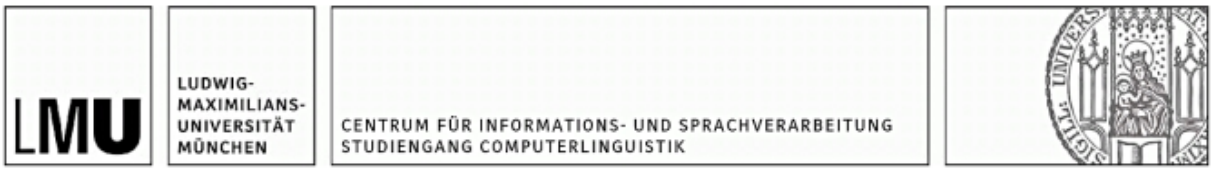

Bachelorarbeit

im Studiengang Computerlinguistik

an der Ludwig- Maximilians- Universität München

Fakultät für Sprach- und Literaturwissenschaften

Exploring Genetic Algorithms to optimize Convolutional Architectures for Slot Filling Tasks

vorgelegt von
Yannick Kaiser

Betreuer:	Nina Pörner
Prüfer:	Prof. Dr. Schütze
Bearbeitungszeitraum:	19. März - 28. Mai 2018

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, den 28. Mai 2018

.....
Yannick Kaiser

Abstract

Dieses Dokument dient als Muster für eine Ausarbeitung einer Bachelorarbeit am CIS und wird in deutscher oder englischer Sprache erstellt (hier max. 250 Wörter)

Inhaltsverzeichnis

1 Genetic Algorithms

The following chapter describes the idea of Genetic Algorithms. These algorithms base upon the idea of emulating the process that made complex life possible: Evolution managed to accomplish astonishing results where human systems often fail: In producing novel solutions to very high complexity problems. Genetic algorithms therefore are a method of searching the solution space for a or the optimal solution.

1.1 Evolution

"Living organisms are consummate problem solvers. They exhibit a versatility that puts the best computer programs to shame.", Holland begins his fundamental Paper on Genetic Algorithms [1].

In fact, the mechanisms of evolution have created the very species *Homo Sapiens*, that has created those computer programs in turn. It is therefore easy to see for any researcher, that in trying to understand and replicate these mechanisms, there is potential for novel insight in the realm of problem solving. And indeed, one of the first breakthrough applications of such knowdledge led to improvements in jet engine design, a very complex piece of machinery [1, p.72].

It is widely accepted that evolutionary processes use three mechanisms to accomplish design improvements: Selection, Crossover and Mutation.

The central role in all of this however falls to the genome, the DNA, a string of information that encodes the information necessary to build the phenotype life form. (Although newer research suggests that there are more sources of genetic information than the DNA alone, this simplification lends itself well for the purposes of evolutionary algorithms.) This distinction between encoded life form information and the actual life form lends itself to the mechanisms named above: Crossover and Mutation are easiely imagined when looking at strings of information, and selection is a mechanism purely affecting the phenotype life form.

Selection, Crossover and Mutation will be looked at in more detail in the sections to follow.

1.2 Evolutionary Algorithms

Deriving from the evolutionary processes are the class of Evolutionary Algorithms. The term Evolutionary Algorithms is a common denominator bundling Genetic Algorithms, Evolution Strategies, Particle Swarm Optimization, and Evolutionary Programming. [4,9]

Common to all of these approaches is the method of iteratively solving a problem by searching the solution space using a population of encoded candidate solutions, and then iteratively improving on them. Using the information gained from the population of candidate solutions to push the population towards the global maximum using combinations of candidate solutions and variations of individual solutions can be seen as emulating selection, crossover and mutation. [4, p.1]

1.3 The Genetic Algorithm

The Genetic Algorithm was first explored more deeply in the 1970s, mainly by John H. Holland [3, e.g.], and recieved more widespread recognition when in 1993 the first issue of the international journal on Evolutionary Computation [5] appeared. It has since then found many real-world applications due to its rubust ability in locating global optima in arbitrary and multimodal search spaces, which are common in engineering challenges such as neural network design or training, structure design or flow optimisation problems [7].

