

OPTIMIZING THE STRUCTURES OF TRANSFORMER NEURAL NETWORKS USING PARALLEL SIMULATED ANNEALING

Maciej Trzciński^{2,3}, Szymon Łukasik^{2,3,4,*}, Amir H. Gandomi^{1,5}

¹*University of Technology Sydney, Faculty of Engineering and Information Technology, Ultimo, Sydney, NSW 2007, Australia*

²*AGH University of Krakow, Faculty of Physics and Applied Computer Science, Al. Mickiewicza 30, Krakow 30-059, Poland*

³*NASK National Research Institute, ul. Kolska 12, Warsaw 01-045, Poland*

⁴*Systems Research Institute, Polish Academy of Sciences, ul. Newelska 6, Warsaw 01-447, Poland*

⁵*University Research and Innovation Center (EKIK), Óbuda University, Bécsi út 96/B, Budapest 1034, Hungary*

*E-mail: slukasik@agh.edu.pl

Submitted: 9th January 2024; Accepted: 26th May 2024

Abstract

The Transformer is an important addition to the rapidly increasing list of different Artificial Neural Networks (ANNs) suited for extremely complex automation tasks. It has already gained the position of the tool of choice in automatic translation in many business solutions. In this paper, we present an automated approach to optimizing the Transformer structure based upon Simulated Annealing, an algorithm widely recognized for both its simplicity and usability in optimization tasks where the search space may be highly complex. The proposed method allows for the use of parallel computing and time-efficient optimization, thanks to modifying the structure while training the network rather than performing the two one after another. The algorithm presented does not reset the weights after changes in the transformer structure. Instead, it continues the training process to allow the results to be adapted without randomizing all the training parameters. The algorithm has shown a promising performance during experiments compared to traditional training methods without structural modifications. The solution has been released as open-source to facilitate further development and use by the machine learning community.

Keywords: neural architecture search, transformers, natural language processing, simulated annealing

1 Introduction

Transformers [1] have been proven to constitute a prominent step in various fields of Artificial Neu-

ral Network (ANN) research, such as Natural Language Processing (NLP) or Computer Vision (CV), even challenging the now long-established position

of Convolutional Neural Networks (CNN) in the latter. They are an important addition to the rapidly increasing list of different ANNs suited for extremely complex automation tasks and have already gained their niche in NLP. Originally proposed as a model strictly aimed at sequence-to-sequence modeling, the Transformer is now researched as a solution to many complex problems in different areas of science, including, but not limited to, audio processing [2], chemistry [3], medical imaging [4] and even life sciences [5]. Pre-trained Transformer models have already gained the position of the tool of choice in automatic translation and text generation in many business and entertainment solutions [6].

This type of network can be classified as being a complex ANN, as the number of hyperparameters (i.e. the parameters describing the structure, activation functions, or dimensions of internal feed-forward networks) of the transformer can reach hundreds of thousands. The original paper proposes different sets of hyperparameters unified between every structural component [1]. Usually, the problem of the Transformer structure is approached with regard to general, 'global' hyperparameters such that, for example, the number of attention heads shared in all the multiheaded attention blocks in the transformer. However, unifying hyperparameters between the blocks leads to little flexibility of the network design. Some generalizations of the transformer touched on this problem [7], but their effectiveness is still in question.

As many machine learning algorithms, not only ANNs, depend on their hyperparameters, different black-box optimization algorithms are being adapted for the needs of tuning them effectively [8]. Both nature-inspired and based on certain mathematical optimization techniques can be highlighted among them. Methods such as Bayesian optimization [9, 10] have been used successfully in this field; however, more straightforward methods, like random search, are still in use because of their universality [11].

The aim of this paper is to present an automated approach to optimize the Transformer structure based on Simulated Annealing, an algorithm widely recognized for its simplicity and usability in optimization tasks where the search space contains many local minima. The proposed method

allows using parallel computing and provides for time-efficient optimization, thanks to modifying the structure while training the network rather than performing the two subsequently. Although the field of NAS has been studied intensively, especially in recent years [12], the approach presented here is novel as the possibility of heuristic structural optimization of the Transformer during the training process has not been explored yet. The paper demonstrates that it is an attractive alternative to training a single Transformer or a set of architectures selected randomly for exactly the same number of epochs.

2 Methodological background

2.1 Transformer Networks

Transformer networks, proposed by Vaswani et al. [1] in 2017, quickly gained popularity in their field of deep learning. The central idea of the network is the utilization of the so-called attention mechanism, which allows one to process time-invariant dependencies of the input data series. Rather than processing each token one after another, all are handled in parallel. The key operation of the attention mechanism calculates the so-called 'attention matrix' where each token is assigned a value of importance in the context of other tokens. The attention is calculated using the equation:

$$A = \text{softmax}\left(\frac{KQ^T}{\sqrt{d}}\right) * V \quad (1)$$

where K and Q are obtained by multiplying the input by, respectively, Key K and Query Q matrices (which are trained as weights of neural networks) and d is the dimension of the key vector, which is usually the dimension of the embedding itself. The obtained attention A matrix is then multiplied by V , which is calculated similarly to K and Q . After this multiplication, the result is weighted by the attention scores between each token. To ensure that various features and dependencies may be learned, each time an attention mechanism is involved, several heads are utilized. Then their concatenated input is reduced back to the dimensions of the word embedding.

Transformers consist of two stacks: encoders and decoders. They function similarly but with a difference in their inputs and a slight discrepancy

in the attention mechanism. Each encoder consists of multiheaded self-attention that only performs attention over its input, concatenates it, and returns to the original embedding dimension using a feed-forward network. The result is then passed to the next encoder in the block, and so on. The decoder, however, has two multi-headed attentions, stacked one on another. The first is self-attention which functions just as in encoders. Encoder-decoded attention, however, uses Q obtained from the output of the self-attention and K and V that are the result of the encoder stack. Each encoder and decoder is also fitted with a skip connection that sums its input with its output and then performs a normalization on the result before providing its input to the next block in the stack.

Transformer networks have seen many structural developments, including generalizations such as universal transformers [7]. Primarily created to perform text-to-text operations, such as automatic translations or text generation, transformer architectures have proven to yield acceptable results in different fields, including computer vision. Both pure transformers and transformers with convolution were tested in recent years [13]. Versions of transformer aimed at specific tasks include ViT [14], iGPT [15], Image Transformer [16] and others. Another notable field in which Transformers can be used is the processing of time series, where Transformers may perform calculations such as forecasting, anomaly detection, or classification with great success [17].

