and pooling kernel dimensions of (2,2) is lower than the the expected mean accuracy score on test set of the Deep-SLM variant with pooling kernel dimensions of (4,4) and no sparseness, can be rejected.

The results of this analysis are summarised in box plots in figure 4.8. The p-values returned from the Kolmogorov-Smirnov test were 3.52e-65 and 1.39e-47 respectively. This indicates that th sample means are not normally distributed. The p-value returned from the rank-based Mann-Whitney U Test was 2.01e-11, indicating that the expected means of test set accuracy scores of both variants are different, and that the alternative hypothesis, stating that the expected mean value of the Deep-SLM variant with 99% sparseness with pooling kernel dimensions of (2,2) is lower than the variant with pooling kernel dimensions of (4,4), cannot be rejected.

Chapter 5

Conclusion

The aim of this thesis was to explore the Deep-SLM algorithm on the CIFAR-10 problem task, and to provide a validation of its proof of concept. To accomplish this objective, a comparative benchmark study between the Deep-SLM and SLM algorithms was performed, in which the behaviour of the Deep-SLM algorithm was studied. The accuracy and CE loss metrics were selected as the performance metrics for the experiments and based on these metrics, both algorithms were explored, analysed and compared against each other in a disquisition.

Due to limited computational resources, it was only succeeded to test a variant of the Deep-SLM algorithm. This variant only constructs *parent node* $\rightarrow$ *mutation node* connections in the CP of the CNN. In the NCP of the CNN, it forms connections exclusively between the mutation nodes, which results in the formation of sparse network structures in the NCP of the CNN. Further research must be conducted to investigate the limitations of this design, and whether the findings and conclusions in this thesis still hold if fully connected layers are formed throughout the complete network structure.

The SLM baseline performance on the CIFAR-10 problem task was established by grid-searching various NCP topologies. Here, it was observed that deeper NCP topologies did not improve the performance of the SLM. It was also observed that a higher number of nodes added to the last hidden layer of the NCP increased the learning rate of the training set, but resulted in more overfitting on the test set. The SLM structure consisting of a single hidden layer and adding +50 nodes in each mutation was decided

as the baseline SLM performer to be used in future comparisons. The test set accuracy was established at 49.8%. Applying sparse connections improved the baseline performance of the SLM to a limited extent. A higher percentage of sparseness was found to be more effective compared to a lower percentage. The SLM variant with 99.2% sparseness improved the test set accuracy to 51.2%, which was found to be statistically significant compared the SLM variant with no sparseness.

A single 2D CL in the CP of the Deep-SLM was added and explored. For a valid comparison, the Deep-SLM's NCP structure was kept equivalent to the SLM baseline structure. The convolution kernels of the single 2D CL were initialized with random kernel weights. A variety of kernel dimensions and stride values were tested, but none of these improved the performance significantly over the SLM baseline. Also, sparseness applied to the NCP the Deep-SLM did not improve the performance. A small effect was observed with an increased number of feature maps (the number of kernels) added in each mutation. This resulted also in an increased number of input dimensions into the NCP, which could be an explanation for the improved performance. All in all, these observations point in the direction that random sampling of the weights of convolution kernels will not provide useful transformations for the Deep-SLM. The weights have to be optimized beforehand, to produce well generalizing features for the Deep-SLM.

The effect of adding a 2D max-PL to the CP of the Deep-SLM was explored and various pooling kernel dimensions and stride values were tested. This experiment showed that a pooling layer can improve the performance of the Deep-SLM, unless it is properly tuned. Max-pooling kernels with kernel dimensions of (2,2) and 99.2% sparseness applied to the NCP increased, with statistically significance, the test set accuracy up to 54.2%. Bigger kernel dimensions of (4,4) without sparseness in the NCP increased test set accuracy up to 57.2%. This improvement was also found to be statistically significant. The improvements with PLs, which have zero tun-able degrees of freedom, support the hypothesis that convolutional or pooling operations can improve generalization of the Deep-SLM, unless the components are properly tuned.

All in all, this work has shown that the performance of the Deep-SLM can be improved with a simple 2D convolution - pooling structure in the CP of the Deep-SLM. Nevertheless, it should be acknowledged that the performance achieved with the Deep-SLM at this moment is not compatible with the state-of-the-art performance of CNNs trained with the conventional BP approach.

Bibliography

Anwar, S. M., M. Majid, A. Qayyum, M. Awais, M. Alnowami, and M. K. Khan (2018). "Medical image analysis using convolutional neural networks: a review." In: *Journal of medical systems* 42.11, p. 226.

Back, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press.

Badan, F. and L. Sekanina (2019). "Optimizing Convolutional Neural Networks for Embedded Systems by Means of Neuroevolution." In: *International Conference on Theory and Practice of Natural Computing*. Springer, pp. 109–121.