The Genetic Algorithm has some distinct advantages over more traditional methods:

- Probably the biggest asset of the Genetic Algorithm is its capablity of escaping local minima. It achieves this by conducting a random, but directed search, which is not aimed at the nearest local minimum like gradient descend would be [7]. To do so, a *population* of candidate solutions (called *genomes* or *individuals*) are kept, and follow up generations of candidate solutions are iteratively computed throgh means of selection, crossover and mutation.

- Candidate solutions are evaluated via their so called *Fitness* value, the result of a function that measures their ability in solving the problem. This function can provide absolute measures if the optimal solution is known (but the parameters producing it are not) or, it can provide relative Fitnesses by comparing the candidate solutions in a given population. In any case it is used to allow for a goal-oriented selection of candidate solutions to further process via selection and crossover.

- The genetic algorithm in its most basic form does not rely on derivatives of target functions, like gradient descend does. Therefore it has a good capability of handling search spaces where derivatives are hard to compute or even impossible to obtain. QUOTENEEDED

While the details on the individual operations will be covered in the following sections, the basic Genetic Algorithm can be postulated as follows:

```
Population ← generate random population;
Fitnesses ← evaluate(P);
while not max(Fitnesses) > k do
    MatingPool ← SelectionOperator(Population);
    Population ← CrossoverOperator(MatingPool);
    for Individual ∈ Population do
        Individual ← MutationOperator(Individual);
    end
    Fitnesses ← evaluate(P);
end
```

Algorithm 1: The Genetic Algorithm

After creating a randomized initial population of genomes, their Fitness values are evaluated by applying the Fitness function on all of them. From there on, the actual Genetic Algorithm can begin: While the solution is not yet found (i.e. the maximum attained fitness is below a certain cutoff value in case the maximum fitness is unknown) a follow-up generation is computed by means of the selection operator, crossover operator and mutation operator. The selection operator builds a mating pool of genomes to allow for selective crossover of genomes according to their Fitness values. Then the actual crossover happens, where a new population of genomes is built from the last. Lastly, mutation is applied to each genome in the new population.

1.4 Genome Encoding

Perhaps the most important component of a Genetic Algorithm is the so called *genome*. A genome is an encoded solution out of the solution space. While real-valued alleles are possible and actively being used [13, e.g.], the most basic representation would be a fixed-size binary genome, that is, a string of bits.

$$1011011010011001$$

Each bit or group of bits then encodes some characteristic of the encoded solution. A good example might be a geographic search for the lowest point in a confined area. Then, the above genome might encode x and y coordinates in the area, and the genome encodes one proposed solution by mapping directly onto one discrete point in the solution space.

$$\underbrace{10110110}_{x=182}\underbrace{10011001}_{y=153}$$

From there, the genome's Fitness value can be computed by getting the altitude of Point $(182, 153)$.

There are two ways to expand on this simple idea on genome encoding: One could allow real valued variables instead of bits, and/or one could allow for variable length genomes. For the sake of simplicity, all further examples will be using binary genome encoding.

Variable length encoding has one mayor concern, especially with respect to eventual crossover operations: How to decide which part of the genome is encoding which aspect of the phenotype? There have been various proposals for this [16, 17, e.g.], three of which will be talked about now.

Using index strings Index strings, as shown in [16], can be used in conjunction with the actual genome, to aid in decoding the genome information. The simplest version is a straightforward indexing of the genome positions:

Genome:	1	0	1	1	0	0	0	1	0	0
Index:	0	1	2	3	4	5	6	7	8	9

But these are unsuitable for variable length encoding. To allow for insertion or deletion of bits, the index string must be able to encode more fine grained information. A solution to this problem is usage real valued indexing values. This way, it is always possible to insert index values between existing values. This will prove very useful for variable length genome crossover.

Genome:	1	0	1	1	1	1	0	0	0	1	0	0
Index:	.0	.1	.2	.21	.22	.3	.4	.5	.6	.7	.8	.9

Crossover can then use the indexing information to avoid splitting up associated parts of the genome. In the above example, genome parts starting with the same decimal integer are seen as indivisable.

Visual Crossover Example??

Using identifier genes Another approach that has been explored is the insertion of regulatory sequences into genomes [17]. Here, a predefined string of bits marks the starting (and optionally ending) point of a sequence of information on a longer genome. This is modeled very much after the human DNA, where special sequences are used to determine wich parts of the DNA to read and decode.