2.2 Neural Architecture Search

Generally, every ANN conveys a function in the form of

$$f(P, w, x) = y \quad (2)$$

where P are the hyperparameters decided during the ANN initialization, w are the weights optimized during the training process, and x is an input. Although weight calculation methods are well covered in the literature [18] and many of them, such as ADAM, have their efficiency tested experimentally or mathematically [19], the area of optimizing hyperparameters is still developing and there is little consensus on the choice of methods to decide them [20]. The task of creating such methods is signifi-

cant in the context of ANNs development. Recent neural networks require a wide range of hyperparameters that have a considerable impact not only on the accuracy of the solutions but also on training time. However, the challenge of Neural Architecture Search is multi-fold: the search space is usually very complex and high-dimensional, mixing both continuous, discrete, and categorical variables, the evaluation of functions may be extremely expensive, and there is usually little information on the range of hyperparameters that should be considered and the bounds within which optimization should take place [8].

One of the core issues in NAS is related to the algorithm's search space. The set of all possible hyperparameter values is usually very difficult to grasp, and one should decide on how complex and detailed architectures to search for. The choice of search space is crucial to the process, as it directly influences the results that the network may obtain and can significantly impact the computational efficiency of the solution [21, 22, 23]. It is related to the choice of the encoding paradigm, as only the solution existing within the encoding scheme can appear as the final result of the algorithm [20]. Neural Architecture Search is typically employed with one of the following encoding schemes:

- The connection-based search space is initialized with a constant set of nodes and aims to optimize the connections between them, possibly by pruning unnecessary ones. This approach can be found, for example, in [24].
- The block-based [25] and cell-based [26] search space divides the network into uniform elements that share their hyperparameters and structure details with each other. The flexibility of this approach depends on how complex the elements are. For example, elements can be generalized to layers, blocks, or even larger structural pieces.
- The fixed-length solution encoding strategy limits its search space to a certain depth, but is generally easier to implement and generates less computational overhead. In the case of variable length encoding, the search space becomes more flexible, at the cost of complexity, efficiency, and simplicity of implementation [20].

2.3 Popular approaches in NAS

Different algorithms have been proposed in the field of NAS, including many heuristic methods. The variety of strategies can be divided into three general areas: reinforcement learning, heuristic search, and one-shot architecture search [27].

Reinforcement Learning

Reinforcement learning is a relatively well-known area of optimization methods and has already been examined in the field of NAS [28, 27]. The approach relies on an agent that performs actions in the environment and receives feedback as a numerical reward indicating how well the actions push it towards a specific goal. In reinforcement learning for NAS, actions can be chosen from a set of structural modifications. The state represents a particular architecture, and the environment is an optimal network design task for a chosen data set. Algorithms based on RL have shown some promise, but require careful consideration of conditions, update policy and may require powerful computing resources [28, 29].

Heuristic Algorithms

Different heuristics have been used successfully in the field of NAS. Notably, it includes evolutionary algorithms, a well-known class of black-box optimization methods that use several key components, including initialization, parent selection, recombination and mutation, and survivor selection based on objective function evaluation [30, 27]. In NAS this means pooling the area of architectures by applying random mutations that allow for very complex search spaces, but especially when the objective function is only reflecting the network accuracy, redundancies in its structures may occur [27]. Especially in the field of optimizing Convolutional Neural Networks, different genetic algorithms were used with promising results [31, 32]. Other evolutionary methods, such as particle swarm optimization, were also used to automatically select hyperparameters [33]. While evolutionary algorithms aim to discover the global extremes of a given function operating semi-randomly and applying most of its modifications randomly, which requires a considerable amount of computation time to re-evaluate the target function after minor changes in the model, surrogate optimization

takes advantage of a mathematical model that approximates the function between nodes within a certain error. This dramatically reduces the number of required evaluations, which is especially important for cost-intensive Neural Network training. However, the problem is that surrogate optimization usually works best for the continuous search area, while NAS search spaces usually mix discrete and continuous variables while also being highly constrained. Examples of surrogate optimization in NAS can be found in [34]. From other examples of heuristics used in NAS, Simulated Annealing will be distinguished and described in Section 2.8, as the one used in this paper.

One-shot Architecture Search

Several approaches that use only a single training cycle have been proposed. Although optimizing network structure during training is discussed in the further part of the article, it is worth observing that this is a relatively less explored area of NAS [35].

Other

It should be noted that there exist methods that utilize a mix of the aforementioned, for example, SASA (Surrogate Assisted Simulated Annealing) uses both Surrogate Optimization and Simulated Annealing [36]. For a more extensive overview of the area of neural architecture search, one could refer to surveys such as [34] and [20].

2.4 Transformers and NAS

There were already some architecture search methods proposed for transformer networks [37]. Classic optimization algorithms, such as genetic algorithm were proven to yield promising results, but their computational cost may exceed the benefits [38]. Often, NAS methods already used for other networks are utilized, such as Hardware-Aware Framework for Accelerating Neural Architecture Search Across Modalities [39] or One-Shot NAS Algorithm [21]. From solutions dedicated to transformers (rather than applied to them after being created for different networks) algorithms such as Evolved Transformer [40] or searching for Primer [41] can be noted as examples of dedicated solutions. Still, however, the field remains dynamic and different search spaces, approaches and algorithms are tested. Many challenges that the transformers

pose (such as a considerate training time) push towards more robust methods. Our algorithm, thanks to its single training process and intense utilization of parallel computing follows this trend.

2.5 Training ANNs during Structural Changes

Several approaches to hyperparameter optimization in ANNs have been proposed, but only a few of them consider changes in the structure of the network during a single training run. Auto-sizing is an example of such a method [24, 42]. It modifies the loss function so that the more weights are close to zero, the lower the loss. After training, the zero weights can simply be removed from the network. This approach, however, requires starting from a considerably large structure and only allows for reducing the size of the network and not expanding it. However, the need for mechanisms that allow reevaluating the structure of ANNs after minor changes without repeating the entire training process is increasing [43].

2.6 Neural Network Morphisms

The rise of network morphisms can be considered one of the most interesting recent developments in the field of NAS [20]. They are an example of a method that allows for the change of the structure of the Neural Network during training. The concept is based on the mathematical modification of the network blocks (such as additional networks, neurons, etc.) in such a way that they do not change the function that the Neural Network represents, i.e., the hyperparameters of the network are modified so that the output stays the same for every input, as presented in Equation 3.

$$\forall_x : f(P', w', x) = f(P, w, x) \quad (3)$$

Methods allowing for network morphisms were described in the net2net mechanism [44] and further researched and expanded [45]. The first morphisms consisted of two operators allowing for the widening of network layers and splitting them into equal-sized consecutive layers.

There are several advantages when considering this approach, including the ability to exactly preserve what was already trained, allowing for further training without any need for resetting the

weights of the network. The method also requires less CPU or GPU time. Recent developments in Neural Network morphisms have seen the expansion of the idea of network morphisms, resulting in different mechanisms such as Auto-Keras [46], or a method of optimization inspired by Lamarckian evolution [47].