Bengio, Y. et al. (2009). "Learning deep architectures for AI." In: *Foundations and trends® in Machine Learning* 2.1, pp. 1–127.

Beyer, H.-G. and H.-P. Schwefel (2002). "Evolution strategies–A comprehensive introduction." In: *Natural computing* 1.1, pp. 3–52.

Bojarski, M., D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. (2016). "End to end learning for self-driving cars." In: *arXiv preprint arXiv:1604.07316*.

Brameier, M. F. and W. Banzhaf (2007). *Linear genetic programming*. Springer Science & Business Media.

Chollet, F. (2017). "Xception: Deep learning with depthwise separable convolutions." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258.

Desell, T. (2017). "Large scale evolution of convolutional neural networks using volunteer computing." In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 127–128.

Floreano, D., P. Dürr, and C. Mattiussi (2008). "Neuroevolution: from architectures to learning." In: *Evolutionary intelligence* 1.1, pp. 47–62.

Fogel, L. J., A. J. Owens, and M. J. Walsh (1966). "Artificial intelligence through simulated evolution." In:

Gonçalves, I., M. Seca, and M. Castelli (2020). "Explorations of the Semantic Learning Machine Neuroevolution Algorithm: Dynamic Training Data Use, Ensemble Construction Methods, and Deep Learning Perspectives." In: *Genetic Programming Theory and Practice XVII*. Springer, pp. 39–62.

Gonçalves, I., S. Silva, and C. M. Fonseca (2015a). "On the generalization ability of geometric semantic genetic programming." In: *European Conference on Genetic Programming*. Springer, pp. 41–52.

— (2015b). "Semantic learning machine: a feedforward neural network construction algorithm inspired by geometric semantic genetic programming." In: *Portuguese Conference on Artificial Intelligence*. Springer, pp. 280–285.

— (2015c). "Semantic learning machine: a feedforward neural network construction algorithm inspired by geometric semantic genetic programming." In: *Portuguese Conference on Artificial Intelligence*. Springer, pp. 280–285.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press.

Gori, M. and A. Tesi (1992). "On The Problem Of Local Minima In Backpropagation." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14, pp. 76–86.

Hartman, E. J., J. D. Keeler, and J. M. Kowalski (1990). "Layered neural networks with Gaussian hidden units as universal approximations." In: *Neural computation* 2.2, pp. 210–215.

He, K., X. Zhang, S. Ren, and J. Sun (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

— (2016). "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Holland, J. (1975). "Adaptation in natural and artificial systems: an introductory analysis with application to biology." In: *Control and artificial intelligence*.

Holt, N. J., A. Bremner, E. Sutherland, M. Vliek, M. Passer, and R. Smith (2012). *Psychology: The science of mind and behaviour*. McGraw-Hill Education.

Jagusch, J.-B., I. Gonçalves, and M. Castelli (2018). "Neuroevolution under unimodal error landscapes: an exploration of the semantic learning machine algorithm." In: *Proceedings of the genetic and evolutionary computation conference companion*, pp. 159–160.

Kaufmann, P. (2013). *Adapting Hardware Systems by Means of Multi-Objective Evolution*. Logos Verlag Berlin GmbH.

Khan, S., H. Rahmani, S. A. A. Shah, and M. Bennamoun (2018). "A guide to convolutional neural networks for computer vision." In: *Synthesis Lectures on Computer Vision* 8.1, pp. 1–207.

Koza, J. R. (1989). "Hierarchical genetic algorithms operating on populations of computer programs." In: *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence IJCAI-89*. Ed. by N. S. Sridharan. Vol. 1. Detroit, MI, USA: Morgan Kaufmann, pp. 768–774. URL: http://www.genetic-programming.com/jkpdf/ijcai1989.pdf.

— (1992). *Genetic programming: on the programming of computers by means of natural selection*. Vol. 1. MIT press.

Koza, J. R., W. Langdon, R. Poli, and N. Mcphee (Jan. 2008). "Genetic Programming: An Introduction and Tutorial, with a Survey of Techniques and Applications." In: vol. 115, pp. 927–1028. DOI: 10.1007/978-3-540-78293-3_22.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems*, pp. 1097–1105.

Lapa, P., I. Gonçalves, L. Rundo, and M. Castelli (2019). "Semantic learning machine improves the CNN-based detection of prostate cancer in non-contrast-enhanced MRI." In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1837–1845.

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel (1989). "Backpropagation applied to handwritten zip code recognition." In: *Neural computation* 1.4, pp. 541–551.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

LeCun, Y., D Touresky, G Hinton, and T Sejnowski (1988). "A theoretical framework for back-propagation." In: *Proceedings of the 1988 connectionist models summer school*. Vol. 1. CMU, Pittsburgh, Pa: Morgan Kaufmann, pp. 21–28.