Genomes containing identifier genes can be split into regions of continuous information via iteration over the genome string:

$$\overbrace{1\,1\,0\,0}^{identifier\ gene,\ (1^{st}\ attribute)} \quad \underbrace{0\,1\,1\,0\,0\,1\,1\,1}_{value,\ (1^{st}\ attr.)} \quad \overbrace{1\,1\,0\,1}^{id.\ gene,\ (2^{nd}\ attr.)} \quad \underbrace{0\,0\,1\,1}_{value,\ (2^{nd}\ attr.)}$$

For crossover the individual regions for each attribute can the be aligned to allow for controlled crossover operations without destroying the encoded structure.

Using complex data A third approach towards variable length genomes tries to circumvene the above issues by taking the easy route: Make the genome a complex object containing meta information alongside the actual genomic string. This is perhaps a less mathematically elegant solution towards genome design, but in the light of modern programming standards where object orientation is a fundamental aspect of many programming languages, this approach offers ease of implementation along with great flexibility.

Genome Class

attribute lengths : 4, 8, 4

genome string : $\underbrace{1011}_{4}\underbrace{00011011}_{8}\underbrace{0110}_{4}$

1.5 Crossover

With genome design established the next thing to look at in Genetic Algorithms is the crossover operation, as it is closely tied to genome design. The goal of crossover is interpolating new locations in the solution space by using two or more proposed solutions of the current generation as anchors [2].

Elitism!!

Single Point Crossover The most simple and most straightforward method for crossover is single point crossover [3, p.65]. As both parent genomes might contain parts of a better or optimal solution, it is desired to combine the information of these two to try and create an even better performing offspring. This is done by picking a random crossover point in the parent genomes on which to split the parent genome information.

The next illustration demonstrates the principle and uses a and b on the second parent to provide a better visual cue:

Parent 1: 1 1 1 0 1 1 | 1 0 1 0 0 0 0
Parent 2: $a\ a\ b\ a\ b\ a$ | $a\ b\ b\ a\ a\ a\ b$
Offspring 1: 1 1 1 0 1 1 $a\ b\ b\ a\ a\ a\ b$
Offspring 2: $a\ a\ b\ a\ b\ a$ 1 0 1 0 0 0 0

Multiple Point Crossover Multiple Point Crossover takes the idea of single point crossover one step further by allowing an arbitrary number k of crossover points. Akin to single point crossover, upon reaching any crossover point, the origin of the subsequent genome section switches between the parents.

$k = 3$
Parent 1: 1 1 | 1 0 1 | 1 1 0 1 0 0 0 | 0
Parent 2: $a\ a$ | $b\ a\ b$ | $a\ a\ b\ b\ a\ a\ a$ | b
Offspring 1: 1 1 b a b 1 1 0 1 0 0 0 b
Offspring 2: a a 1 0 1 a a b b a a a 0

Uniform Crossover Uniform crossover [18] is a random recombination method that takes multiple point crossover to its extreme and lets k equal the length of the genome, while copying the value of either parent with a probability of $p_{parent_1} = 0.5$, or with a probability that is proportional to the relative Fitness values of the parents: $p_{parent_1} = \frac{f_{parent_1}}{f_{parent_1}+f_{parent_2}}$. A second child can be created by inversing the decisions made for the first child.

$k = 3$
Parent 1: 1 1 1 0 1 1 1 0 1 0 0 0 0
Parent 2: *a a b a b a a b b a a a b*
Offspring 1: 1 a b 0 b 1 1 b 1 0 a 0 b
Offspring 1: a 1 1 a 1 a a 0 b a 0 a 0

Uniform Crossover has advantages and disadvantages: It is able to reach any recombination of the parent genome's bits with a single crossover operation, thus it is very versatile in searching the solution space. This however comes at the cost of destroying structures in the genome that might have already proven useful. Therefore random recombination is most useful when the individual values in the genome are largely independent from each other in regards to producing solutions.