2.7 Approximate Morphisms

Although the approach can be viewed as generally very efficient, creating a morphism is not always possible, especially when the functions conveyed by the network are very complex. Transformers can be an example of such networks, mainly because of their dimensionality operations and utilization of softmax functions that are difficult, if not impossible, to prepare in such a way that the input is preserved. Therefore, in this article, a new structure is generated in such a way that the function represented by the ANN is only similar to the function before modification (Equation 4). The first paper to introduce approximate morphisms was [47], where they were used in convolutional networks, where certain number of filters was removed at random. In the environment of transformers we propose morphisms that remove the blocks that have the lowest impact on the result (i.e. the lowest norm between input and output).

$$\forall_x : f(P', w', x) \approx f(P, w, x) \quad (4)$$

The way the operator of approximation is defined may vary between approaches and implementations, the most common way, however, is to use a certain norm and ensure that the output of the modified network is never more divergent than a certain threshold. This is presented in Equation 5, where L is the norm and ϵ is the threshold.

$$\forall_x : L(f(P', w', x), f(P, w, x)) \leq \epsilon \quad (5)$$

However, the use of strict norms and their calculation may require additional training and evaluation considerations, which reduces the efficiency of the algorithm. Transformers, however, have many built-in mechanisms of skip connections that ensure that some of the information omits a block, which opens possibilities for adding blocks while the previous parts of the stack still convey their outputs to the network output.

2.8 Simulated Annealing

Simulated annealing (SA) is an optimization algorithm with a wide variety of uses. It improves the random search by allowing new candidates worse than the current solution to be accepted under certain circumstances, allowing escape from local minima, especially during early iterations [48]. It is also easily parallelized [49]. One of the most important advantages of SA is the flexibility with regard to the neighbor operator used. In SA, the solutions may exist in arbitrary spaces that only require some kind of distance function, which does not need to be Euclidean.

In each step of the SA algorithm, new candidates to the solution are accepted or rejected based on a semi-random principle where a decreasing factor of temperature influences whether there is a chance for a solution worse than the previous one to be accepted nonetheless. The most common methods include variations of a formula that simulated the temperature mechanisms in real-life cooling of a material [50], as presented in so called Metropolis rule6, where P_{accept} is the probability of accepting a new candidate X rather than candidate $X_{previous}$ according to objective function f and temperature T .

$$P_{accept}(X) = \begin{cases} 1, & f(X) > f(X_{previous}) \\ e^{-\frac{f(X_{previous}) - f(X)}{T}}, & f(X) < f(X_{previous}) \end{cases} \quad (6)$$

There are several important decisions in the design of SA that impact its applicability and general efficiency in given problems.

Neighbor Generation

In every iteration of the algorithm, usually a neighbor or several neighbors (if parallel processing is being used) of the previous solution are generated using a certain neighbor operator. Depending on the task, this operator may randomly search the space within some neighborhood. However, it may get ineffective given that the number of dimensions of the problem can be very large and the procedure may require some adaptation. However, it was observed that in the case of problems with high dimensions, SA outperforms many different methods, including population-based methods, especially regarding memory efficiency [51]. Several improved neigh-

borhood operators have been proposed to search the continuous, multidimensional space. Adaptive neighborhoods, where the search algorithm uses the information obtained, have been also present for a long time [52]. One of the most notable examples is the dynamic nonuniform operator borrowed from evolutionary algorithms [53] that shrinks the radius of the search space over time, adding an additional layer of constricting large jump as the algorithm progresses. Other methods may be dependent on the nature of the problem; for example, SA used in the Traveling Salesman Problem creates neighboring solutions by swapping cities and utilizes distance measures such as the number of swaps [54].

Parallelization

Regardless of the used neighborhood operator, candidate solution can be generated and evaluated independently, as the algorithm is simple to parallelize over many threads. Parallel SA algorithms can vary slightly in how exactly the results are communicated between threads: they can be treated as separate chains that do not exchange any information at all (this approach is also called Multiple Runs, because there is no difference other than speed between optimizing several independent chains simultaneously and doing so one after another [55]). The results can also be exchanged after each iteration. after a fixed number of iterations or randomly [56].

2.9 Justification for the use of Simulated Annealing in Transformers NAS

In this work, several factors contributed to the choice of SA specifically. Due to the need of very specific neighbor operator, the elasticity in creating neighbors in SA was deemed very promising. Simplicity of parallelization and the universality of SA were two another issues considered and it was decided that those factors outweigh several more intricate optimization algorithm that would require an unreasonable increase in complexity and more constricted structure space searched.

Selection of Initial Parameters and Candidates

While the most obvious way to generate initial candidates (or candidates) in SA is simply to randomly generate them in the search space, they can be selected using more sophisticated methods such

as K-means clustering [57], as long as the search space allows this clustering algorithm to produce reasonable results.

2.10 Related Work

The attempts to use SA in the field of Neural Networks are now more than two decades old, and different applications of SA have been tried far early in the development of parameter optimization [58]. Although these attempts were mostly eclipsed by the rise of better optimization algorithms such as ADAM [59], SA still found its niche in NAS. Some experiments are already present [60], most of them, however, apply to well-known CNN networks and there is little or no research on the use of SA in transformers. The notable examples of the use of SA for other types of Neural Networks include optimizing Deep Neural Networks for predicting public transit flow [61] where a neighbor operator was used to change the number of neurons in hidden layers. The use of SA was estimated to improve the prediction time in the problem analyzed by around 15% in most cases. In another example, robust networks aimed at optimizing structures of high-performance magnets were decided using the Very Fast Simulated Annealing Algorithm [62, 63]. In other examples, SA was used in single-training hyperparameter tuning [64].

3 Proposed Approach

The algorithm introduced here for optimizing the architecture of Transformer Neural Network is meant to perform as many calculations as possible in parallel threads, thus increasing its efficiency. In each thread, a separate annealing process is performed for a certain number of iterations, after which all the threads communicate and continue from the best solution obtained. In each of the aforementioned threads, a search process inspired by SA is performed: in subsequent epochs, an operator preserves most of the weights in the network, thus preserving some of the knowledge obtained during the previous training epochs. The old and new candidates are then trained for a fixed number of iterations and their results are then compared. The old candidate is kept or discarded according to the SA acceptance rule considering the current temperature. The neighbors are created by

either adding, removing, or modifying a feature at the top or bottom of the stack of either encoders or decoders. Each time a new element is added, its structure is randomized so that different architectures can be created.

The neighbor operator randomly applies each of the following modifications. The probability of modifications is given as the algorithm parameter. The modifications are as follows:

1. creation of a new randomized encoder at the top or at the bottom of the stack;
2. creation of a new randomized decoder at the top or at the bottom of the stack;
3. deletion of an encoder from the top or from the bottom of the stack;
4. deletion of a decoder from the top or from the bottom of the stack;
5. creation of additional attention head in an encoder or decoder at the top or at the bottom of the stack;
6. deletion of a random attention head in an encoder or decoder at the top or at the bottom of the stack;
7. change in dimensions of the feed-forward network in an encoder or decoder at the top or at the bottom of the stack.

Each operation is performed without changing the trained parameters of the transformer to which it is applied. Thus the patterns learned in the process up to the moment of modification are being preserved. That is also why the operations are applied only to the top or bottom of the stacks, in order to keep the structure and possible patterns as intact as possible. The operator is presented in a schematic in Figure 1.