Lopez, M. M. and J. Kalita (2017). "Deep Learning applied to NLP." In: *arXiv preprint arXiv:1703.03091*.

Miikkulainen, R., J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, et al. (2019). "Evolving deep neural networks." In: *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, pp. 293–312.

Miller, G. F., P. M. Todd, and S. U. Hegde (1989). "Designing Neural Networks using Genetic Algorithms." In: *ICGA*. Vol. 89, pp. 379–384.

Miller, J. F. and S. L. Harding (2008). "Cartesian genetic programming." In: *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*, pp. 2701–2726.

Moraglio, A. (2007). "Towards a geometric unification of evolutionary algorithms." In:

Moraglio, A. and K. Krawiec (2015). "Semantic genetic programming." In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 603–627.

Moraglio, A., K. Krawiec, and C. G. Johnson (2012). "Geometric semantic genetic programming." In: *International Conference on Parallel Problem Solving from Nature*. Springer, pp. 21–31.

Moraglio, A. and A. Mambrini (2013). "Runtime analysis of mutation-based geometric semantic genetic programming for basis functions regression." In: *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pp. 989–996.

Rawat, W. and Z. Wang (2017). "Deep convolutional neural networks for image classification: A comprehensive review." In: *Neural computation* 29.9, pp. 2352–2449.

Real, E., S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin (2017). "Large-scale evolution of image classifiers." In: *arXiv preprint arXiv:1703.01041*.

Rebala, G., A. Ravi, and S. Churiwala (2019). *An introduction to machine learning*. Springer.

Rojas, R. (2013). *Neural networks: a systematic introduction*. Springer Science & Business Media.

Rosenblatt, F. (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Learning representations by back-propagating errors." In: *nature* 323.6088, pp. 533–536.

Schaller, R. R. (1997). "Moore's law: past, present and future." In: *IEEE spectrum* 34.6, pp. 52–59.

Sharma, P. and A. Singh (2017). "Era of deep neural networks: A review." In: *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, pp. 1–5.

Stanley, K. O., J. Clune, J. Lehman, and R. Miikkulainen (2019). "Designing neural networks through neuroevolution." In: *Nature Machine Intelligence* 1.1, pp. 24–35.

Stanley, K. O. and R. Miikkulainen (2002). "Evolving neural networks through augmenting topologies." In: *Evolutionary computation* 10.2, pp. 99–127.

Such, F. P., V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune (2017). "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning." In: *arXiv preprint arXiv:1712.06567*.

Suganuma, M., S. Shirakawa, and T. Nagao (2017). "A genetic programming approach to designing convolutional neural network architectures." In: *Proceedings of the genetic and evolutionary computation conference*, pp. 497–504.

Sun, Y., B. Xue, M. Zhang, and G. G. Yen (2019a). "Completely automated CNN architecture design based on blocks." In: *IEEE transactions on neural networks and learning systems* 31.4, pp. 1242–1254.

— (2019b). "Evolving deep convolutional neural networks for image classification." In: *IEEE Transactions on Evolutionary Computation* 24.2, pp. 394–407.

Szegedy, C., S. Ioffe, V. Vanhoucke, and A. A. Alemi (2017). "Inception-v4, inception-resnet and the impact of residual connections on learning." In: *Thirty-first AAAI conference on artificial intelligence*.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015). "Going deeper with convolutions." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Teixeira, M. J. S. (2019). "Improving malware detection with neuroevolution: a study with the semantic learning machine." Doctoral dissertation.

Vanneschi, L. (Aug. 2017). "An Introduction to Geometric Semantic Genetic Programming." In: 663, pp. 3–42. DOI: 10.1007/978-3-319-44003-3_1.

Vanneschi, L., M. Castelli, and S. Silva (June 2014). "A survey of semantic methods in genetic programming." In: *Genetic Programming and Evolvable Machines* 15. DOI: 10.1007/s10710-013-9210-0.

Vapnik, V. (1992). "Principles of risk minimization for learning theory." In: *Advances in neural information processing systems*, pp. 831–838.

Zhao, Q., S. Lyu, B. Zhang, and W. Feng (2018). "Multiactivation pooling method in convolutional neural networks for image recognition." In: *Wireless Communications and Mobile Computing* 2018.

Zhou, P., G. Zhou, Z. Zhu, C. Tang, Z. He, W. Li, and F. Jiang (Aug. 2018). "Health Monitoring for Balancing Tail Ropes of a Hoisting System Using a Convolutional Neural Network." In: *Applied Sciences* 8, p. 1346. DOI: 10.3390/app8081346.

Zhu, W., W. Yeh, J. Chen, D. Chen, A. Li, and Y. Lin (2019). "Evolutionary convolutional neural networks using abc." In: *Proceedings of the 2019 11th International Conference on Machine Learning and Computing*, pp. 156–162.