Multi Parent Recombination Multi parent recombination, as discussed in Eiben et. al. (1994) [18], takes an arbitrary number of parents into account when computing an offspring.

multi parent crossover: Occurence based scanning, Fitness based probabilistic scanning [18]

mating pool

1.6 Selection

There are several ways to select wich individuals to crossover, as long as the basic heuristic of favouring better performing individuals is being kept. Goldberg et al. (1991) [6] compare a number of basic approaches to crossover selection and evaluates their performance tradeoffs regarding convergence time and other metrics.

The single most important denominator for selection decisions is the *Fitness* function $F(x)$, which evaluates a given individual (as encoded by its genome) regarding the problem it should solve. $F(x)$ therefore is the function that returns the surface altitude of the solution space on the point the individual represents. Sometimes the fitness function is called *objective function*, or, inversedly, *cost function*. As the value of $F(x)$ is highest at the optimal solution, is is preferable to mate individuals with high Fitness values to produce offsprings closer to the optimal solution.

Proportional Selection Proportional Selection, often nicknamed 'Roulette Selection' [10, 11], is a basic selection strategy that models the probability for any one individual x_k to be chosen for reproduction proportionally to its Fitness value $F(x_k)$. The basic formula would then look like this, where n is the number of individuals in the population:

$$p_k = \frac{F(x_k)}{\sum_{j=1}^{n} F(x_j)} \tag{1.1}$$

From the pool of individuals are then selected two individuals to mate, where the probability of choosing any one individual is weighted by the above formula. It is unusual to allow repetition in the selection process, but may be advantageous in certain cases.

Time complexity?

Ranking Selection The Idea of ranking based selection was introduced by Baker (1985) [8]. Before doing a weighted selection, a rank based assignment function is used to determine the weights to use. The mathematical explanation to follow is a simplified version from that presented in Goldberg et al. (1991) [6].

Let $x_1, ..., x_n$ be the sorted population of individuals, where $f_1, ..., f_n$ is their Fitness value $f_i = F(x_i)$ and $f_i >= f_{i+1}$ for all $i \in [1, n]$.

A probability distribution α is introduced, where for each x_i, $\alpha(i)$ is the probability for x_i to be picked. Using the probability distribution, ranking selection can then be done just like proportional selection via simple probabilistic selection.

To be suitable as a probability distribution, α however needs to complay to a set of rules:

1. $\alpha(i) >= 0$
2. $\alpha(i) is non - increasing$
3. $\sum_{i=1}^{n} \alpha(x_i) = 1$

It is easy to see that these constrains force α to work as a probability distribution, where a total probability of 1 is distributed among the individuals in the population.

The main advantage of ranked selection is a discoupling of the proportions of individuals in the population from the selection process, slowing the takeover of very fit individuals during subsequent generations [6, p.77].

Tournament Selection Tournament selection is a popular method of selecting individuals for mating, as it can effectively contain the selection process, preventing very fit individuals from taking over the entire population to easily. In fact, the selection pressure is proportional to the tournament size t [12, p.194], and can thus be controlled via adjusting the parameters of the tournament selection [12].

For tournament selection a set of t individuals is randomly chosen from the population, and the 'tournament winning' individual gets added to a mating pool. This is repeated until the mating pool is filled.

Winning a tournament can be deterministic. Then the winner is decided by

$$\arg\max_x F(x)$$

alternatively, the tournament can be probabilistic. Then a probability distribution is applied to the ordered list of individuals before choosing the winner.

The simplest case for this is when $t = 2$, then the individual with the higher Fitness score wins with a probability p, where

$$0.5 < p \leq 1.0$$

Larger Tournaments can be simulated by a bracketed knock-out system. In this case, successive rounds of binary ($t = 2$) tournaments are held, whereby the winners advance into the next round. Alternatively, akin to ranking selection, a probability distribution α is applied to all participating individuals after they have been sorted for their Fitness values. Let $x_1 ... x_t$ be the descendingly ordered participants in the tournament. Then the probability of winning the tournament for the i^{th} individual is

$$p_i = \alpha(i)$$

To achieve a probability distribution, α needs to meet the requirements postulated in the section on ranking selection:

1. $\alpha(i) >= 0$

2. $\alpha(i) is non-increasing$
3. $\sum_{i=1}^{t} \alpha(x_i) = 1$

TODO write smth!