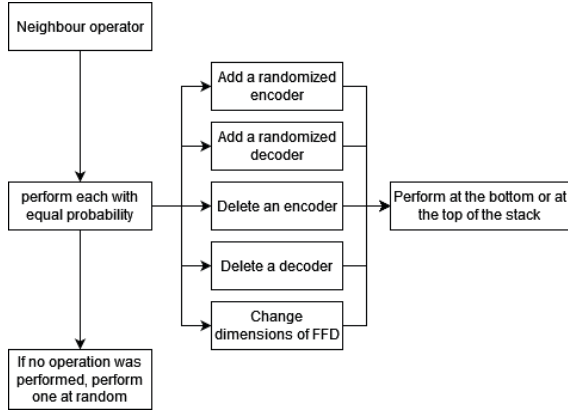


Figure 1. The schematic representation of neighbor operator

The general process is presented as Algorithm 1. In the algorithm, the parameters which need to be supplied are as follows: α is the temperature parameter of SA, P is the set of initial parameters of transformers, e is the number of epochs in which the candidates are trained before evaluating them, n_T is the number of threads, I is the number of annealing iterations, i_c is the number of iterations between connecting solutions from different threads, and $neighbor()$ is the function used to create neighbors of transformers using the method described above. The learning rate and learning decay (lr and ld respectively) are used in training the network. The optimization process is shown in Figure 2.

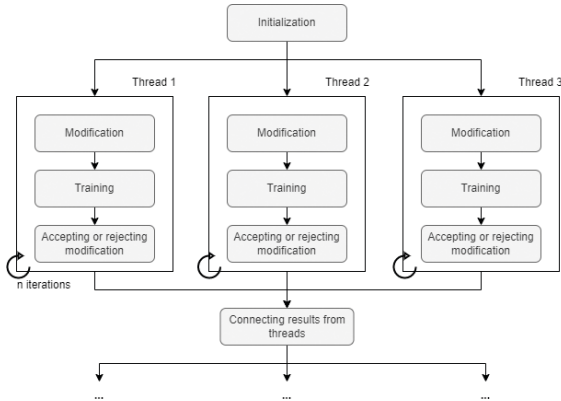


Figure 2. The graphical representation of the algorithm

The search space allowed by those operations is somehow similar to the ones proposed by earlier research, such as Autoformer [65], but rather than optimizing the parameters globally (i.e. adjusting certain dimensions simultaneously across all

blocks) it allows for more granular approach, with each encoder/decoder to possess different number of heads and the encoder/decoder stacks to vary in depth. This way, more complex structures may emerge from the search space, while still retaining the benefits of a relatively quick optimization.

3.1 Adding and Removing Randomized Blocks

Each time an encoder or decoder is added, the number of its heads is chosen randomly. Their weights are initialized according to the rules used in the training algorithm. An example of adding elements is shown in Figure 3, whereas an example of removing elements is shown in Figure 4.

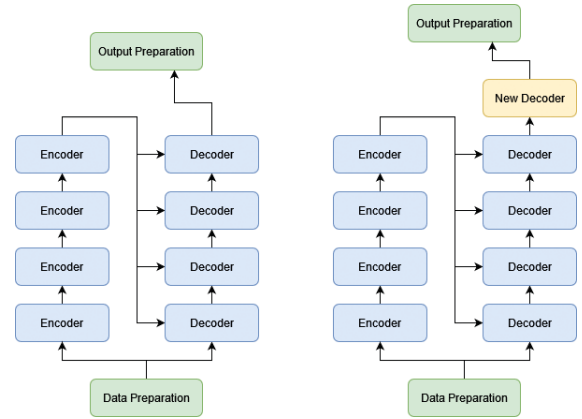


Figure 3. Adding elements to the transformer

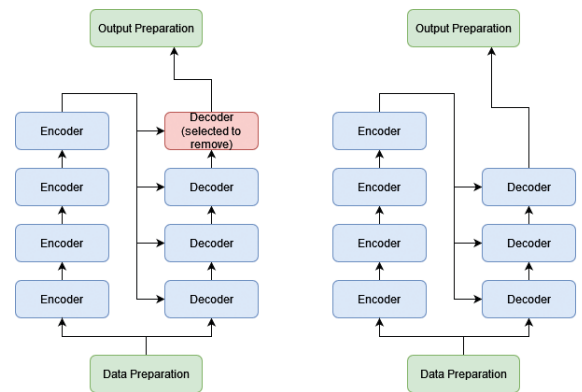


Figure 4. Removing elements from the transformer

3.2 Utilization of Skip-connections

Transformer networks utilize connections that skip the attention mechanisms and then apply addition and norm operation. This allows some infor-

mation about the result to be preserved even after adding a new block, thus allowing for safer random modifications of the stack.

3.3 Solution Representation

The solutions are stored as transformer objects together with their parameters. Neighborhood operators apply directly to the objects rather than to their simplified representations. This way weights are preserved and the structure may undergo flexible modifications.

3.4 Training Considerations

Because the structure of the network changes during the training, algorithms that accumulate previous steps are prone to error and require additional consideration for weights that were either non-existent or that were deleted. Therefore, for training the network the SGD algorithm was used, utilizing the momentum mechanism, which was proven to amend general efficiency [66].

3.5 Implementation Considerations

The algorithm was implemented with basic considerations of optimization, utilizing a simple mechanism aiming at improving its performance but not optimized for speed. The implementation was heavily using CUDA technology, which is widely used in almost every large-scale approach to ANNs nowadays. The working prototype code is available online at the open-source repository [67].

4 Experiments

To demonstrate the effectiveness of the proposed approach, experiments were carried out on different sets of parallel sentences in different languages used in the NLP domain. The task of the network was to perform automatic translation. The Transformers obtained by the algorithm were compared with two other approaches. In the first approach (*Original Transformer*), the initial architecture as presented in Table 2 was trained for the number of epochs equal to the number of Transformers n_T in the proposed algorithm times the average number of epochs performed, but without any modifications to their structure. In the second approach *Random search Set*, the n_T different architectures

were found with a random search. Transformers were created randomly by drawing each of their hyperparameters at random from the range of $[P_i, 6P_i]$ for every hyperparameter P_i as shown in Table 2. They were trained for the average epochs it took to train each transformer using SA modifications. The average was used here because of the training termination criterion involving the threshold for the change of training performance metric.