"The convergence rate of a GA is largely determined by the selection pressure, with higher selection pressures resulting in higher convergence rates." [12, p.195] Pressure to high: higher chance of convergence on a suboptimal solution. Pressure to low: algorithm takes unneccessarily long to compute

1.7 Mutation

escaping local minima [16, 2] binary: switching vs random generation real valued/complex values length mutation

1.8 Evaluation

lorem ipsum

1.9 Past Success/Use cases?GA in NN?

lorem ipsum

Literaturverzeichnis

[1] Holland, J. H. (1992). Genetic algorithms. *Scientific american, 267*(1), 66-7.

[2] Haupt, R. L., & Haupt, S.E (1998). *Practical genetic algorithms* (Vol. 2). New York: Wiley.

[3] Holland, J. H. (1975). Adaptation in natural and artificial systems: an introductory analysis with applicatins to biology, control, and artificial intelligence.

[4] Schwefel, H. P., & Rudolph, G. (1995, June). Contemporary evulution strategies. In *European conference on artificial life* (pp.891-907). Springer, Berlin, Heidelberg.

[5] De Jong, K. (Ed.) (1993). Evolutionary computation (journal). MIT press, Cambridge MA.

[6] Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms* (Vol. 1, pp. 69-93). Elsevier.

[7] Srinivas, M., & Patnaik, L. M. (1994). Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, 24*(4), 656-667.

[8] Baker, J. E. (1985, July). Adaptive selection methods for genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and their applications* (pp. 101-111).

[9] Kennedy, J. (2011). Particle swarm optimizsation. In *Encyclopedia of machine learning* (pp. 760-766). Springer US.

[10] De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. (Doctoral dissertation, University of Michigan). *Dissertation Abstracts International, 36*(10), 5140B. (University Microfilms No. 76-9381).

[11] Goldberg, D. E. Genetic algorithms in search, optimization, and machine learning (1989). *New York, Adison-Wesley.*

[12] Miller, B. L.,& Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex systems, 9*(3), 193-212.

[13] Wright, A. H. (1991). Genetic algorithms for real parameter optimization. In *Foundations of genetic algorithms* (Vol. 1, pp. 205-218). Elsevier.

[14] Tur, G.,Hakkani-Tür, D., &Heck, L. (2010, December). What is left to be understood in ATIS?. In *Spoken Language Technology Workshop (SLT), 2010 IEEE* (pp. 19-24). IEEE.

[15] Mesnil, G., Dauphin, Y., Yao, K., Bengio, Y., Deng, L., Hakkani-Tur, D., ... & Zweig, G. (2015). Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing, 23*(3), 530-539.

[16] Lee, C. Y., & Antonsson, E. K. (2000, July). Variable Length Genomes for Evolutionary Algorithms. In *GECCO* (Vol. 2000, p.806).

[17] Mattiussi, C., & Floreano, D. (2007). Analog genetic encoding for the evolution of circuits and networks. *IEEE Transactions on evolutionary computation, 11*(5), 596-607.

[18] Eiben, A. E., Raue, P. E., & Ruttkay, Z. (1994, October). Genetic algorithms with multi-parent recombination. In *International Conference on Parallel Problem Solving from Nature* (pp. 78-87). Springer, Berlin, Heidelberg.

[19] Bonabeau, E., Dorigo, M., & Theraulaz, G. (2000). Inspiration for optimization from social insect behaviour. *Nature, 406*(6791), 39.

[20] Pearce, P. (1990). *Structure in Nature is a Strategy for Design*. MIT press.

[21] Lee, L. P., & Szema, R. (2005). Inspirations from biological optics for advanced photonic systems. *Science, 310*(5751), 1148-1150.

[22] Wang, Y. Y., Deng, L., & Acero, A. (2005). Spoken language understanding. *IEEE Signal Processing Magazine, 22*(5), 16-31.

[23] Mesnil, G., He, X., Deng, L., & Bengio, Y. (2013, August). Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *Interspeech* (pp. 3771-3775).

[24] MILLER, Geoffrey F.; TODD, Peter M.; HEGDE, Shailesh U. Designing Neural Networks using Genetic Algorithms. In: *ICGA*. 1989. S. 379-384.

Abbildungsverzeichnis

Tabellenverzeichnis

Inhalt der beigelegten CD