Algorithm 1 Parallel Simulated Annealing Hyperparameter Optimization

```

1: procedure ANNEALINGOPTIMIZE( $\alpha, P, e, n_T,$ 
    $I, i_c, neighbor(), lr, ld$ )
2:   init  $T$  as a list of  $n_T$  threads
3:   init  $S, C, Q, Q_{new}$  as an empty lists of size  $n_T$ 
   (Those are short for Solution, Candidate and
   Quality)
4:   for  $t$  in  $T$  do
5:      $transformer = neighbor(initFrom(P))$ 
6:      $Q[t] = transformer.quality()$ 
7:      $S[t] = transformer$ 
8:   end for
9:    $\tau = initTemperature(Q)$ 
10:  for  $i = 0$  to  $I$  do
11:    for  $t$  in  $T$  do
12:       $C[t] = NO(S[t])$ 
13:       $C[t].train(e, lr, ld)$ 
14:       $Q_{new}[t] = C[t].quality()$ 
15:       $S[t] = annealingRules(S[t], Q[t], C[t],$ 
         $Q_{new}, \tau)$ 
16:       $Q[t] = S[t].quality()$ 
17:    end for
18:    if  $i \bmod i_c$  then
19:       $bestS = S[indexOfMax(Q)]$ 
20:      for  $t$  in  $T$  do
21:         $S[t] = bestS$ 
22:      end for
23:    end if
24:     $\tau = \tau * \alpha$ 
25:  end for
26:  return  $S[indexOfMax(Q)]$ 
27: end procedure

```

4.1 Algorithm Parameters

The parameters of the algorithms (other than the initial transformer parameters discussed below) were decided during the pilot runs, as shown in Table 1.

Table 1. Parameters of the algorithm

Parameter	Value
α	0.9
e	10
n_T	8
I	5
i_c	1
lr	0.001
ld	0.9

The parameters demonstrated in the Table 1 were the same across all performed experiments – thus no fine-tuning was being made.

4.2 Initial Parameters of the Transformers

As the initial parameters of transformers, a minimal architecture was chosen in order to present the down-top approach and the ability to evolve structures from relatively simple and minimalist starting conditions, utilizing only a single encoder and decoder to check the ability of the algorithm to evolve them. The parameters are shown in Table 2.

Table 2. Initial parameters of the transformers

Parameter	Value
Embedding dimensions	1024
Query-key matrix dimension	32
Value matrix dimension	32
Feed-forward inner dimension	256
Number of heads	6
Number of encoders	1
Number of decoders	1
Dropout rate	0.1

4.3 Test Sets

The tests were performed on test sets known in the literature to serve as benchmarks for automatic translation. The sets are presented in Table 3.

The sets were chosen from different sources known in the literature.

Set 1

Set 1 was chosen from the parallel corpus created and curated by Internet users on the Tatoeba.org website. From the large set, 10,000 sentences were chosen randomly due to limitations in the computing power available. Different 1000

sentences were chosen for the network to be tested. The set consists mostly of simple, everyday expressions and repeatable sentences, utilizing many informal and slang expressions, as well as common idioms.

Set 2

Set 2 was chosen from the parallel corpus of documents in English and German published by the bureaus of the European Parliament. 10,000 sentences were used as a training set and 1000 as a test set. The set mostly consists of formal, structured language terms of the official European Parliament proceedings.

Set 3

For Set 3, a set available as part of the Flores 200 Project was chosen, using a special parallel corpus of English and Dutch sentences. The set of 1000 sentences was randomly split into 800 training data sentences and 200 testing data sentences. This set is prepared to test NLP capabilities when dealing with a minor data sample consisting of difficult and language-specific structures in the idea that 'no language is left behind' and the specific features of languages are explored, rather than just common terms.

4.4 Evaluation

The structures obtained were evaluated using BLEU and ROGUE metrics, commonly used in the evaluation of automatic machine translation [71, 72] and compared with the scores obtained in the same conditions by two other approaches. The tests were performed on a dedicated GPU architecture.

BLEU Metric

BLEU (BiLingual Evaluation Understudy) evaluates the quality of a translation by comparing it with a set of translations prepared by human translators. It compares a sentence translated by an algorithm with a set of translations provided by an external source. The score considers all n-grams to be coherent between the input and the target data or the number of n-grams of the target reproduced in the input in their exact form. It also penalizes brevity, i.e. reproduced instances of 1-grams, thus awarding the translations that have longer coher-

Table 3. Datasets used in the experiments

Dataset	Translation	Training sentences	Test sentences	Source
1	English-German	10,000	1000	[68]
2	English-German	10,000	1000	[69]
3	English-Danish	800	200	[70]

ent sequences, which is especially important in languages where grammar concepts like sentence order are significant to the meaning. The formula to calculate BLEU was presented as Equation 7.

$$BLEU(X, Y) = BP(X, Y) * \exp\left(\sum_{n=1}^{\infty} w_n \ln(p_n(X, Y))\right) \quad (7)$$

In Equation 7 BP is the Brevity Penalty calculated as $BP(X, Y) = e^{-\max(0, \frac{r}{c}-1)}$, where r is the effective reference corpus length, while c is the length of the candidate corpus. In the second part of the equation, w_n is the weight (the original paper proposes to set w_1 to w_4 as $\frac{1}{4}$, and $p_n(X, Y)$ is the modified n -gram precision function presented in equation 8, where $X = x^1, x^2, \dots, x^M$ is the candidate corpus, $Y = Y_1, Y_2, \dots, Y_M, Y_i = y^{i,1}, y^{i,2}, \dots, y^{i,N_i}$ is the correct translation corpus, G_n is the set of n -grams, and C is the substring count function.

$$p_n(X, Y) = \frac{\sum_{i=1}^M \sum_{r \in G_n(x^i)} \min(C(r, x^i), \max_{y \in Y_i} C(r, y))}{\sum_{i=1}^M \sum_{r \in G_n(x^i)} C(r, x^i)} \quad (8)$$

Although the BLEU score was criticized for limited comparison possibilities between corpora and even algorithms [73], it still remains a tool of choice in many NLP experiments and has been shown to be well correlated with human judgment [74].

ROGUE Metric

ROGUE (Recall-Oriented Understudy for Gisting Evaluation) was developed as an alternative to BLEU and as a tool for a more comprehensive comparison of translations. While BLEU awards fidelity and precision, ROGUE focuses on recall and thus is better at evaluating the grammar, coherence, and readability of the translated content. There are several versions of ROGUE and it has even seen its variations such as ROGUE 2.0 [75] that are relatively new. ROGUE uses precision and recall as the basis for its scores. Recall is calculated as shown in Equation 9, where N counts the number of n -grams

in the specified set [72]

$$Recall_n = \frac{N(X+Y)}{N(Y)} \quad (9)$$

Precision is calculated as shown in Equation 10.

$$Precision_n = \frac{N(X+Y)}{N(model)} \quad (10)$$

In the paper, for unigrams (1-grams) precision is provided together with F-Measure, calculated as the ratio of sum of precision and recall and their product. In the experiments, another metric was also used, ROGUE-L Sum Recall which uses the longest common sequences (hence 'L') rather than N -grams.

4.5 Results

The results are presented in Tables 4, 5, 6 and 7. Each result was obtained by calculating the average of three independent runs. The final column informs of the significance of the results – evaluated with pairwise t-tests. '0' denotes that the difference between the result presented in the row and the result obtained by the proposed algorithm is insignificant, and '-' that the difference is significant and points to the advantage of the proposed algorithm.

Table 4. BLEU metrics of compared algorithms

Algorithm	Result	Sign.
Set 1		
Original Transformer	0.0740 ± 0.0051	—
Random Search	0.1366 ± 0.0103	0
Proposed algorithm	0.1313 ± 0.0150	
Set 2		
Original Transformer	0.0450 ± 0.0034	—
Random Search	0.0170 ± 0.0062	—
Proposed algorithm	0.0533 ± 0.0099	
Set 3		
Original Transformer	0.0064 ± 0.0006	—
Random Search	0.0394 ± 0.0002	0
Proposed algorithm	0.0418 ± 0.0014	

Table 5. ROGUE1 FMEASURE metrics of compared algorithms

Algorithm	Result	Sign.
Set 1		
Original Transformer	0.1507 ± 0.0021	—
Random Search	0.2307 ± 0.0185	0
Proposed algorithm	0.2143 ± 0.0070	
Set 2		
Original Transformer	0.0614 ± 0.0043	—
Random Search	0.0129 ± 0.0032	—
Proposed algorithm	0.0673 ± 0.0070	
Set 3		
Original Transformer	0.0260 ± 0.0051	—
Random Search	0.0480 ± 0.0076	0
Proposed algorithm	0.0506 ± 0.0048	

Table 6. ROGUE1 PRECISION metrics of compared algorithms

Algorithm	Result	Sign.
Set 1		
Original Transformer	0.5133 ± 0.0055	—
Random Search	0.7197 ± 0.0156	0
Proposed algorithm	0.7280 ± 0.0381	
Set 2		
Original Transformer	0.0450 ± 0.0298	—
Random Search	0.1853 ± 0.0718	—
Proposed algorithm	0.6857 ± 0.0047	
Set 3		
Original Transformer	0.3056 ± 0.0921	—
Random Search	0.5793 ± 0.0030	0
Proposed algorithm	0.583 ± 0.0384	

Table 7. ROGUEL SUM RECALL metrics of compared algorithms

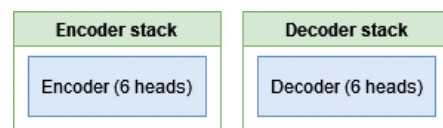
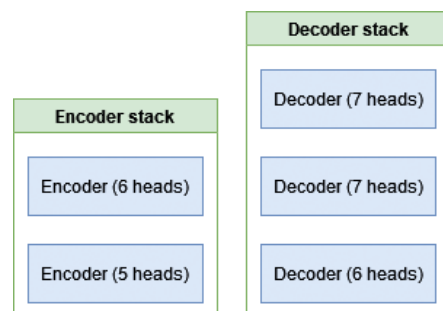
Algorithm	Result	Sign.
Set 1		
Original Transformer	0.0886 ± 0.0034	—
Random Search	0.1329 ± 0.0075	0
Proposed algorithm	0.1262 ± 0.0065	
Set 2		
Original Transformer	0.0319 ± 0.0021	—
Random Search	0.0053 ± 0.0003	—
Proposed algorithm	0.0369 ± 0.0039	
Set 3		
Original Transformer	0.0256 ± 0.0040	0
Random Search	0.0254 ± 0.0022	—
Proposed algorithm	0.0261 ± 0.0045	

4.6 Discussion

It can be seen that the comparison between the proposed algorithm and the classic approaches with training without structure modifications and random search was in favor of the Parallel Simulated Annealing performed during training. Though some results failed to provide statistical significance, this can be attributed to rather minuscule sample, caused by computational constraints. The results were behind the state-of-the-art architectures because of the limited complexity of the networks considered. It can be also noticed that the BLEU score was not always satisfactory and comparative, which can be attributed to the complexity of sentences, especially in the Flores set. The proposed approach is better or the same, in terms of considered performance metrics, for 11 out of 12 cases – for the comparison with the original Transformers using the same computational budget, and similarly for 11 out of 12 cases, when compared with random creation of Transformers, trained for the similar number of epochs.

As for the analysis of obtained architectures, it was noted that in most of the created architectures, there were more encoders evolved than decoders. In addition, a tendency was observed for more attention heads to be created at the top of the stacks rather than at the bottom.

An example evolution of a transformer structure was shown in Figure 5 and 6.

**Figure 5.** Initial transformer structure**Figure 6.** Final structure obtained by the algorithm for Set 1

5 Conclusion

In this paper, a novel approach to structure modifications in Transformers was presented. It was not tested before, and it shows promising results when compared to other methods in similar environments. The method reduces the time and resources needed to obtain new architectures compared to algorithms that require the network to be trained from scratch. The results obtained during the experiments demonstrate that the approach may serve as an important tool for structural optimization and as a basis for further study. The algorithm may be developed further by considering additional operators of neighborhood generation or the introduction of multiobjective optimization considering e.g. a factor of computational time. In this way, this research could contribute to the trend of developing more environmentally sustainable computational algorithms and adjusting resources to the task at hand. The code of the approach being worked out here was released as open source to facilitate its development and to attract the interest of the NAS community.

Acknowledgements

Research project was supported by program „Excellence initiative – research university” for the AGH University of Krakow. We gratefully acknowledge Polish high-performance computing infrastructure PLGrid (HPC Center: ACK Cyfronet AGH) for providing computer facilities and support within computational grant no. PLG/2024/017142.

References

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, <https://arxiv.org/abs/1706.03762> (2017). <https://doi.org/10.48550/ARXIV.1706.03762>.
- [2] N. Li, S. Liu, Y. Liu, S. Zhao, M. Liu, Neural speech synthesis with transformer network, in: Proceedings of the AAAI conference on artificial intelligence, Vol. 33, 2019, pp. 6706–6713.
- [3] P. Morris, R. St. Clair, W. E. Hahn, E. Barenholtz, Predicting binding from screening assays with transformer network embeddings, *Journal of Chemical Information and Modeling* 60 (9) (2020) 4191–4199.
- [4] F. Shamshad, S. Khan, S. W. Zamir, M. H. Khan, M. Hayat, F. S. Khan, H. Fu, Transformers in medical imaging: A survey, <https://arxiv.org/abs/2201.09873> (2022). <https://doi.org/10.48550/ARXIV.2201.09873>.
- [5] T. Lin, Y. Wang, X. Liu, X. Qiu, A survey of transformers (2021). <https://doi.org/10.48550/ARXIV.2106.04554>.
- [6] M. Zhang, J. Li, A commentary of gpt-3 in mit technology review 2021, *Fundamental Research* 1 (6) (2021) 831–833.
- [7] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, L. Kaiser, Universal transformers, *arXiv preprint arXiv:1807.03819* (2018).
- [8] M. Feurer, F. Hutter, Hyperparameter optimization, in: *Automated machine learning*, Springer, Cham, 2019, pp. 3–33.
- [9] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, S.-H. Deng, Hyperparameter optimization for machine learning models based on bayesian optimization, *Journal of Electronic Science and Technology* 17 (1) (2019) 26–40.
- [10] R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, I. Guyon, Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020, in: *NeurIPS 2020 Competition and Demonstration Track*, PMLR, 2021, pp. 3–26.
- [11] R. G. Mantovani, A. L. Rossi, J. Vanschoren, B. Bischl, A. C. De Carvalho, Effectiveness of random search in svm hyper-parameter tuning, in: *2015 International Joint Conference on Neural Networks (IJCNN)*, Ieee, 2015, pp. 1–8.
- [12] X. He, K. Zhao, X. Chu, Automl: A survey of the state-of-the-art, *Knowledge-Based Systems* 212 (2021) 106622.
- [13] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu, et al., A survey on vision transformer, *IEEE transactions on pattern analysis and machine intelligence* 45 (1) (2022) 87–110.
- [14] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al., An image is worth 16x16 words: Transformers for image recognition at scale, *arXiv preprint arXiv:2010.11929* (2020).
- [15] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, D. Luan, I. Sutskever, Generative pretraining from pixels, in: *International conference on machine learning*, PMLR, 2020, pp. 1691–1703.

- [16] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, D. Tran, Image transformer, in: International conference on machine learning, PMLR, 2018, pp. 4055–4064.
- [17] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, L. Sun, Transformers in time series: A survey, arXiv preprint arXiv:2202.07125 (2022).
- [18] E. Dogo, O. Afolabi, N. Nwulu, B. Twala, C. Aigbavboa, A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks, in: 2018 international conference on computational techniques, electronics and mechanical systems (CTEMS), IEEE, 2018, pp. 92–99.
- [19] S. J. Reddi, S. Kale, S. Kumar, On the convergence of adam and beyond, arXiv preprint arXiv:1904.09237 (2019).
- [20] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, K. C. Tan, A survey on evolutionary neural architecture search, IEEE transactions on neural networks and learning systems (2021).
- [21] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, Q. Le, Understanding and simplifying one-shot architecture search, in: International conference on machine learning, PMLR, 2018, pp. 550–559.
- [22] J. Fang, Y. Sun, Q. Zhang, Y. Li, W. Liu, X. Wang, Densely connected search space for more flexible neural architecture search, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 10628–10637.
- [23] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, X. Wang, A comprehensive survey of neural architecture search: Challenges and solutions, ACM Computing Surveys (CSUR) 54 (4) (2021) 1–34.
- [24] K. Murray, J. Kinnison, T. Q. Nguyen, W. Scheirer, D. Chiang, Auto-sizing the transformer network: Improving speed, efficiency, and performance for low-resource machine translation, <https://arxiv.org/abs/1910.06717> (2019), <http://arxiv.org/abs/1910.06717>.
- [25] M. Baldeon-Calisto, S. K. Lai-Yuen, Adaresu-net: Multiobjective adaptive convolutional neural network for medical image segmentation, Neurocomputing 392 (2020) 325–340.
- [26] K. Chen, W. Pang, Immunetnas: An immune-network approach for searching convolutional neural network architectures, arXiv preprint arXiv:2002.12704 (2020).
- [27] M. Wistuba, A. Rawat, T. Pedapati, A survey on neural architecture search, arXiv preprint arXiv:1905.01392 (2019).
- [28] J. G. Robles, J. Vanschoren, Learning to reinforcement learn for neural architecture search, arXiv preprint arXiv:1911.03769 (2019).
- [29] A. Vahdat, A. Mallya, M.-Y. Liu, J. Kautz, Unas: Differentiable architecture search meets reinforcement learning, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 11266–11275.
- [30] K. De Jong, Evolutionary computation: a unified approach, in: Proceedings of the 2016 on genetic and evolutionary computation conference companion, 2016, pp. 185–199.
- [31] S. Gibb, H. M. La, S. Louis, A genetic algorithm for convolutional network structure optimization for concrete crack detection, in: 2018 IEEE congress on evolutionary computation (CEC), IEEE, 2018, pp. 1–8.
- [32] L. Xie, A. Yuille, Genetic cnn, in: Proceedings of the IEEE international conference on computer vision, 2017, pp. 1379–1388.
- [33] F. Ye, Particle swarm optimization-based automatic parameter selection for deep neural networks and its applications in large-scale and high-dimensional data, PloS one 12 (12) (2017) e0188746.
- [34] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, E. Xing, <https://arxiv.org/abs/1802.07191>, Neural architecture search with bayesian optimisation and optimal transport (2018), <https://doi.org/10.48550/ARXIV.1802.07191>, <https://arxiv.org/abs/1802.07191>.
- [35] F. P. Casale, J. Gordon, N. Fusi, Probabilistic neural architecture search, arXiv preprint arXiv:1902.05116 (2019).
- [36] H. K. Singh, T. Ray, W. Smith, Surrogate assisted simulated annealing (SASA) for constrained multi-objective optimization, in: IEEE congress on evolutionary computation, IEEE, 2010, pp. 1–8.
- [37] K. T. Chitty-Venkata, M. Emani, V. Vishwanath, A. K. Somani, Neural architecture search for transformers: A survey, IEEE Access 10 (2022) 108374–108412, <https://doi.org/10.1109/ACCESS.2022.3212767>.
- [38] J. Kim, J. Wang, S. Kim, Y. Lee, Evolved speech-transformer: Applying neural architecture search to end-to-end automatic speech recognition., in: Interspeech, 2020, pp. 1788–1792.

- [39] D. Cummings, A. Sarah, S. N. Sridhar, M. Szankin, J. P. Munoz, S. Sundaresan, A hardware-aware framework for accelerating neural architecture search across modalities (2022), <http://arxiv.org/abs/2205.10358>.
- [40] D. So, Q. Le, C. Liang, The evolved transformer, in: International conference on machine learning, PMLR, 2019, pp. 5877–5886.
- [41] D. So, W. Mañke, H. Liu, Z. Dai, N. Shazeer, Q. V. Le, Searching for efficient transformers for language modeling, *Advances in neural information processing systems* 34 (2021) 6010–6022.
- [42] V. Cahlik, P. Kordik, M. Cepek, Adapting the size of artificial neural networks using dynamic auto-sizing, in: 2022 IEEE 17th International Conference on Computer Sciences and Information Technologies (CSIT), IEEE, 2022, pp. 592–596.
- [43] B. Goldberger, G. Katz, Y. Adi, J. Keshet, Minimal modifications of deep neural networks using verification., in: LPAR, Vol. 2020, 2020, p. 23.
- [44] T. Chen, I. Goodfellow, J. Shlens, Net2net: Accelerating learning via knowledge transfer, *arXiv preprint arXiv:1511.05641* (2015).
- [45] T. Wei, C. Wang, Y. Rui, C. W. Chen, Network morphism, in: International conference on machine learning, PMLR, 2016, pp. 564–572.
- [46] H. Jin, Q. Song, X. Hu, Auto-keras: An efficient neural architecture search system, in: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, 2019, pp. 1946–1956.
- [47] T. Elsken, J. H. Metzen, F. Hutter, <https://openreview.net/forum?id=ByME42AqK7> Efficient multi-objective neural architecture search via lamarckian evolution (2019), <https://openreview.net/forum?id=ByME42AqK7>
- [48] P. J. Van Laarhoven, E. H. Aarts, Simulated annealing, in: *Simulated annealing: Theory and applications*, Springer, 1987, pp. 7–15.
- [49] D. J. Ram, T. Sreenivas, K. G. Subramaniam, Parallel simulated annealing algorithms, *Journal of parallel and distributed computing* 37 (2) (1996) 207–212, <https://doi.org/https://doi.org/10.1006/jpdc.1996.0121>.
- [50] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, Equation of state calculations by fast computing machines, *The journal of chemical physics* 21 (6) (1953) 1087–1092.
- [51] D. Delahaye, S. Chaimatanan, M. Mongeau, Simulated annealing: From basics to applications, *Handbook of metaheuristics* (2019) 1–35.
- [52] Z. Xinchao, Simulated annealing algorithm with adaptive neighborhood, *Applied Soft Computing* 11 (2) (2011) 1827–1836.
- [53] Z. Michalewicz, Z. Michalewicz, *GAs: Why Do They Work?*, Springer, 1996.
- [54] S.-H. Zhan, J. Lin, Z.-J. Zhang, Y.-W. Zhong, List-based simulated annealing algorithm for traveling salesman problem, *Computational intelligence and neuroscience* 2016 (2016).
- [55] M. E. Aydin, V. Yigit, 12 parallel simulated annealing, *Parallel Metaheuristics: A new Class of Algorithms* (2005) 267.
- [56] L. Ozdamar, Parallel simulated annealing algorithms in global optimization, *Journal of global optimization* 19 (2001) 27–50.
- [57] G. P. Babu, M. N. Murty, Simulated annealing for selecting optimal initial seeds in the k-means algorithm, *Indian Journal of Pure and Applied Mathematics* 25 (1-2) (1994) 85–94.
- [58] R. S. Sexton, R. E. Dorsey, J. D. Johnson, Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing, *European Journal of Operational Research* 114 (3) (1999) 589–601.
- [59] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, *arXiv preprint arXiv:1412.6980* (2014).
- [60] S. Mo, J. Xia, P. Ren, Simulated annealing for neural architecture search, *Advances in Neural Information Processing Systems (NeurIPS)* (2021).
- [61] C.-W. Tsai, C.-H. Hsia, S.-J. Yang, S.-J. Liu, Z.-Y. Fang, Optimizing hyperparameters of deep learning in predicting bus passengers based on simulated annealing, *Applied soft computing* 88 (2020) 106068.
- [62] H.-K. Park, J.-H. Lee, J. Lee, S.-K. Kim, Optimizing machine learning models for granular ndfeb magnets by very fast simulated annealing, *Scientific Reports* 11 (1) (2021) 3792.
- [63] L. Ingber, Very fast simulated re-annealing, *Mathematical and computer modelling* 12 (8) (1989) 967–973.
- [64] M. Fischetti, M. Stringher, <https://arxiv.org/abs/1906.01504>, Embedded hyper-parameter tuning by simulated annealing (2019), <https://doi.org/10.48550/ARXIV.1906.01504>, <https://arxiv.org/abs/1906.01504>.
- [65] M. Chen, H. Peng, J. Fu, H. Ling, Autoformer: Searching transformers for visual recognition, in: *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 12270–12280.

- [66] A. Cutkosky, H. Mehta, Momentum improves normalized sgd, in: International conference on machine learning, PMLR, 2020, pp. 2260–2268.
- [67] M. Trzciński, Optimizing the Structures of Transformer Neural Networks using Parallel Simulated Annealing (3 2023).
- [68] F. Guzmán, P.-J. Chen, M. Ott, J. Pino, G. Lample, P. Koehn, V. Chaudhary, M. Ranzato, Two new evaluation datasets for low-resource machine translation: Nepali-english and sinhala-english, 2019.
- [69] P. Koehn, <https://aclanthology.org/2005.mtsummit-papers.11>, Europarl: A parallel corpus for statistical machine translation, in: Proceedings of Machine Translation Summit X: Papers, Phuket, Thailand, 2005, pp. 79–86. <https://aclanthology.org/2005.mtsummit-papers.11>.
- [70] J. Tiedemann, Parallel data, tools and interfaces in opus., in: Lrec, Vol. 2012, Citeseer, 2012, pp. 2214–2218.
- [71] K. Papineni, S. Roukos, T. Ward, W.-J. Zhu, Bleu: a method for automatic evaluation of machine translation, in: Proceedings of the 40th annual meeting of the Association for Computational Linguistics, 2002, pp. 311–318.
- [72] C.-Y. Lin, Rouge: A package for automatic evaluation of summaries, in: Text summarization branches out, 2004, pp. 74–81.
- [73] M. Post, A call for clarity in reporting bleu scores, arXiv preprint arXiv:1804.08771 (2018).
- [74] D. Coughlin, Correlating automated and human assessments of machine translation quality, in: Proceedings of Machine Translation Summit IX: Papers, 2003.
- [75] K. Ganesan, Rouge 2.0: Updated and improved measures for evaluation of summarization tasks, arXiv preprint arXiv:1803.01937 (2018).



Maciej Trzciński received Master's Degree in applied computer science from AGH University of Krakow in 2022. He is currently pursuing Ph.D. at AGH University in the field of information and communication technology. His current research interests include neural architecture search and natural language processing.

<https://orcid.org/0000-0003-4139-4969>



Szymon Łukasik is an associate professor at the AGH University of Kraków and the Systems Research Institute of the Polish Academy of Sciences, and the head of AI at NASK National Research Institute. Ph.D. and habilitation in the field of data analysis and computational intelligence. A graduate of the Top 500 Innovators

program at the Haas School of Business at the University of California, Berkeley, visiting scientist at UNINOVA (Portugal), National Laboratory of Pattern Recognition (China) and Australia National University. Author of almost 100 publications. His main area of research interests cover nature inspired metaheuristics and multimodal deep learning.

<https://orcid.org/0000-0001-6716-610X>



Amir H. Gandomi is a Professor of Data Science and an ARC DECRA Fellow at the Faculty of Engineering & Information Technology, University of Technology Sydney. He is also affiliated with Obuda University, Budapest, as a Distinguished Professor. Prof. Gandomi has published over three hundred journal papers and 12

books which collectively have been cited 50,000+ times (H-index: 100+). He has been named as one of the most influential scientific minds and received the Highly Cited Researcher award (top 1% publications and 0.1% researchers) from Web of Science for six consecutive years. He has served as associate editor, editor, and guest editor in several prestigious journals such as AE of IEEE TBD and IEEE IoTJ. His research interests are global optimisation and (big) data analytics using machine learning and evolutionary computations in particular.

<https://orcid.org/0000-0002-2798-0104>